Adaptive learning by a target-tracking system

Daniel Lockery and James F. Peters

Electrical and Computer Engineering, University of Manitoba, Winnipeg, Canada

Abstract

Purpose – The purpose of this paper is to report upon research into developing a biologically inspired target-tracking system (TTS) capable of acquiring quality images of a known target type for a robotic inspection application.

Design/methodology/approach – The approach used in the design of the TTS hearkens back to the work on adaptive learning by Oliver Selfridge and Chris J.C.H. Watkins and the work on the classification of objects by Zdzislaw Pawlak during the 1980s in an approximation space-based form of feedback during learning. Also, during the 1980s, it was Ewa Orlowska who called attention to the importance of approximation spaces as a formal counterpart of perception. This insight by Orlowska has been important in working toward a new form of adaptive learning useful in controlling the behaviour of machines to accomplish system goals. The adaptive learning algorithms presented in this paper are strictly temporal difference methods, including Q-learning, sarsa, and the actor-critic method. Learning itself is considered episodic. During each episode, the equivalent of a Tinbergen-like ethogram is constructed. Such an ethogram provides a basis for the construction of an approximation space at the end of each episode. The combination of episodic ethograms and approximation spaces provides an extremely effective means of feedback useful in guiding learning during the lifetime of a robotic system such as the TTS reported in this paper.

Findings – It was discovered that even though the adaptive learning methods were computationally more expensive than the classical algorithm implementations, they proved to be more effective in a number of cases, especially in noisy environments.

Originality/value – The novelty associated with this work is the introduction of an approach to adaptive learning carried out within the framework of ethology-based approximation spaces to provide performance feedback during the learning process.

Keywords Behaviour, Biology, Robots, Tracking, Intelligent agents

Paper type Research paper

1. Introduction

This paper introduces a target-tracking subsystem on a line-crawling robot (called ALiCE II) that represents the culmination of a five-year long research project with Manitoba Hydro (for a very detailed presentation of the hardware and software for this project, see Lockery (2007)). The ALiCE II target-tracking subsystem learns adaptively to follow a moving target. The problem considered in this paper is how a target-tracking system (TTS) can adaptively learn to control its actions based on information granules that reflect knowledge about acceptable behaviour patterns. The solution to this problem stems from an application of approximation spaces introduced

The authors gratefully acknowledge suggestions and insights by Christopher Henry, Maciej Borkowski, Andrzej Skowron, and David Gunderson concerning topics in this paper. This research has been supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) Grant 185986 and a number of grants from Manitoba Hydro.
by Pawlak (1981) starting in the early 1980s and which provide a basis for perception of objects that are imperfectly known. The particular form of perception we have in mind comes from Orlowska (1982, 1985), who suggested making observations at the class level rather than the individual object level.

To instill adaptivity in a TTS that learns, an Orlowska form of perception is combined with an ethological form of observation of agent behaviour inspired by Tinbergen (1963). That is, the TTS has been designed to observe and record each of its own behaviours in a table called an ethogram. The particular form of ethogram constructed by the TTS is approximation space-based and was introduced in Peters et al. (2005) and Peters (2005). Adaptive learning is considered episodic (Peters et al., 2006a). During each episode, the TTS constructs a new ethogram that provides a record of its states, actions, and rewards (for each action). As long as things are going well (i.e. the average reward received), an episode continues. Let $V(s), V(s')$ denote the value the current state $s$ and value of the next state $s_{-}$, respectively. In each state $s$, the TTS estimates $V(s')$ and compares $V(s')$ with $V(s)$. This comparison provides a stopping strategy for ending an episode (the $V(s)$ stopping strategy was suggested by Watkins (1989) based on Selfridge’s (1984) work on ill-defined system and what Selfridge called the run-and-twiddle control mechanism. Conventional actor-critic (Sutton and Barto, 1998) and Q-learning (Watkins, 1989) provide a framework for the control strategies introduced in this paper. The contribution of this paper is a framework for ethology-based adaptive learning defined in the context of approximation spaces and information granulation.

This paper has the following organization. A very brief overview of the ALiCE II robot is given in section two. This is followed in section three by an introduction to a suite of conventional reinforcement learning algorithms and the basic notation and framework for an approximation space and a rough coverage form of Q-learning. In Section 4, an overview of the TTS is presented. The experimental setup is presented in Section 5 and the results are given in Section 6. The conclusions and discussion of future possibilities are found in Section 7.

2. The ALiCE II robot
The target platform for the study discussed in this paper is the ALiCE II robot. The moniker ALiCE II stands for automated line crawling equipment and the design is in its second revision. The robot design was intended to operate autonomously once attached to a section of skywire, navigating back and forth, and hunting for targets of interest based on pre-programmed behaviours. ALiCE II can be seen in Figure 1, at home on a section of skywire during testing.

3. Background theory
This section gives a very brief overview of the reinforcement learning methods that provide a basis for adaptive learning.

3.1 Learning by trial-and-error
Rewarding behaviours are learned by a machine through trial and error experiences derived from an array of input sensors in a dynamic environment (Kaelbling et al., 1996). The trial-and-error method has associated rewards and punishment values for each action taken (Kaelbling et al., 1996). A benefit of this technique is that the reward
and punishment values can be provided without giving a definite specification of how tasks or goals are accomplished (Kaelbling et al., 1996; Sutton and Barto, 1998). This is an iterative procedure with an agent learning by selecting both desirable and undesirable actions, gathering experience and then basing future decisions on pre-defined goals or tasks. The iterative nature of reinforcement learning follows equation (1) as a general form of the update rule for each step in refining a policy (Sutton and Barto, 1998).

\[
\text{NewEstimate} \leftarrow \text{OldEstimate} + \text{StepSize} \times [\text{Target} - \text{OldEstimate}]
\]  

(1)

The informal goal of learning methods is to maximize the available returns in accordance with a pre-defined set of behaviours. There are three main parts included in the RL problem, states, actions and rewards. These three components make it possible to describe the state space in terms of a range of possible actions and their associated rewards (Watkins, 1989). Mapping from states to actions is commonly referred to as discovering a policy (denoted \(\pi\)) for exhibiting a desired behaviour based on pre-programmed goals through selecting actions meriting the greatest rewards (Kaelbling et al., 1996).

The RL problem is discussed next, followed by the algorithms implemented in the work reported including actor-critic, Q-learning, and sarsa. These algorithms provide a range of temporal difference reinforcement learning approaches.

3.1.1 The learning problem. Selecting a representation for states, actions and rewards are briefly discussed before including the adaptive learning algorithms. For the target-tracking problem, the states and actions are best described by a diagram representing the field of view of the camera (Figure 2).

The goal state is S4 and all others are considered sub-optimal states. The range of actions consist of servo motor steps and the arrows shown display the suggested direction based on the current state. The amount of movement is controlled by the particular algorithm during operation. The third component of the learning problem is
the definition of an appropriate reward function. In this work, a Euclidian distance measure was used from the image center to the target (equation (2)) and the reward is a normalized distance measure ranging from 0 (furthest distance) to 1 (goal state), shown in equation (3).

$$\text{distance} = \sqrt{x^2 + y^2}$$  \hspace{1cm} (2)

$$\text{reward} = 1 - \frac{\text{distance}}{\text{maxdistance}}$$  \hspace{1cm} (3)

3.1.2 The Actor-Critic algorithm. The symbol descriptions for this section are:
- $\pi(s,a)$: Policy mapping state to action selection.
- $p(s,a)$: Preference associated with selection action, $a$, given state, $s$.
- $\beta$: Step size parameter for preference revision.
- $\delta$: Temporal difference error.
- $r$: The reward provided for selecting an action in a given state.
- $\gamma$: Discount factor for weighting future rewards.
- $V(s_t)$: The value assigned to the current state at step $t$.
- $\alpha$: The learning rate step size adjustment parameter.

The Actor-Critic algorithm (Algorithm 1) is composed of two separate parts, an actor and a critic (Figure 3). The main idea stems from a concept called reinforcement comparison (Sutton and Barto, 1998), where positive actions are encouraged and negative actions are discouraged by adjusting the preference of selecting them for future decisions in a positive and negative manner respectfully (Sutton and Barto, 1998). The given policy being followed is referred to as a search element or actor since it is responsible for mapping any given state to an action (Sutton et al., 1983; Sutton and Barto, 1998). The value function or estimate is labeled the critic element since it is able to quantify the actions taken by the actor and suggest improvements to the policy (Sutton et al., 1983; Sutton and Barto, 1998). Together, the interaction of the policy and the value function make up the Actor-Critic approach to reinforcement learning. Through refining the actor’s approach, the critic helps to direct the agent’s focus toward the best policy for achieving its goal (Peters et al., 2006b). The algorithm is presented here but for a more in depth analysis of the algorithm, we refer the reader to Sutton and Barto (1998) and Sutton et al., 1983).
3.1.3 The Q-learning algorithm. The symbol descriptions for this section are:

- **Q(s,a)**: Action value associated with state.
- **α**: The learning rate step size adjustment parameter.
- **r**: The reward provided for selecting an action in a given state.
- **γ**: Discount factor for weighting future rewards.
- **π(s,a)**: Policy mapping state to action selection.

The Q-learning algorithm (Algorithm 2) presents a different approach to the RL problem, it makes use of an actor which learns the action value (Q) associated with an action in a given state (Figure 4). The simplest form is a single step temporal difference learning method that is capable of maximizing the action value of an agent regardless of the policy (Sutton and Barto, 1998; Kaelbling et al., 1996; Watkins and Dayan, 1992; Watkins, 1989). A single step algorithm looks into the future one step in advance when
estimating the best course of action to take from the current state. The best action is the choice that maximizes the future discounted reward available from all possible actions pertaining to the current state. Q-learning is an off-policy approach, meaning during revision of q-values, regardless of the policy being followed, the algorithm will always select the best action (Watkins, 1989). This is a result of the update rule for Q-learning (equation (4)).

\[ Q(s, a) \leftarrow Q(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a'} Q(s', a') - Q(s, a)] \] (4)

This approach provides exploration using the regular policy and during action value revision, exploitation of the best possible actions takes place. Together, they make a strong case for discovering the best policy and there have been several works that detail convergence proofs for Q-learning including Watkins (1989), Watkins and Dayan (1992) and Jaakkola et al. (1994).

For a more in-depth analysis, we refer the reader to the original work by Watkins (1989), formally reported in 1989.

3.1.4 The sarsa algorithm. The symbol descriptions for this section are:
- \( Q(s, a) \): Action value associated with state.
- \( \alpha \): The learning rate step size adjustment parameter.
- \( r \): The reward provided for selecting an action in a given state.
- \( \gamma \): Discount factor for weighting future rewards.
- \( \pi(s, a) \): Policy mapping state to action selection.

The final RL algorithm included is sarsa, named by Sutton (1996) and Singh and Sutton (1996). Originally, sarsa was treated as a variant of Q-learning and it was first introduced as modified Q-learning in the literature by Rummery and Niranjan (1994). A concern with single step Q-learning was that it would select only greedy actions with the nature of its off-policy approach to maximize the next state-action selection, potentially missing out on actions that could provide much improved rewards at a later point in time (Rummery and Niranjan, 1994). The difference between the two is in the update rule. Sarsa is an on-policy method, following the same policy for exploration and revising action values (equation (5)).

\[ Q(s, a) \leftarrow Q(s, a) + \alpha \cdot [r + \gamma \cdot Q(s', a') - Q(s, a)] \] (5)

Often, the exploring policy will be somewhat greedy, usually based on a parameter \( \epsilon \), also known as \( \epsilon \)-greedy policies (Sutton and Barto, 1998). This provides some degree of exploration along with a generally greedy trend when selecting actions. The parameter \( \epsilon \) is often selected through trial and error or from prior knowledge about the operating environment. Lower values favour more static environments where greedy tactics prevail and higher values favour more dynamic environments where some exploration is required to find the best actions for any given situation. The formal algorithm is included here in Figure 5 and for more detailed analysis of the algorithm, we refer the reader to Sutton and Barto(1998), Rummery and Niranjan (1994), Sutton (1996), Singh and Sutton (1996) and Singh et al. (2000).

3.1.5 Approximation spaces. In rough set theory, approximations are carried out within the context of an approximation space \( (O, F, \sim, \mu) \), where \( O \) is a set of objects,
$F$ is a set of functions representing object features, and \( \sim_B \) is an indiscernibility relation defined relative to \( B \subseteq F \). This space is considered fundamental because it provided a framework for the original rough set theory (Pawlak, 1981). It has also been observed that an approximation space is the formal counterpart of perception (Orlowska, 1982, 1985). Approximation starts with the partition \( \xi_B \) of \( O \) defined by the relation \( \sim_B \). Next, any set \( X \subseteq O \) is approximated by considering the relation between \( X \) and the classes:

\[
[x]_B \in \frac{O}{\sim_B}, x \in O.
\]

An approximation space \((O, F, \sim_B)\) defined relative to a set of perceptual objects \( O \), a set functions \( F \), and relation \( \sim_B \), has the following basic framework that facilitates observations concerning sample objects.

Framework for an approximation space:

\[
\begin{align*}
O &= \text{Set of perceived objects}, \\
F &= \text{Set of probe function objects}, \\
B &\in F, \\
\sim_B &= \{(x, x') \in X \times X \mid f(x) = f(x') \forall f \in B\}, \\
\frac{O}{\sim_B} &= \{[x]_B \mid x \in O, B \subseteq F\}, B - \text{partition of } O, \\
[x]_B &\in \frac{O}{\sim_B}, \\
X &\subseteq O, \text{ set of objects of interest}, \\
B \cdot X &= \bigcup_{x \in [x]_B \subseteq X} [x]_B, X \subseteq O, B - \text{lower approximation}, \\
B^* X &= \bigcup_{x \in [x]_B \cap X \neq 0} [x]_B, X \subseteq O, B - \text{upper approximation},
\end{align*}
\]
Affinities between objects of interest in the set \( X \subseteq O \) and classes in the quotient set \( O/\sim_B \) can be discovered by identifying those classes that have objects in common with \( X \). Approximation of the set \( X \) begins by determining which elementary sets \([x]_B \subseteq O/\sim_B \) are subsets of \( X \). This discovery process leads to the \( B \)-lower approximation of \( X \) denoted by \( B_*X \), where:

\[
B_*X = \bigcup_{x \in [x]_B, [x]_B \subseteq X} [x]_B.
\]

In effect, if \( B_*X \) (\( B \)-lower approximation of \( X \)) is not empty, then the objects in each class in \( B_*X \) have descriptions (i.e. function values for each function \( f \in B \)) matching the descriptions of the corresponding objects in \( X \). An upper approximation \( B^*X \) of a set \( X \) is defined by:

\[
B^*X = \bigcup_{x \in [x]_B \cap X \neq \emptyset} [x]_B.
\]

In effect, the \( B \)-upper approximation \( B^*X \) is a collection of equivalence classes \([x]_B \subseteq O/\sim_B \), where each class in \( B^*X \) contains at least one object with a description that matches the description of an object in \( X \). The lower and upper approximations of \( X \) provide a basis for defining the boundary of an approximation. Notice that \( B_*X \) is always a subset of \( B^*X \). Notice, also, that there may or may not be one or more equivalence classes \([x]_B \subseteq O/\sim_B \) that are not subsets of \( B^*X \).

3.1.6 Adaptive learning rate. A variant of the traditional reinforcement learning algorithms is included in architecture of the TTS by adding approximation spacebased refinement of the learning process. This entails refining the step size parameter for adjusting the learning rate of the algorithms during the learning process. This was achieved by capturing the observed behaviour at each step during an episode and storing it in an ethogram-like table (Tinbergen, 1963). At the end of an episode, the stored objects are partitioned into equivalence classes containing similar behaviours. Comparisons between the current behaviour and previously known acceptable behaviours provided the basis for a measure of the degree of rough coverage represented by each class. Averaging the rough coverage values from all behaviour classes provided a feedback metric that indicates how well the current set of behaviours is performing compared to previously known acceptable behaviours. The value of the rough coverage, or \( \nu \), was then subtracted from 1 and multiplied by the learning rate, \( \alpha \), providing an adaptive learning rate that adjusted to any given performance. In the event that the system was performing well and the rough coverage value was high, the adjusted value of \( \alpha \) would become small with the intent of avoiding overshooting the desired behaviour. When worse performance occurs, the rough coverage metric value would be smaller, resulting in a larger value of \( \alpha \), increasing the step size with the intent of moving more rapidly away from a bad policy.

This modification was made to all three algorithms and then compared to the classical versions. The revised formal algorithm including rough feedback is presented in Figure 6,
demonstrating the changes made. The other two algorithms have the exact same modification and as a result are not displayed.

The modified learning algorithms initialized the value for the average rough coverage $\bar{\gamma}$ to zero and the data table (ethogram) was cleared at the beginning of each episode. Episode lengths were variable, but averaged around 5,000 steps each. Reducing the amount of computation required for generating $\bar{\gamma}$, the amount of steps included in the data table was restricted to approximately 20 per cent or 1,000 steps out of each episode. This was done because slower performance was anticipated due to the nature of the extra computational requirements when compared to the standard reinforcement learning algorithms.

4. System overview

To control the robot behaviour through sensors, actuated joints and motor drives, several components were necessary. A TS-5500 single board computer was used for the heavy computational tasks whilst a PIC was used for low-level motor control to operate all servo-actuated joints and the motor drive.

For the software systems onboard the TS-5500, the chosen implementation platform was C++ and the development environment was gcc, Version 3.2.2 20030222, provided with Linux (2003). C++ was selected as the development language due to its speed, portability and familiarity. In addition, code written for the PIC was implemented using CC5X, a free C-compiler tailored specifically for PIC processors with reduced instructions and limited memory (Knudsen, 2006). The resulting object code from CC5X was compatible with the Microchip MPLAB integrated development environment (IDE) which was used to generate the machine code and program the PIC devices (Microchip Technology Inc., 2006). In addition to these software development packages, MATLAB was used for data analysis and output plots as it offered a wide range of analysis tools for generating results. The main goal of designing the software was to keep the code as compact and efficient as possible whilst providing a flexible environment that was easy to use for experimental work.

---

Figure 6.
Algorithm 4: the Q-learning method with rough feedback

<table>
<thead>
<tr>
<th>Input</th>
<th>States, $s \in S$, Actions $a \in A(s)$, Initialize $Q(s, a)$, $\bar{\gamma}$, $\alpha$, $\gamma$, $\pi$ to an arbitrary policy (non-greedy);</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>Optimal action value $Q(s, a)$ for each state-action pair;</td>
</tr>
<tr>
<td>while</td>
<td>True do</td>
</tr>
<tr>
<td>for</td>
<td>$(i = 0; i \leq # \text{ of episodes}; i++)$ do</td>
</tr>
<tr>
<td>Initialize</td>
<td>$s$ and data table;</td>
</tr>
<tr>
<td>Choose</td>
<td>$a$ from $s$, using policy derived from $Q$;</td>
</tr>
<tr>
<td>for</td>
<td>Repeat(for each step of episodes): do</td>
</tr>
<tr>
<td>Take action</td>
<td>$a$; observe reward, $r$, and next state, $s'$;</td>
</tr>
<tr>
<td>Record state, action, and associated reward in data table;</td>
<td></td>
</tr>
<tr>
<td>$Q(s, a) \leftarrow Q(s, a) + (1 - \bar{\gamma})\alpha[r + \gamma \max_{a'}Q(s', a') - Q(s, a)]$;</td>
<td></td>
</tr>
<tr>
<td>$s \leftarrow s'$; $a \leftarrow a'$;</td>
<td></td>
</tr>
<tr>
<td>until</td>
<td>$s$ is terminal</td>
</tr>
<tr>
<td>end</td>
<td>Generate $\bar{\gamma}$ from results recorded in data table for current episode;</td>
</tr>
<tr>
<td>Update new value of $\bar{\gamma}$;</td>
<td></td>
</tr>
<tr>
<td>Clear data table;</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td>while True do</td>
</tr>
</tbody>
</table>

---

54
The software was broken up into three separate components to separate processing tasks for the line crawling robot. Although some overlap was necessary, the three main components consisted of target tracking, reinforcement learning and the PIC interface. These three modules cover the software framework employed to operate the line crawling robot, including both the TS-5500 and the PIC controllers.

4.1 Target-tracking system

The purpose of the TTS was to lock onto specific targets and then to maintain it as close to the centre of the field of view as possible in order to capture the best possible images. Two separate techniques were implemented for comparison, a template matching method and an average grey-level approach. Template matching had greater accuracy at a cost of extra computational complexity. In contrast, the average grey-level method required much less-computational complexity but with a loss of resolution since it treats targets as sub-sections of an image instead of individual pixels. Although the two methods were compared to test, which was most suitable for the experimental environment, only the results for template matching are included in this paper. The remaining results can be found in Lockery (2007).

4.2 Classical target tracking

Adaptive learning algorithms were used to generate behaviours for target tracking during navigational behaviours of the ALiCE II robot. Classical target tracking was employed to provide a basis for comparison with the adaptive methods; it differs because no learning takes place during the tracking process. As a result, there were no rewards, and the states and actions differed. Instead of having states relating to subsections of the image, each pixel was treated as a separate state, implying that when the target was located at a given pixel, tracking the centre of the field of view to that pixel was the goal. Rather than having a range of actions available for movement, the tracking procedure moved to the exact location of the target within the field of view (Figure 7).

4.3 Software system overview

The class diagram for the TTS with reinforcement learning is presented in this section (Figure 8) with a few comments on how everything fits together.

The main method was responsible for initializing the system and starting the tracking behaviour. An additional client-server architecture was used so that remote monitoring of the results was possible as the TTS was intended to be implemented in a mobile robotic platform. The tracking system started up after the camera drivers were loaded and the first step in setting up the target tracking procedure was to lock onto a known target (or template) for the template-matching scheme. Once a template had

---

**Algorithm 5: classical target tracking**

Input: States, $s \in S$, one state for each possible set of target coordinates; 
Output: Deterministic Policy, $\pi(s)$; 
while True do 
  get current state; (coordinates of target) 
  pan $\leftarrow$ target's horizontal distance from centre of camera view; 
  tilt $\leftarrow$ target's vertical distance from centre of camera view; 
  move servos by (pan, tilt); 
end

---

Figure 7. Adaptive learning by TTS

55
been found and provided, the tracking system was ready to hunt for similar targets. Some image processing took place before examining each captured frame, consisting of conversion to grayscale and decimation to reduce the computational cost of processing each image. The driving reason behind concern over computational cost related to the target platform of an autonomous robot with limited processing capabilities and power supply. During the target-tracking procedure, the current state was determined by using a sum of squared differences metric for each template compared in an image.
The state was then passed to the control algorithm (either the classical tracking approach or one of the learning methods). At this point, the classical tracking algorithm would adjust the position directly to the target coordinate with the least error. The learning algorithms selected an action based on their given policies which were determined by various methods discussed in the section on reinforcement learning (see Section 3.1). The reinforcement learning algorithms updated their values associated with selecting an action in a given state during the tracking procedure. The rough coverage modified learning algorithms were identical except that they revised their step size parameter, $\alpha$ based on the performance metric $\bar{v}$. This was done through storing performance choices of the algorithm during an episode and later deriving the average rough coverage value from the stored ethogram containing the behavioural information. The multiplier allowed larger steps to be taken towards or smaller steps taken away from an improved policy depending upon the system performance (larger steps if performance is poor and smaller steps if performance is good).

The framework presented in the class diagram is intended for tracking targets and minimizing the position error from the centre of the field of view of the camera to the centre of the target. This applies for both stationary and mobile targets.

4.4 Learning methods
The purpose of the learning methods was to build a behaviour policy from a set of goals that were provided before releasing the robot into the experimental environment. Support for all three adaptive learning methods discussed was included for both the classic and the rough coverage modified versions. The learning methods were each responsible for taking the data presented from the tracking methods related to target location in the field of view and forming a picture of what the camera looked at for any given time step during the experiment. The data recorded formed the state and the learning method was then responsible for coming up with the best possible action to take for the given situation. The action was then forwarded to the PIC interface.

4.5 PIC control system
The PIC system was put into place for operating all of the low-level motor control strategies, separating it from the behaviour processing to speed things up and isolate potentially noisy motors. The PIC and the TS-5500 mainboard had a bi-directional communication link to allow the behaviours to dictate orders to the motors and then receive feedback. The PIC system operated all of the servo-actuated joints and the main locomotion drive. Collectively allowing for complete control over the robot and leaving room for future expansion to accommodate alternate goals with varied requirements.

4.6 Optimization
There were two areas investigated for optimization during development the timing interval for servo motors and the amount of processing power used for RL methods using rough coverage.

The timing interval for the servo motor updates was restricted to the length of the control signal refresh rate. As a number of different servos were used, the control signals varied from one to another, the shortest maximum duration was used. The total time for one refresh cycle of the control signal was 20 ms.
All commands needed to be issued within one 20 ms window. The position of the line grip actuated joints moved in a $90^\circ$ range and since most position control servo motors have a range of about $180^\circ$ starting from 0 to $90^\circ$ (or from $90$ to $0$) reduces the length of the control signal as it translates directly to the position based on the length of the signal. Forcing the range of the grip position servos to move from 0 to $90^\circ$ when opening and closing reduced half the control signal length for both motors. This provided enough extra time to include another servo motor (future expansion) and helped reduce the amount of signal traffic to a minimum during the 20 ms window.

The other area for optimization that was investigated related to RL methods during target tracking. The rough coverage methods were more computationally intensive than the original algorithms. Each time an episode completed and the new coverage values were generated, the processing requirements were expensive enough to halt the camera system temporarily. This was problematic for keeping a target in the field of view at all times to acquire the best images. A means to limit the computational cost was investigated resulting in approximately 20 per cent of the steps in any given episode being used to form the ethogram for generating coverage values. This provided a large enough sample of an episode to give a reasonable view of the behaviour performance and at the same time yielded a significant reduction in computational cost, minimizing the delay during tracking.

5. Experiment design
The experimental setup consisted of simulating environments for the line crawling robot that closely approximate actual scenarios that could be expected. Test situations were developed through an iterative design procedure for both the hardware and software systems to provide a collection of results from which conclusions could be drawn regarding system performance.

5.1 Hardware experimental environment setup
In order to run experiments, a replica monocular vision system was built separate from the line crawling robot with the intent of having a separate system for testing and experimental work, eliminating the need for battery power, periodic recharging and reducing possible risk of damage to the line-crawling robot. The target-tracking experiments required larger periods of time to gather useful data. As a result, building a safe testing environment was desirable to avoid the need for constant supervision. The replica was built as close as possible to the actual camera system configuration to ensure comparable results (Figure 9).

5.2 Hardware controllable parameters
There were three variable hardware parameters, distance from the target to the camera, height of the target and slope of the sky wire. During experimental work, the target for the line-crawling robot was insulator stacks. Based on tower dimensions and the location of insulators, a ratio of four to one for distance versus height was used with the targets. The distance to the target was selected at 17 cm, and the height of the corresponding object was 4.25 cm. The slope of the skywire was controlled by proximity of the end towers to the central tower in the experimental setup.
5.3 Software system parameters

There were a number of different variable parameters in the software systems. A discussion related to each learning algorithm and its associated parameters are included for the Actor-Critic, Q-learning and sarsa methods.

5.3.1 The Actor-Critic algorithm. The Actor-Critic algorithm contained several variables. Selection of the initial values were made as follows. The learning rate, $\alpha$ was chosen to have a value of 0.1, preventing large correcting steps during the learning process. The step size parameter, $\beta$ for controlling preference adjustment was also conservatively selected at 0.1 to avoid giving too much or too little preference to actions that performed well or poor early on and mistakenly either avoiding or over selecting them as a result. The discount factor, $\gamma$ was initialized at a value of 0.1 to avoid giving too much weight to future states, making the system somewhat near-sighted. The reason for choosing a low value of $\gamma$ was based on the minimal amount of environmental noise expected. The parameter, $\epsilon$ which prevented the policy $\pi$ from always being greedy was selected as 0.1 as well, implying that only 10 per cent of the time exploratory actions were taken. The preference values were all initialized to zero which set the starting values for the policy $\pi$. The initial policy values were unbiased by setting all values to one, making it equally likely to select any action before experience was gathered.

5.3.2 The Q-learning algorithm. To keep the learning algorithms' respective performances on a reasonably level playing field, similar parameter values were chosen whenever possible. The values of $\alpha$, $\epsilon$ and $\gamma$ were once again selected at 0.1. The action values, $Q(s,a)$ were initialized by setting all values to zero and the initial policy $\pi$ was set to an arbitrary soft or non-greedy policy with at least some chance to visit all possible state-action pairs.

5.3.3 The sarsa algorithm. The sarsa algorithm parameters were chosen with the same values for $\alpha$, $\epsilon$ and $\gamma$ at a value of 0.1. The action values, $Q(s,a)$ were again initialized to zero, and the policy $\pi$ was selected as an arbitrary soft policy as well.
5.4 Test vector development

Generating a set of test vectors to exploit the differences between the various learning methods was an iterative process. Careful selection of components including the period of time for each experiment, the distance and height of the target, movement patterns and the addition of noise were all considered.

The length of time for each of the tracking experiments was kept uniform for all algorithms. Varying amounts of time for the learning process allowed performance comparison for slower learning methods. Times ranged from 1 min up to 2 h, including experiments for 5 min, 15 min, and 1 h. The distance and height of the target being tracked were fixed (see Section 5.2). Target movement included two patterns, circular and random trajectories. Finally, noise was added to provide some degree of similarity to environmental stresses that are possible outdoors, including high winds, uneven lighting, or precipitation.

The additive noise was limited to dimensions of the field of view, implying that the mis-perceived target remained within the field of view of the camera. Generating additive noise was accomplished with a pseudo-random number generator to supply uniformly distributed random numbers ranging from $-1$ to $1$. The random values were scaled up to integer values and added to the target location to attempt to confuse the tracking system. The random number generator was considered pseudorandom as it relied on a seeded value from the system clock to generate a sequence of random numbers (Press et al., 2002). A uniform deviate was selected to provide an equally likely chance of any value occurring throughout the range of $-1$ to $+1$.

The last adjustable parameter was speed of movement for the target. An initial fixed speed of $(6,6)$ was used, which refers to the amount of pixels moved in both the $x$- and $y$-directions in 2D space that the camera can see. The time between movements was 50 ms, resulting in a total movement rate of $(120,120)$ pixels/second. The range of speeds included from $(6,6)$ to $(10,10)$ pixels per 50 ms.

6. Experimental results and discussions

This section contains results compiled from experiments with the line crawling robot. They were designed to explore the differences in tracking techniques under a variety of conditions with the intent of acquiring the best quality images. Each experiment includes a discussion for what was expected, actual results and implications of the results. Several experiments were processed to determine the performance of each algorithm, including varying the time available for the learning process, altering target speed, noise susceptibility, and variable target trajectories.

6.1 Varied time target tracking

The first experiment varied the length of continuous time for the target tracking process. As the time increased, results were expected to favour the learning methods as they tend to converge toward an improved policy over time.

Each algorithm was implemented and reported using two different lengths of time, one minute, and 5 min as this was sufficient to demonstrate performance trends. The algorithms included were the classical tracking, actor-critic, Q-learning, sarsa and the rough feedback modified versions of each of the learning methods yielding seven in total for each experiment. Static components of the test vectors included target speed at
(6,6) pixels of movement per 50 ms, target distance and height (17 and 4.25 cm, respectively) and target movement pattern (circular clock-wise direction).

The first set of results included were for one minute in duration. The 1 min experiments seen in Figure 10 are somewhat erratic due to the short duration of the test. The learning methods often take a little bit of time to settle after preliminary exploration of new state space. Classical tracking remained relatively stable throughout since no learning took place. The Actor-Critic and sarsa methods exhibited the lowest RMS error when tracking the target. Q-learning had the most erratic profile and averaged the highest error rates during the first minute. The rough-feedback methods for both the Actor-Critic and Q-learning algorithms provided improved results over time compared to the classical implementations. The improvements were small, approximately 0.1 to 0.2 pixels of accuracy for the short period of time.

The next set of results was for a duration of 5 min. An improvement was expected in the adaptive learning methods due to increased time for generating a policy. The rough-feedback methods were expected to provide performance equal to or better than the classic learning algorithm implementations as the adaptable learning rate allowed for a more intelligent policy adjustment. The results seen in Figure 11 show that several of the predictions were accurate. However, the classical tracking method

---

**Figure 10.**
Template matching one minute tracking experiment average RMS error results
performed worse than in the one minute trials, yielding an error rate of 4.2 pixels, most likely caused by varied illumination. Unfortunately the experiments were not all performed at the same time of day and the ambient light was difficult to control. Both Q-learning and sarsa showed improved performance with the rough-feedback versions over the longer period of time. Rough-feedback Q-learning exhibited the least error during the tracking task, averaging close to three pixels of average RMS error, closely followed by the classical Actor-Critic method, averaging 3.2 pixels of RMS error. The results for Q-learning and sarsa suggest that the rough-feedback modified algorithms were not demanding too much computational time or sacrificing accuracy based on the revised ethogram length.

Examining the results for each of the tracking algorithms indicated that the trade-off associated with extra computations required by the adaptive learning methods paid off in accuracy. Rough-feedback Q-learning was the most consistent performer suited to the task of target tracking in the different lengths of time. This was not a surprising result since Q-learning exhibits a more deterministic approach in selecting future actions in any given state due to the nature of the algorithm and the environment was relatively static with minimal external noise.

6.2 Variable target speed
To provide insight into performance with faster moving targets, the speed was increased. Results are included for a speed of (10, 10) pixels of movement per 50 ms for
comparison with the results from the previous section where movement was restricted to (6,6) pixels/50 ms. This consisted of an increase of movement rate from 120 pixels/s up to 200 pixels/s.

The algorithms, and the remaining parameters for the test vectors remained static, including the distance and height of the target (17 and 4.25 cm, respectively), target velocity (10,10) pixels/50 ms, length of each experiment (set at 5 min) and the target movement pattern (clock-wise circular). A total of 5 min length experiments were selected as they provided enough time for learning methods to settle.

The results are shown in Figure 12. Each of the algorithms exhibited an increased average RMS error. The Actor-Critic algorithm outperformed the rest, with an average RMS error of 3.7 pixels. Classical Q-learning was the only learning method that had worse performance than classical target tracking. The remaining learning methods either decreased or maintained an average RMS error similar to their starting values over time. The likely reason for a drop in performance of action-value methods is based on how they learn. The entire set of possible actions are scanned to determine which is most suitable in any given state. A faster target would move further by the time a new action was selected, increasing the error. The Actor-Critic algorithm has a separate policy that can help avoid the need for scanning the entire action space as it will dictate

Figure 12. Template matching, five minute high speed (10, 10) tracking experiment average RMS error.
which actions to take using the policy. This implies that once it has become experienced, the policy can reduce the amount of hunting for actions.

6.3 Random target trajectories

Random movements were employed to see how the algorithms fared when there was no standard movement pattern. This also helped provide some insight into how short sharp movements with the line crawler during travel, operations on the line or even in gusting wind conditions might be handled.

The fixed parameters included distance and height of the target (17 and 4.25 cm, respectively), target velocity (6,6) pixels/50 ms, target-trajectory pattern (random), and the time duration for each experiment (5 min). The results in Figure 13 showed that the learning methods improved with the random movement versus the circular clock-wise trajectory. This is likely because once the outer edge of the range was reached the target tended to move back the way it came from for at least a short duration before moving in a different direction. The result being that the camera did not need to move much to centre the target. In comparison, the circular trajectory continued moving in a wide enough arc that continual tracking was needed to centre the target in the field of view. Even classical tracking without learning exhibited an improved performance over the original 5-min test.

Figure 13. Template matching, five minute random trajectory target tracking experiment average RMS error
The TTS responded more favourably to targets with random movement trajectories. Random trajectory target movements were often interior to the maximum circumference, making it less difficult to centre in the field of view and simulate more closely actual target movement as circular motion would be fairly uncommon in field trials. The most accurate algorithms were rough-feedback Q-learning and the classical Actor-Critic implementations. Both of these techniques provided an average RMS error rate of approximately 2.4 pixels.

6.4 Noise susceptibility

A uniform deviate was used to add random noise to the target location before presenting the information to the tracking algorithms. All parameters for the test vectors were kept constant during this experiment, including distance and height of the target (17 and 4.25 cm, respectively), target velocity (6,6 pixels/50 ms), target trajectory (clock-wise circular pattern), duration (5 min), the chance of noise at any given iteration (50 per cent), and possible range of noise magnitude in camera movement steps ($\pm 5$).

The results expected for noisy environments favoured the learning methods since they were able to make adjustments to their actions in any given state over time they should offer improved performance over classical tracking. The Actor-Critic method has been most resilient up to now and its efficiency in dealing with rapid change made it the favourite prior to running the experiments. The results are shown in Figure 14. After examining the results in Figure 14, the performance predictions were not far off. The tracking algorithm least affected by noise was the Actor-Critic method, exhibiting an average RMS error of approximately 3.05 pixels. Surprisingly, rough feedback Q-learning performed comparably well in the noisy environment, yielding an average RMS error of 3.18 pixels. The greedy nature of both Q-learning algorithms saw them outperform their sarsa equivalents. As expected, the classical algorithm performed the worst, with an average RMS error of 3.95 pixels as it was unable to compensate for input noise.

The actor-critic algorithm responded well to a noisy environment. The maximum possible movement allowed in any state was 11 steps, providing noise with a magnitude of five steps had the potential to easily confuse the tracking methods, causing more average RMS error. This can be seen readily in the classical algorithm results as well as the action value learning methods. The two algorithms most suited for target tracking in a noisy environment include the classical implementation of the actor-critic algorithm for its speed and accuracy and the rough-feedback Q-learning algorithm for its accuracy.

The complete set of results containing the average RMS error, rated in pixels for each experiment is presented in Table I. Each of the algorithms is included and the letters “RF” denote rough feedback (adaptive) learning.

7. Conclusions and future work

Various forms of adaptive learning are introduced in this paper. In a noisy environment, experimental evidence has shown that the adaptive learning methods outperform classical target tracking. Classical actor-critic and rough feedback (adaptive) Q-learning do best in the noisy environments used to obtain the reported results. Rough feedback (adaptive) Q-learning consistently did better in tracking
targets in variable lengths of time. The classical actor-critic method outperforms the other learning methods in tracking fast-moving targets. Depending on the conditions, either conventional actor-critic or adaptive Q-learning are good choices as a control strategy during target tracking. Future work includes the application of these methods in tracking the movements of patients during video game-assisted, therapeutic exercise sessions.

**Table I.**
Experimental results in average RMS error (pixels)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>1-min test</th>
<th>5-min test</th>
<th>15-min test</th>
<th>Random trajectory</th>
<th>High-speed target</th>
<th>Additive noise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classic track</td>
<td>3.8138</td>
<td>4.1905</td>
<td>3.829</td>
<td>3.4228</td>
<td>5.3605</td>
<td>3.9404</td>
</tr>
<tr>
<td>Actor critic</td>
<td>3.5107</td>
<td>3.1835</td>
<td>3.0605</td>
<td>2.3757</td>
<td>3.7063</td>
<td>3.0543</td>
</tr>
<tr>
<td>Q-learning</td>
<td>4.2086</td>
<td>3.3548</td>
<td>3.4794</td>
<td>2.6673</td>
<td>5.4906</td>
<td>3.3225</td>
</tr>
<tr>
<td>Sarsa</td>
<td>3.276</td>
<td>3.6599</td>
<td>3.5874</td>
<td>2.6604</td>
<td>5.9056</td>
<td>3.8578</td>
</tr>
<tr>
<td>RFActorCritic</td>
<td>3.3628</td>
<td>3.1301</td>
<td>3.0509</td>
<td>2.5526</td>
<td>3.6433</td>
<td>3.163</td>
</tr>
<tr>
<td>RFQ-learn</td>
<td>3.9593</td>
<td>3.1384</td>
<td>2.8454</td>
<td>2.4576</td>
<td>4.8577</td>
<td>3.1773</td>
</tr>
<tr>
<td>RF sarsa</td>
<td>4.4166</td>
<td>3.7095</td>
<td>3.5751</td>
<td>2.9405</td>
<td>5.4647</td>
<td>3.5866</td>
</tr>
</tbody>
</table>

**Figure 14.**
Template matching, five minute noise susceptibility tracking experiment average RMS error
References


**About the authors**

Daniel Lockery (EIT, BSc, 2003, MSc, 2007) is a PhD student in the Department of Electrical and Computer Engineering (ECE) at the University of Manitoba. He is a student member of the IEEE, RAS, and APEGM. His research interests include image processing, robotics and automation, control systems, reinforcement learning, and biologically inspired designs of intelligent systems.

James F. Peters received his PhD from Kansas State University in 1991. He is a Full Professor in the Department of ECE, University of Manitoba. He received a 2007 Springer Best Paper Award for his work with Dan Lockery on a telerobotic TTS. His scientific interests include adaptive learning, adaptive gaming, machine intelligence, near sets. James F. Peters is the corresponding author and can be contacted at: jfpeters@ee.umanitoba.ca

To purchase reprints of this article please e-mail: reprints@emeraldinsight.com
Or visit our web site for further details: www.emeraldinsight.com/reprints