Advances in Computational Intelligence (2022) 2:26
https://doi.org/10.1007/s43674-022-00037-9

REVIEW ARTICLE

# An unsupervised autonomous learning framework for goal-directed behaviours in dynamic contexts

**Chinedu Pascal Ezenkwu**[1,2] · **Andrew Starkey**[1]

Received: 23 August 2021 / Revised: 25 April 2022 / Accepted: 27 April 2022


**Abstract**
Due to their dependence on a task-specific reward function, reinforcement learning agents are ineffective at responding to a dynamic goal or environment. This paper seeks to overcome this limitation of traditional reinforcement learning through a task-agnostic, self-organising autonomous agent framework. The proposed algorithm is a hybrid of TMGWR for self-adaptive learning of sensorimotor maps and value iteration for goal-directed planning. TMGWR has been previously demonstrated to overcome the problems associated with competing sensorimotor techniques such SOM, GNG, and GWR; these problems include: difficulty in setting a suitable number of neurons for a task, inflexibility, the inability to cope with non-markovian environments, challenges with noise, and inappropriate representation of sensory observations and actions together. However, the binary sensorimotor-link implementation in the original TMGWR enables catastrophic forgetting when the agent experiences changes in the task and it is therefore not suitable for self-adaptive learning. A new sensorimotor-link update rule is presented in this paper to enable the adaptation of the sensorimotor map to new experiences. This paper has demonstrated that the TMGWR-based algorithm has better sample efficiency than model-free reinforcement learning and better self-adaptivity than both the model-free and the traditional model-based reinforcement learning algorithms. Moreover, the algorithm has been demonstrated to give the lowest overall computational cost when compared to traditional reinforcement learning algorithms.

**Keywords** Autonomous agent · Planning · Unsupervised learning · Sensorimotor · Artificial intelligence

## 1 Introduction

As the scope for robotic applications extends from structured to unstructured and more complex environments, autonomy has become a desideratum for most of today's robots. The practice of handcrafting robots does not give them the capability to cope with unforeseen situations. Although several research contributions have been made towards robot autonomy, we are nowhere near the level of autonomy that is exhibited by animals, even ones at the lowest biological level of organisation. This is because animals are born with innate capabilities, both in their body structure and intelligence, to survive and develop in their milieus; their behaviours and sometimes their morphological traits can evolve to adapt to persistent changes in their habitats. While it can be argued that highly specialised robots can be developed for any given environment, current approaches in AI do not give the ability for the robot to self-adapt to changes in its environment (or from its simulated world) or to itself (for example through damage to actuators) and this is a significant gap to overcome to produce truly autonomous systems.

An autonomous robot, without any need for an external change of the underlying algorithms, should learn to develop novel skills to cope with unpredictable situations. The desire to attain this level of intelligence has been a major motivation for most sophisticated and popular AI techniques such as deep learning, reinforcement learning (RL), active learning, and imitation learning. Some of these methods either depend on labelled data or require environment-dependent reward functions, making it difficult for them to cope with unknown environments (Irpan 2018; Marcus 2018; Dulac-Arnold et al. 2021). For example, the model-free RL (Sutton and Barto 2018; Bozkurt et al. 2021), although one of the most applied

✉ Chinedu Pascal Ezenkwu
chineduezenkwu@abdn.ac.uk

Andrew Starkey
a.starkey@abdn.ac.uk

[1] School of Engineering, University of Aberdeen, Aberdeen, UK

[2] Electrical, Electronic and Computer Engineering, University of Uyo, Uyo, Nigeria

Published online: 02 June 2022

techniques in autonomous agent research with a number of impressive results suffer from sample inefficiency, delayed reward, and the need to design an environment-dependent reward function (Irpan 2018; Sermanet et al. 2016). Moreover, the model-based RL, although it has been proven to be sample-efficient and exhibits good generalisation ability, requires prior knowledge of the environment dynamics which are often not available in many real-world scenarios (Bozkurt et al. 2021; Sutton and Barto 2018). These limitations, however, pose difficulties in applying these methods in unknown or dynamic environments.

Through interactions with its environment, without any need for an environment-dependent reward function and/or predefined transition probabilities as may be required in the conventional RL algorithms, a truly autonomous agent should be able to learn a non-task-specific sensorimotor schema (Tsou 2006; Piaget and Cook 1952; Nguyen et al. 2021), which could later be useful for causal reasoning and planning towards a desired state leveraging rewards or motivations derivable from the sensorimotor space given desire. The derivation of the reward signals in the sensorimotor space can be likened to the activity of dopamine, a brain substrate that has been proven to cause pleasure, satisfaction, or dissatisfaction (Berridge et al. 2009; Liu et al. 2021) in humans and animals. This is a good way to avoid the challenge of defining reward function in the environment space, especially in situations where the environment is unpredictable, inaccessible, or completely unknown.

The preceding statement forms the thesis of this research. This study provides a strategy based on the unsupervised learning paradigm, specifically, the self-organising map (SOM) (Kohonen 1990). However, the standard SOM and its variants have fundamental issues. There is no one self-organising technique that has addressed the following problems: difficulty in setting a suitable number of neurons for a task, inflexibility, the inability to cope with non-markovian environments, challenges with noise, and inappropriate representation of sensory observations and actions together. The purpose of this research is to design an unsupervised, self-adaptive autonomous learning framework based on a variant of self-organising map called temporospatial merge grow when required (TMGWR) network (Ezenkwu and Starkey 2019b). The key contributions of this research include the following: (1) instead of employing a binary sensorimotor-link or lateral connections between nodes, this research provides a strategy that continuously strengthens or weakens a sensorimotor-link according to how reliable the link is; (2) an autonomous learning framework that incorporates value iteration in TMGWR for self-adaptive model-based planning in dynamic contexts has been proposed; and (3) research demonstrates that the proposed method is more flexible to changes in the agent environment than RL agents.

As a sensorimotor map learning algorithm, TMGWR yields a graph model ( or a sensorimotor map) with nodes representing the sensory abstractions and edges representing possible affordances from each node. It equips the agent with some knowledge of the world, making it possible to plan with an informed graph search technique which is more sample-efficient than the model-free RL methods such as Q-learning (Bozkurt et al. 2021; Sutton and Barto 2018). This paper demonstrates how the sensorimotor map learnt using the TMGWR algorithm can be exploited for goal-directedness using value iteration. The proposed method is compared with both the model-free RL (Q-Learning) and the model-based RL agents in terms of their sample efficiencies and their abilities to self-adapt when there is a change in the world or goal state.

The results show that both the TMGWR-based agent and model-based RL agent are far more sample-efficient and adapt faster to changes in goal states. However, in the change of environment scenario, the TMGWR-based agent adapts faster than the model-free RL agent, while the model-based RL agent completely fails to cope with any change in the environment. Although both require environment models to plan, the difference between the TMGWR-based method and the conventional model-based RL is that while the model-based RL requires the prior definition of the environment transition model, the TMGWR-based method learns this transition model. This advantage makes the TMGWR-based method suitable for applications in scenarios where the environment dynamics are not known *a priori*.

The remaining part of the paper proceeds as follows: Section 2 provides background information on some key concepts. Section 3 presents the conceptual framework of the proposed method with detailed descriptions of its major components. Section 4 compares the proposed method and the RL algorithms. Section 5 explains the experiments. Section 6 presents and discusses the results of the experiments, while Sect. 7 concludes the paper and highlights areas of future research.

## 2 Background

In this section, we provide background on machine autonomy, RL, sensorimotor map, and value iteration.

### 2.1 Machine autonomy

There have been many attempts towards creating autonomous systems with the target of achieving human-level performance in different scenarios. These have led to the development of a number of novel state-of-the-art AI techniques in recent years (Vamvoudakis et al. 2015; Saba et al. 2021).

However, these methods are not being implemented against a coherent evaluation framework for assessing the autonomous agent. In an attempt to provide a reasonable definition of machine autonomy, we previously categorised the attributes of autonomous agents into low-level and high-level attributes. This definition builds on a number of other definitions of the autonomous agent in the literature (Ezenkwu and Starkey 2019a).

The low-level attributes are must-have attributes for any autonomous agent as they provide the smallest distinction between the autonomous agents and other automated agents. These attributes include learning, context-awareness, actuation, perception, and decision-making. In contrast, the high-level attributes are advanced attributes of autonomy, which have proven difficult to achieve using the current AI techniques. They include domain-independence, self-motivation, self-recoverability, and self-identification of goals.

### 2.2 Reinforcement learning

RL is a trial-and-error method of learning policies and planning in Markov decision processes (MDPs) (Belousov et al. 2021). An MDP "consists of states $s \in S$, actions $a \in A$, a reward function $R(s, a)$ and transition probability, $P^a_{ss'}$, to each possible next state $s'$ given any state $s$ and action $a$". The goal of a RL agent is to maximise expected long-term discounted rewards over a horizon (episodic or continual) (Szepesvári 2010; Belousov et al. 2021).

Based on an explicitly defined immediate reward of states, $R(s)$, or state-action pairs, $R(s, a)$, all RL algorithms estimate the value functions which give the measure of how good each state (or state-action pair) is for a given task. If the process converges to the optimal value function then a greedy policy following the value function will be the best policy for the task. MDPs can be approached either as model-based or model-free RL. The model-based approach is suitable only when the environment dynamics are well known.

Typical examples of the model-based method are dynamic programming (Bellman 1952) techniques such as value iteration and policy iteration (Yang et al. 2021); the model-free method does not require knowledge of environment dynamics to arrive at the optimal policy, e.g., Q-learning (Ge et al. 2021) and SARSA (Sutton and Barto 2018). Although model-free methods are more popular than model-based methods because of their simplicity, the advantages of model-based methods over them are that model-based methods are more sample-efficient and have stronger generalisation.

### 2.3 Sensorimotor maps

According to Piaget, human infants are born with innate schemas or reflexes which develop into constructed schemas as they use these reflexes to adapt to their environments (Huitt and Hummel 2003; Hakimzadeh et al. 2021). This idea has influenced numerous research works in developmental AI and autonomous systems. Drescher was the first to formalise Piaget's sensorimotor schema, describing it as a tuple of context, action, and result. This is to enable the prediction of the result if an action is taken in a known context (Drescher 1991; Guerin and Starkey 2009). With a chain of regular schemas, composite actions can be generated and executed to attain a desired state. To improve the efficiency of Drescher's schema mechanism, Chaput (Chaput 2004) proposed the constructivist learning architecture (CLA), an unsupervised hierarchical neural implementation of the schema mechanism.

However, the CLA has a scale-up problem, because it is limited to binary sensory environments. Hierarchical SOM-based Encoding (HSOME) (Pierris and Dahl 2017; Parisi and Wermter 2013) is among the works inspired by the CLA. Although HSOME removes the restriction on the height of the hierarchy of self-organising maps (SOMs) (Kohonen 1990) that are required for learning, its major limitation is that "it produces lossless encoding instead of a principle component representation" (Pierris and Dahl 2017) of the agent's sensory observations.

Due to the preceding limitation and because HSOME uses the standard SOM which requires that its dimension is predetermined during design time, the sizes of the SOMs at different layers should be large enough to represent the agent's world properly. This makes HSOME less suitable for unknown scenarios. Similarly, Toussaint proposed the CWM (Toussaint 2004), which is a monolithic growing self-organising neural network designed to couple sensory observations and motor signals. Toussaint called this the sensorimotor map (Toussaint 2006). The CWM uses the growing variant of SOM to avoid the difficulty that the predetermination of accurate SOM size could pose to sensorimotor map learning in unknown environments.

Since sensorimotor map learning has been proven to be domain-independent or non-task-specific, it meets some of the requirements of the autonomous agent. Other advantages of employing sensorimotor maps are that they can provide data compression by encoding values from a multidimensional vector space into a finite set of values from a discrete subspace of lower dimension. Additionally, sensorimotor maps can provide both forward and backward functions for decision-making as we have demonstrated in this paper.

## 3 Conceptual framework for the proposed method

Figure 1a shows our concept of the autonomous learning agent, which will be referred to as the TMGWR-based agent for the remainder of this paper. The framework consists of

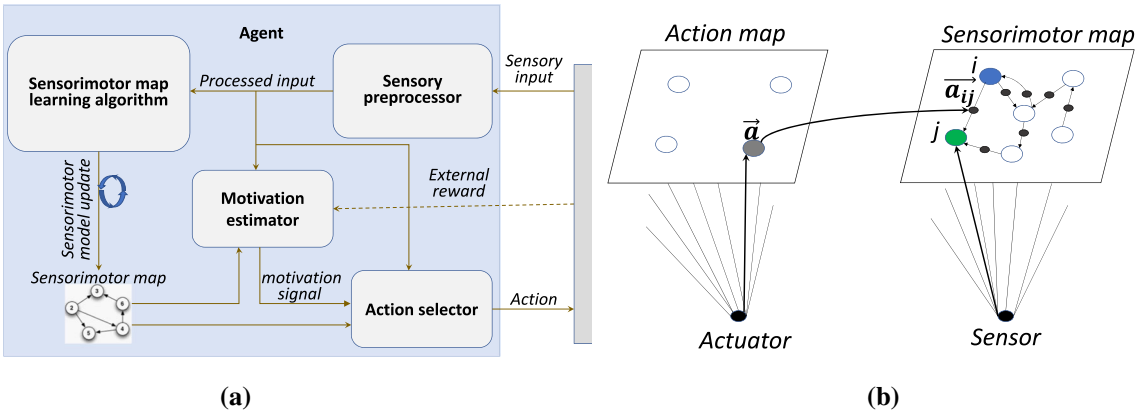



**Fig. 1** **a** Shows the conceptual framework for the autonomous agent. **b** Presents the sensorimotor map learning module of the architecture

four main modules—the sensorimotor map learning module, the sensory preprocessor, the motivation estimator, and the action selector. The agent is equipped with suitable sensors and actuators which enable it to observe the environment and react to these observations using the actuators. The observations or sensory inputs can be preprocessed or transformed into a form that conveys meaningful or contextual information to the agent. The preprocessed sensory observations are passed on to the sensorimotor map learning module which enables the agent to develop or refine its mental model of the scenario. The sensorimotor map learning occurs continually in an open-ended manner to enable the agent to keep track of changes in the environment by continuously updating the sensorimotor map. The motivation estimator provides the motivation signal that enables the agent to plan towards a goal or behave in a given manner in the environment. The action selector considers the current observation and the agent's motivation in selecting the best action using the sensorimotor map. Execution of this action causes a change in the environment and the cycle continues. Subsequent sections provide detailed explanations of the modules.

### 3.1 Sensorimotor map learning

The TMGWR network was used as the sensorimotor map learning method. The key features of the method are that:

- the nodes are linked based on their sensorimotor proximities to one another;
- it uses the temporal context of the Merge Grow Neural Gas (MNG) Strickert and Hammer 2005 to keep track of the sensorimotor history;
- the GWR method of adding new nodes is employed to enable the system to keep track of changes in the environment;
- all the hyperparameters are kept constant throughout the lifetime of the agent to encourage continual learning.

The action map in Fig. 1b learns the codebook vector $\vec{a}$ for each motor activity, while the sensorimotor map learns the input weight vectors and the possible action vectors linking them to each other. At each time step, the activated action vector on the action map is associated with the sensorimotor-link from the previous winning node $i$ to the current winning node $j$ in the sensorimotor map; this action vector is labelled $\vec{a}_{ij}$ in Fig. 1b. In the original TMGWR network, the sensorimotor-link $E[i, j]$ from node $i$ to $j$ is binary i.e. $E[i, j] = 1$ if $E[i, j]$ is a possible transition, otherwise $E[i, j] = 0$. However, in this work, the sensorimotor-link is updated according to Eq. (1)

$$E[i, j] = E[i, j] + \alpha \Delta_E, \tag{1}$$

where $\alpha$ is the learning rate and $\Delta_E$ is given by Eq. (2)

$$\Delta_E = sim(\vec{a}_{ij}, \vec{a}) \mathcal{I}\{i \nsim j\} - E[i, j]; \tag{2}$$

$sim(\vec{a}_{ij}, \vec{a})$ is a similarity function that compares the activated action vector, $\vec{a}$ at a given time with the action vector, and $\vec{a}_{ij}$ that has already been associated with the sensorimotor-link from node $i$ to node $j$. We chose $sim(\vec{a}_{ij}, \vec{a})$ to be a Gaussian function, so that if $\vec{a}_{ij} \simeq \vec{a}$, then $sim(\vec{a}_{ij}, \vec{a})$ tends towards 1; otherwise, it will tend towards 0. The advantage of introducing $sim(\vec{a}_{ij}, \vec{a})$ in the $E[i, j]$ update equation is that it increases the weight of the sensorimotor-link if the same action vector results in the same transition all the time and decreases it if the transition is possible with different action vectors. This is useful during planning as the agent is more likely to select reliable actions for each experience in the environment. $\mathcal{I}\{i \nsim j\}$ is an indicator function which returns 1 if the previous winning node $i$ is not the same as current winning node $j$; otherwise, it returns 0. $\mathcal{I}\{i \nsim j\}$ encourages only actions that result in progress and penalises those that compromise that progress.

### 3.2 Sensory preprocessor

The real world is often characterised by noisy, ambiguous, and/or multidimensional sensor signals. As such, an autonomous agent requires a preprocessor for extracting relevant information for learning and decision-making. The TMGWR network has been shown to inherently handle noisy, ambiguous, and partially observable environments due to its recurrent architecture and the GWR technique of expanding the sensorimotor map. While this has been proven in a 2D feature space, we argue that advanced preprocessing techniques could be required to complement the TMGWR network in a high-dimensional sensor space such as a vision-based scenario. For instance, a pre-trained convolutional feature extractor (Yosinski et al. 2014; Razavian et al. 2014) could be useful in the preprocessor module for feature extraction when the agent's observations are image frames. We intend to investigate the preceding example as well as methods for relevant feature selection (Wang et al. 2019) as the sensory preprocessor in our subsequent research.

### 3.3 Motivation estimator

The motivation estimator assigns a motivation potential to each node on the sensorimotor map. The computation of these motivation potentials can be based on intrinsic (Oudeyer et al. 2007; Oudeyer and Kaplan 2009; Schmidhuber 2006) and/or extrinsic motivation. While intrinsic motivation can enable an agent to engage in informed exploration of its environment, extrinsic motivation is useful for goal-directedness. In this paper, attention has been paid only to the goal-directed context. However, it would be interesting to examine how the habituation mechanism that is associated with the traditional GWR could be leveraged for intrinsic motivation or informed exploration. The motivation estimator in this paper is the model-based value iteration. The value iteration is a dynamic programming technique for solving Markov Decision Processes (MDPs) (Tamar et al. 2016). It iteratively updates the value function until convergence following Bellman's equation (Dai and Goldsmith 2007) as shown in Eq. 3 (Sutton and Barto 2018):

$$V_\pi(s) = \sum_a \pi(a,s) \sum_{s'} p^a_{ss'}[r^a_{ss'} + \gamma V_\pi(s')], \tag{3}$$

$\forall s \in S$, where, $V_\pi(s)$ is the value of state $s$ under a policy $\pi$; $\pi(a,s)$ is the probability of taking action $a$ while in state $s$; $p^a_{ss'}$ is the transition probability to state $s'$ when action $a$ is executed in the state $s$; $r^a_{ss'}$ is the expected reward for the transition; and $\gamma$ is the discount factor. The model-based value iteration is possible only when the transition probability is known. The sensorimotor map learning method tries to learn the transition model for a given scenario.

We will now discuss how the value iteration is applied in this paper. Given a goal state, the associated goal node $g$ can be activated on the sensorimotor map. Equation 4 shows the update rule for the propagation of the motivation potential, $V(.)$ with respect to $g$ for each node on the sensorimotor map

$$V(i) = V(i) + \beta \Delta_V(i), \forall i \in N, \tag{4}$$

where $N$ is a set of all nodes in the sensorimotor map and $\beta$ is the learning rate. $\Delta_V$ is given by Eq. 5

$$\Delta_V(i) = sim(i,g) + \gamma \max\{E[i,k]V(k)\} - V(i) \tag{5}$$

$\forall k \in K$, where $K$ is a set of all nodes in the sensorimotor neighbourhood of $i$; these are nodes that can be reached from $i$ due to actions that are possible in $i$. $sim(i,g)$ is the Gaussian similarity between node $i$ and the goal node $g$. $sim(i,g)$ is used as the immediate reward. $\gamma$ is the discount factor. Through this iterative update, higher motivation potentials are propagated to nodes that are closer to $g$, if they also provide reliable sensorimotor-links for the agent to reach the goal state.

This has been summarised in Algorithm 1. The most expensive parts of the algorithm are the sensorimotor map learning using TMGWR and the value iteration process. TMGWR learns to map $M$ data points to $N$ nodes. Unlike in SOM where the number of nodes is predefined, TMGWR adds new nodes to minimise the quantisation error. Iterating through $M$ data points while mapping them to $N$ will cost $\mathcal{O}(N \times M)$. In the worst-case scenario, TMGWR will require $N(= M)$ nodes to correctly represent $M$ data points. Hence, the time complexity of TMGWR is $\mathcal{O}(N^2)$.

Value iteration sweeps through the entire state space, while considering all possible actions to converge. In the worst-case scenario, the time complexity of the goal-directed algorithm is $\mathcal{O}(N^2 + N \times |A|)$, where $A$ is the action set. However, TMGWR uses forgetting strategy to delete aged sensorimotor-links and redundant nodes. TMGWR demonstrates to be a more efficient sensorimotor map learning algorithm than growing neural gas (GNG), grow when required network (GWR), and time grow neural gas (TGNG) (Ezenkwu and Starkey 2019b).

### 3.4 Action selector

The action selector leverages the motivation potentials computed by the motivation estimator to generate a policy that enables the agent to select actions that can transition the agent in the direction of increasing motivation potentials. The selected action vector $\vec{a}_i$ when node $i$ is the activated node on the sensorimotor map is given by Eq. 6

$$\vec{a}_i = \vec{a}_{ik_{max}}, \tag{6}$$

**Table 1** Differences between TMGWR-based algorithm vs RL algorithms

| Attribute | Model-free RL agent | Model-based RL agent | TMGWR-based agent |
|---|---|---|---|
| **environment model** | not required | must be provided by the designer | learnt from experience |
| **reward function** | defined in the state-action space | defined in the state space | defined in the sensorimotor space |

**Algorithm 1** The motivation estimator for the TMGWR-based goal-directed agent

$\mathcal{SM} \leftarrow TMGWR(\{x_j, a_j\}_{j=1}^{M})$ ▷ training sensorimotor map $\mathcal{SM}$ using TMGWR
$g \leftarrow \mathcal{SM}$.Map($x_g$) ▷ identify the goal node $g$ given a goal state $x_g$
$R_i = sim(i,g)\ \forall i \in \{1,...,\mathcal{SM}$.nodes.Size()$\}$ ▷ compute the similarity between each node $i$ in the $\mathcal{SM}$ and the goal node $g$
$ER = [\,]$ ▷ define an empty array
$V = [0] \times \mathcal{SM}$.nodes.Size() ▷ initialise the motivation potential of each node in $\mathcal{SM}$ to 0
**repeat**
  **for** node in $\mathcal{SM}$.nodes **do**
    **for** k in node.neighbourhood() **do**
      $ER.add(E[node, k] \times R[k])$ ▷ multiply the value of each neighbour of the node with the lateral connectivity between the neighbour and the node
    **end for**
    $V[node] = V[node] + \beta.(R[node] + \gamma.max\{(ER).V[k] - V[node]\})$
    $ER = [\,]$
  **end for**
**until** Convergence

where $k_{max} = argmax_k V(k), \forall k \in K$. $K$ is a set of all nodes in the sensorimotor neighbourhood of $i$ and $V(k)$ is the motivation potential of node $k$, $\forall k \in K$.

This equation means that the agent selects an action that will place it in that node in the neighbourhood of the current node that has the maximum motivation potential. At first glance, it is possible to think that the agent is short-sighted in its action selection. However, the motivation potential of each node is a function of the sensorimotor neighbourhood of the node. This ensures that the agent's decisions are influenced by its long-term satisfaction.

## 4 Comparing the TMGWR-based algorithm and the RL algorithms

Table 1 summarises the key differences between the proposed method and the conventional RL algorithms. In the model-based RL algorithm, the environment dynamics must be well understood and formulated for the algorithm to work. This is a huge limitation to the applications of the model-based RL algorithms in unstructured or unpredictable real-world environments. Unlike the model-based RL methods, the model-free RL algorithms do not require knowledge of environment dynamics to work, and as a result, they are the most applied RL algorithms in the literature. However, the absence of environment models in the model-free RL algorithms results in their high sample complexity and poor generalisation. The TMGWR-based method does not require prior design of the environment dynamics, since the agent is capable of learning it during exploration, and because it uses the environment model for planning, it exhibits good sample efficiency and generalisation as the model-based RL.

For an RL agent to learn to plan towards a given goal, a reward function is formulated to represent the relationships between the goal and the environment states or state-action pairs. In the model-based RL methods, reward functions, $R(s)$, are dependent only on environment states, while in the model-free RL algorithms, reward functions $R(s, a)$ are dependent on state-action pairs, $i.e.$, environment states and actions. Since each of these methods require that their reward functions factor in environment states in addition to goal, extra environment information is required to formulate suitable reward functions for these methods. This will pose some difficulties in unknown or unstructured environments. However, given a goal, the TMGWR-based method does not require that the environment states are well understood for planning, since the reward values are computed in the sensorimotor domain using the Gaussian similarity metric which depends on the agent's own mental model of the environment and not on the actual environment states. This completely removes the difficulty of having to understand the environment to formulate a reward function for a given task.

## 5 Experiments

### 5.1 Setup

The environment used in this paper is a $13X17$ maze (Fig. 2). Solid areas in the environment represent walls, while empty areas represent passages. The agent senses the coordinates of its current location as the sensory observations. Each state in the environment can afford the agent up to four actions: move-up, move-down, move-right, and move-left. It moves one step in the direction of the selected action if there is no obstacle; otherwise, its state remains unchanged. The same random policy applies throughout the experiment. During random exploration, the agent performs a random walk that chooses a new action with the probability of 30% if no obsta-

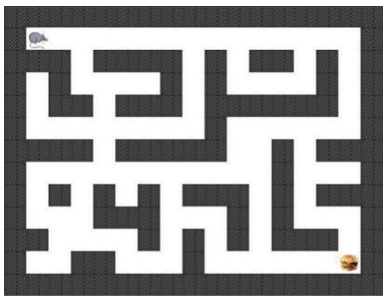


**Fig. 2** Maze with agent, i.e., rat (top left corner) and goal, i.e., food (bottom right corner)

cle is in the direction of the selected action; otherwise, it selects an arbitrary action from any of the other three actions at 100% probability. The agent begins its life-cycle with complete random exploration as described above. This random exploration reduces according to Eq. 10; each time, the agent achieves the goal after it has previously attained the goal for up to five tries. Adaptive or decayed $\epsilon$-greedy exploration (Tokic 2010; Maroti 2019) is a well-known method for handling exploration–exploitation dilemma in RL (Sutton and Barto 2018)

$$\epsilon = max(\epsilon - \frac{1}{10}, \epsilon_{min}); \tag{7}$$

$\epsilon$ is the probability that the agent will select an action based on random policy and $\epsilon_{min}$ is the minimum value that $\epsilon$ can take. In the experiment, $\epsilon$ is initialised to 1 (i.e., 100% random exploration) and decreases each time the agent achieves the goal until it attains the minimum ($\epsilon_{min} = 0.1$). Each experiment ran for ten trials.

The experiments compared three types of agent:

- The TMGWR-based agent (the proposed agent)
- A model-free RL agent (in this case, the Q-learning agent)
- The model-based RL agent using value iteration algorithm.

The reward function for the model-free RL agent is defined in Eq. 8. For the model-based RL agent, the reward function is defined as shown in Eq. 9. These immediate rewards were treated as hyperparameters and were selected by Bayesian optimisation (Pelikan et al. 1999; Grosnit et al. 2021) of the convergence rate (see Sect. 5.3)

$$r(s,a) = \begin{cases} 88, & \text{if action } a \text{ in state } s \text{ leads to goal} \\ -131, & \text{if action } a \text{ in state } s \text{ leads to wall} \\ -29, & \text{if } s \text{ action } a \text{ in state } s \text{ leads to empty} \end{cases} \tag{8}$$

$$r(s) = \begin{cases} 75, & \text{if state } s \text{ contains goal} \\ -18, & \text{if state } s \text{ is empty.} \end{cases} \tag{9}$$

While the reward functions for the RL agents are provided in the environment space, the TMGWR-based agent when given a goal computes its rewards or motivations in the sensorimotor space. This gives the method the potential to overcome the difficulty of designing reward functions for unknown environments as is the case with the traditional RL. Given the goal state, the associated goal node, $g$. can be determined from the sensorimotor map. The TMGWR-based agent derives the desirability of having each node activated based on their proximity to $g$. This is defined in Eq. 10 as follows:

$$r(i) = sim(i,g)\ , \forall i \in N. \tag{10}$$

The reward function is used to derive the motivation potential for each node as seen in Eq. 5.

For the model-based RL agent, we defined the transition dynamics, $T(s,a,s')$, as a deterministic model. $T(s,a,s')$ is the probability of the agent going into $s'$ if action $a$ is taken in state $s$. Throughout the experiment, the transition probability is hard-coded for a given arrangement of obstacles in the environment. However, this is not allowed to change automatically when the environment changes unpredictably

$$T(s,a,s') = \begin{cases} 1, & \text{if s' contains no obstacle} \\ 0, & \text{if if s' contains an obstacle.} \end{cases} \tag{11}$$

## 5.2 Descriptions of experiments

The TMGWR-based goal-directed agent, the traditional model-based RL agent, and a model-free RL agent (in this case, the Q-learning agent) are compared in three scenarios: changing goal state, changing environment, and sample complexity. While the first scenario illustrates their abilities to self-adapt when the goal changes, the second demonstrates their ability to cope when the domain knowledge about the environment no longer applies, and the third compares their sample efficiencies. The following sections describe each of these scenarios in detail.

### 5.2.1 Scenario II: changing goal state

This scenario investigates the ability of the agents to generalise when the task presents a goal other than the one used during the agents' training. The behaviour of each of the algorithms in the changing goal scenario has been simulated. Figure 3 shows four selected goal locations, $G_1$, ..., $G_4$. Each agent is first trained for 1000 iterations with the goal at $G_1$. After every 1000 iterations, the goal locations change in the sequence $G_1 \rightarrow G_2 \rightarrow G_3 \rightarrow G_4 \rightarrow G_1$. Each iteration terminates when the agent visits either the current goal state or the previous goal state. For instance, if the goal is at $G_1$ and

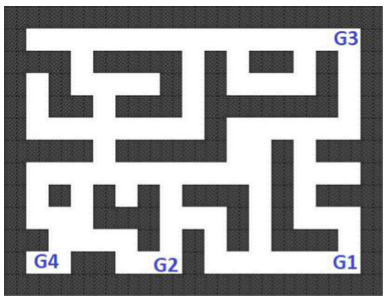



**Fig. 3** Selected goal locations. $G_1$, ..., $G_4$ are the selected goal locations, while $S$ is the starting point of the agents. This applies to scenarios I and II

later moves to $G_2$, then any visit to $G_1$ is a visit to the previous goal state and any visit to $G_2$ is a visit to the current goal state. For the RL agent, the previous goal state has the same reward as any other empty cell in the environment except that it is also a termination point. The number of times the current goal state and the previous goal state are visited beginning at the starting point are recorded separately for each 1000 iterations. We carefully chose the goal locations to reduce the possibility of any of the agents visiting the current goal state with the minimum exploration probability, $\epsilon_{min} = 0.1$ for some consecutive goal states. For example, $G_1$ and $G_2$ are the closest consecutive goal locations, while $G_3$ and $G_4$ are the farthest. While it is likely that the agent can visit $G_2$ instead of $G_1$ due to the minimum exploration probability, $\epsilon_{min} = 0.1$, it is less likely that the agent will visit $G_4$ instead of $G_3$ due to $\epsilon_{min}$.

#### 5.2.2 Scenario II: changing environment

This scenario examines the abilities of the agents to self-adapt if there is a slight change in the environment. The agents are trained on a goal until they learn the optimal path to that goal; then, an obstacle is introduced along this optimal path. For fair investigation of the self-adaptivity of the agents, we assume that the change in the environment is unpredictable and is not captured for whatever reason in the design of the agents. For example, the transition model for the model-based RL agent is modelled for the original environment and not updated when the environment changes. A self-adaptive agent will be able to adapt its world model and identify the best possible alternative to the goal. After the optimal path is blocked, the number of steps it took the agents to reach the goal for the first time using an alternative path is recorded. Figure 4 illustrates the two case studies for this scenario. The arrows in Fig. 4a, c show the optimal paths to the goals in the two case studies, while Fig. 4b, d shows the positions of the obstacles (indicated with red circles), respectively. For reference purpose, we refer to the two contexts as cases A and B, respectively.

#### 5.2.3 Scenario III: Computational cost

This scenario evaluates and compares the computational costs of the algorithms in terms of their sample complexity and amount of CPU time required by each them during training, self-adaptation, and replanning to cope with change in the environment.

#### 5.2.4 Sample complexity

This experiment compares the amount of samples each of the algorithms requires to learn the optimal policy for a given task. Starting from $S$, the sample efficiencies of the agents are investigated considering each of the four goal locations $G_1$, ..., $G_4$, as shown in Fig. 3.

At each iteration, each agent selects actions at each time step based on the prevailing policy until the goal is achieved. Each iteration terminates when the agent successfully visits the goal position. Initially, the agents are 100% exploratory, but they become more deterministic as they learn to achieve the goal. This has been explained in Sect. 5.1.

The convergence rate is a quantity of interest in estimating the sample complexity of goal-directed agents (Kakade et al. 2003) such as the RL agent. As such, this scenario considers the convergence rates of the algorithms as a measure of their sample complexities.

As presented in our previous paper, TMGWR optimises the sensorimotor map building process using the minimal number of nodes that best represent an environment (Ezenkwu and Starkey 2019b). Fewer nodes contribute to efficient convergence during planning, because the agent will not waste time exploring unnecessary nodes. For the sake of self-recoverability, when a condition in an agent's environment changes, efficient sample complexity is of the essence in enabling the agent to learn faster and respond to the changes in real-time.

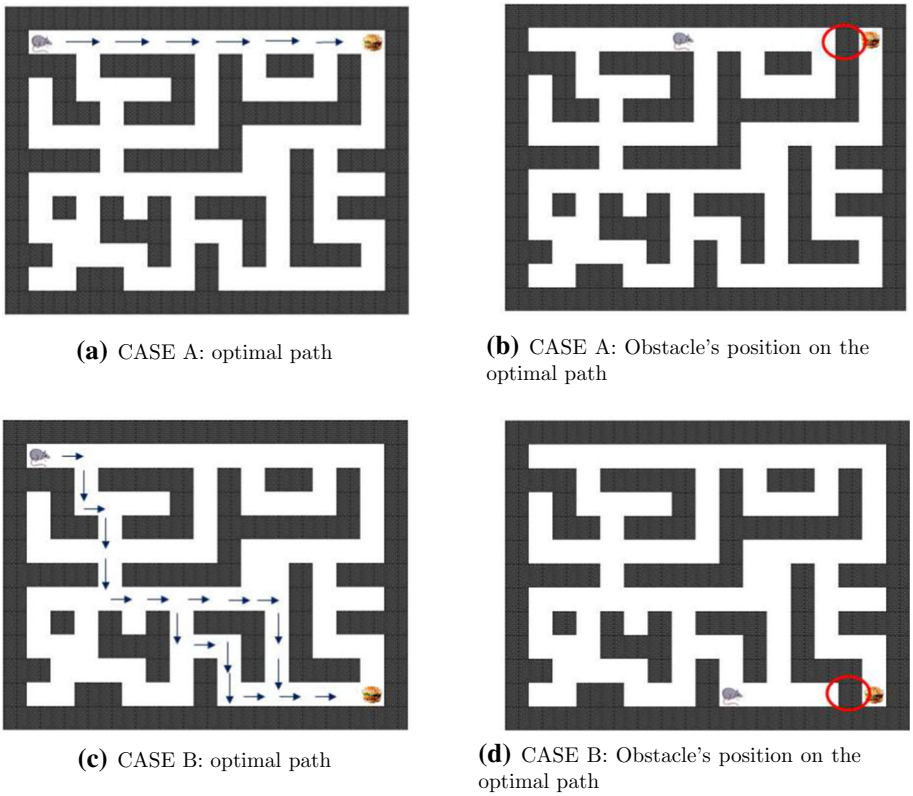
#### 5.2.5 Time-based comparisons of the algorithms

The times required by each of the algorithms for training, for self-adaptivity when the goal changes, and for adjustments to changing environments have been recorded and compared in this experiment.

The algorithms are run on Python version 3.8 on a 64-bit Windows 10 computer with Intel(R) Core(TM) i5-4570 CPU @ 3.20GHz 3.20GHz and 16.0GB (15.9GB usable) RAM. For evaluating the times required for training the algorithms until convergence, the environment of Fig. 2 is used for training each algorithm. The time between the start of the training and the time the agent consistently follows the optimal path ten consecutive times is recorded for each algorithm.

To evaluate the times the algorithms self-adapt to a new goal, the algorithms are trained to learn the optimal path with

**Fig. 4** Changing environments



**(a)** CASE A: optimal path

**(b)** CASE A: Obstacle's position on the optimal path

**(c)** CASE B: optimal path

**(d)** CASE B: Obstacle's position on the optimal path

the goal at $G_4$ as indicated in the environment of Fig. 3, and then, the goal is repositioned to $G_3$ (also see Fig. 3). The time it takes the algorithm to adapt to execute the optimal path to the new goal location is recorded for each algorithm.

Figure 4a, b has been useful in evaluating the response time of each algorithm to a change in the environment. Figure 4a demonstrates the optimal path as identified by the algorithm after training, whereas Fig. 4b shows an obstacle blocking this optimal path. Using these two configurations, the time it takes each algorithm to access the goal state through an alternative path after the initial optimal path is blocked is recorded and reported for the algorithm.

In each experiment, ten different trials are executed and results are presented as bar charts.

### 5.3 Hyperparameters

The hyperparameters for the three methods were selected by the Bayesian optimisation(Pelikan et al. 1999; Grosnit et al. 2021) of their convergence rates in attaining the goal $G_1$ starting from $S$. The hyperopt library in Python[1] was used with maximum number of evaluations set to 150. The algorithms were optimised independently in the environment of Fig. 2 following the exploration policy described above.

[1] http://hyperopt.github.io/hyperopt/

The TMGWR network has been previously optimised in a similar environment in our previous work (Ezenkwu and Starkey 2019b). Moreover, in the TMGWR-based agent, the sensorimotor map learning algorithm is loosely coupled with the planning algorithm. Due to the following reasons, it is easy to transfer hyperparameters from the original TMGWR network to the version applied in this paper by freezing the hyperparameters that are connected to the sensorimotor map learning while optimising those for the value iteration and the sensorimotor-link adaptivity. Table 2 presents the hyperparameters used in the paper; $\gamma$ is the discount factor, $\beta$ is the learning rate for the value iteration and the Model-free RL, $\alpha$ is the learning rate for the sensorimotor-link adaptation equation, and $r(s)$ is the immediate reward for state $s$ where $s$ can be an empty cell, a cell with wall, or the goal state.

## 6 Results and discussion

### 6.1 Scenario I: changing goal state

Tables 3, 4, and 5 present the responses of the model-free RL agent, the TMGWR-based agent, and the model-based RL agent to changes in goals, respectively. As described in Sect. 5.2.1, this scenario compares the generalisation abilities of the three agents when the goal state changes. Apart

**Table 2** Hyperparameters

| Hyperparameter | TMGWR-based agent | Model-free RL agent | Model-based RL agent |
|---|---|---|---|
| $\gamma$ | 0.97 | 0.86 | 0.94 |
| $\beta$ | 0.80 | 0.39 | 0.23 |
| $\alpha$ | 0.52 | – | – |
| $r(s = wall)$ | – | −131 | – |
| $r(s = empty)$ | – | −29 | −18 |
| $r(s = goal)$ | – | 88 | 75 |

from the change from $G_1$ to $G_2$ where the model-free RL agent visited the current goal state up to 89.26% of the time, it spent over 99% of the time visiting the previous goal states for the other change in goal states. For $G_1 \rightarrow G_2$, the model-free RL agent was likely to encounter the current goal state with the minimum exploration due to the close proximity between $G_1$ and $G_2$. Moreover, the model-free RL agent did not need to relearn every aspect of its model of the task to be able to attain $G_2$. For example, the value function that can drive the agent to the bottom of the maze is still useful for getting it to the $G_2$ with slight update. Contrarily, the other changes in goal states require that a large part of the agent's model of the task is updated. This update is slow due to low exploration probability making it spend more time exploiting the previous goal state before learning to attain the current goal. Compared to the model-free RL agent, the TMGWR-based agent and the model-based RL agent quickly respond correctly to all changes in goal states. This is because contrary to the model-free RL agent, the TMGWR-based and the model-based RL agents plan with some knowledge of the environment and are therefore, able to quickly replan to attain any goal state within the environment. However, while the TMGWR-based agent replans on the sensorimotor map using a learnt environment model, the model-based RL agent replans using a defined environment-dependent transition model. Short demonstrations of this scenario with the Model-free RL[2], TMGWR-based[3], and the Model-based RL [4] agents are available on YouTube for view.

## 6.2 Scenario II: changing environment

The results of this scenario have been summarised in Table 6 and Fig. 5. In the two cases, the model-free RL agent took a greater number of steps to reach the goal than the TMGWR-based agent after an obstacle is introduced along the optimal path. This is because the model-free RL agent tries to learn the optimal path to attaining a specific goal. As such, it has to relearn its value function if the environment changes in a way that does not avail it of that path to the goal. Although they are complementary, for the TMGWR-based agent, the environment modelling is separate from the planning mechanism. While the sensorimotor map learning technique is responsible for the environment modelling, the value iteration does the planning. The TMGWR algorithm continually adapts the sensorimotor map to reflect changes in the agent's world at every time stamp. This enables the TMGWR-based agent to keep track of the changes in the environment significantly more quickly than the model-free RL agent. With the correct sensorimotor map, a sample-efficient model-based value iteration algorithm can help to propagate the motivation potentials on the sensorimotor map in such a way that enables the agent to desire an alternative path to the current goal. The model-based RL agent showed the worst performance in this scenario. This is because it requires an environment-dependent transition model, and unlike the TMGWR-based

**Table 3** Response of the model-free RL agent to change in goal state for four selected goal locations, $G_1$, ..., $G_4$

| Change of goal state | %number of visits to previous goal | % number of visits to current goal |
|---|---|---|
| $G_1 \rightarrow G_2$ | $10.64 \pm 16.16$ | $89.36 \pm 16.16$ |
| $G_2 \rightarrow G_3$ | $99.40 \pm 0.32$ | $0.60 \pm 0.32$ |
| $G_3 \rightarrow G_4$ | $99.78 \pm 0.04$ | $0.22 \pm 0.04$ |
| $G_4 \rightarrow G_1$ | $99.58 \pm 0.19$ | $0.42 \pm 0.19$ |

**Table 4** Response of the TMGWR-based agent to change in goal state for four selected goal locations, $G_1$, ..., $G_4$

| Change of goal state | % number of visits to previous goal | % number of visits to current goal |
|---|---|---|
| $G_1 \rightarrow G_2$ | $0 \pm 0$ | $100 \pm 0$ |
| $G_2 \rightarrow G_3$ | $0 \pm 0$ | $100 \pm 0$ |
| $G_3 \rightarrow G_4$ | $0 \pm 0$ | $100 \pm 0$ |
| $G_4 \rightarrow G_1$ | $0 \pm 0$ | $100 \pm 0$ |

**Table 5** Response of the model-based RL agent to change in goal state for four selected goal locations, $G_1$, ..., $G_4$

| Change of goal state | % number of visits to previous goal | % number of visits to current goal |
|---|---|---|
| $G_1 \rightarrow G_2$ | $0 \pm 0$ | $100 \pm 0$ |
| $G_2 \rightarrow G_3$ | $0 \pm 0$ | $100 \pm 0$ |
| $G_3 \rightarrow G_4$ | $0 \pm 0$ | $100 \pm 0$ |
| $G_4 \rightarrow G_1$ | $0 \pm 0$ | $100 \pm 0$ |

[2] Demonstration: response of the model-free RL agent to change in goal state: https://www.youtube.com/watch?v=_j0z6B1RFjs

[3] Demonstration: response of the TMGWR-based agent to change in goal state: https://www.youtube.com/watch?v=x9U0r-6Sct0

[4] Demonstration: response of model-based RL agent to change in goal state: https://youtu.be/4GNbxYvJPhM

**Table 6** Average number of steps it took each agent to attain the goal through an alternative path after the optimal path was blocked

| Case | TMGWR-based agent | Model-free RL agent | Model-based RL agent |
|---|---|---|---|
| *A* | $112.6 \pm 2.0$ | $1212.2 \pm 12.9$ | $\infty$ |
| *B* | $106.3 \pm 1.6$ | $450.4 \pm 4.9$ | $\infty$ |

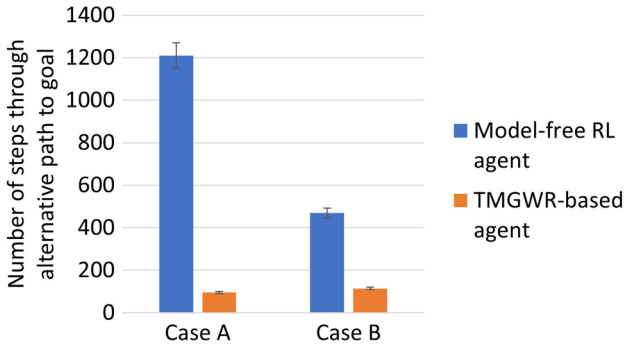



**Fig. 5** Average number of steps it took each agent to attain the goal through an alternative path after the optimal path was blocked. Values for the model-based Rl agent are not shown in the graph, because they are $\infty$ in the two cases

agent, it neither learn its transition model nor adapt it to changes in the environment unless this changes are known *a priori* and are accounted for during the design. Because of this limitation of the model-based RL agent, it get stuck in the original path to the goal when the environment changes in the two cases (see short demonstration for the responses of the Model-free RL[5], TMGWR-based [6] and model-based RL[7]agents in the changing environment scenarios)

#### 6.2.1 Scenario III: computational cost

#### 6.2.2 Sample complexity

Figure 6 displays the results of the experiment of scenario I. The graphs compare the sample efficiencies of the three methods considering the four goal locations.

The number of steps for each agent to attain the goal locations are recorded at every 5th iteration. Figure 6 shows that the TMGWR-based agent and the traditional model-based RL agent converge to the optimal $\epsilon$–greedy policy faster than the model-free RL agent for the four goal locations.

This advantage of the TMGWR-based agent and the traditional model-based RL agent is derived from their model-based attribute. While the model-free RL agent tries to learn the decision model for the task without the knowledge of the environment dynamics, the model-based RL agent uses the transition dynamic defined in Eq. 11 to plan towards attaining the goal in a more sample-efficient manner.

Similar to the model-based RL agent, the TMGWR-based agent also shows a sample-efficient planning but using a learned and self-adaptive transition model of the environment.

#### 6.2.3 Time-based comparisons of the algorithms

Figure 7 demonstrates how much time is required by each algorithm for training, for self-adaptivity when goal changes, and to adjust to changing environments.

Although the model-free RL agent requires a higher number of explorations during training, Fig. 7a shows that it requires a minimal amount of training time compared to the TMGWR-based agent and the traditional model-based agent. This minimal training time is because the model-free RL agent is lightweight and involves fewer computations than the other two methods, which repetitively utilise the environment model during planning. The effect of this lack of task model by the model-free RL is that any change in goal state requires a complete overwriting of the existing knowledge of the task that the agent has acquired so far. This demonstrates why in Fig. 7b the model-free RL agent takes a relatively long time, when compared to the other two agents, to adapt in solving the task if the goal changes.

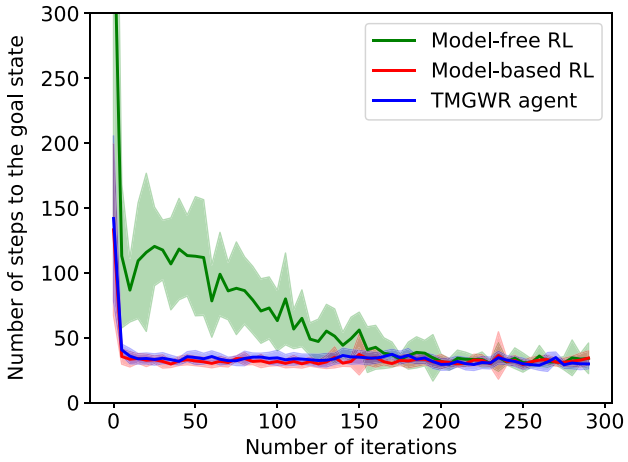
Each step in the TMGWR-based algorithm's learning process involves the identification of the best matching unit, the adaptation of the map in response to the current context, the computation of the motivation potentials, and then value iteration for planning. These computations slow down the training of the algorithm, although they will prove to be useful as will be shown later in this section. The model-based RL equally has a quicker training time than the TMGWR-based algorithm, because the environment model and the rewards are hard-coded. Therefore, unlike in the TMGWR-based agent, model-based RL does not have to learn any model of the environment or compute any motivation potentials. The aforesaid makes it have a faster training time than the TMGWR-based agent.

However, one disadvantage is that it is rigid and does not adapt to changes in the environment as can be seen in Fig.

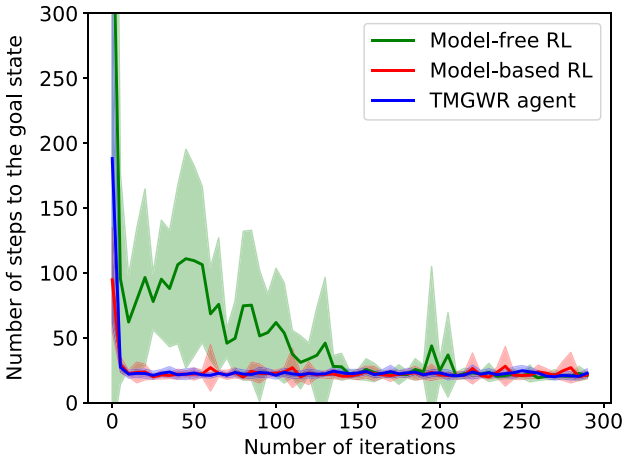
[5] Demonstration: response of the model-free RL agent to change in the environment: https://youtu.be/aRr4Ja9TspQ

[6] Demonstration: response of the TMGWR-based agent to change in the environment: https://youtu.be/-YpxGEjRoXA

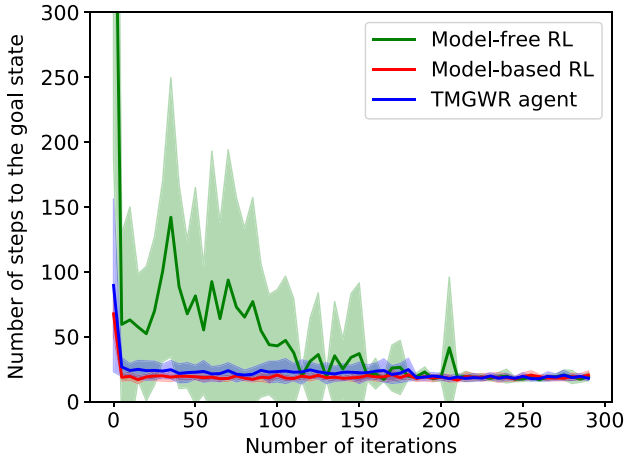
[7] Demonstration: response of model-based RL agent to change in the environment: https://www.youtube.com/watch?v=peEYriVEK2k



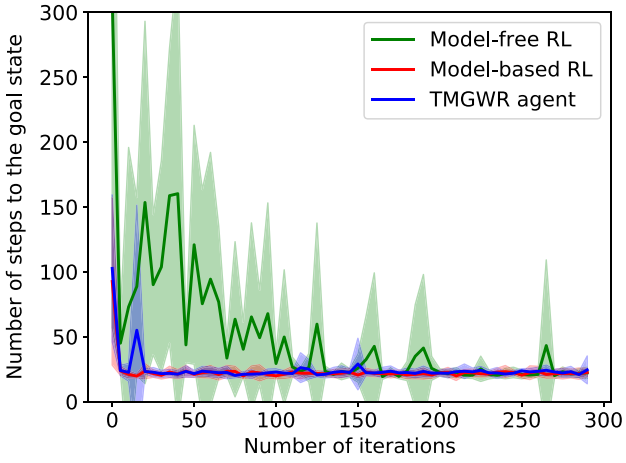

**(a)** Convergence rates of the algorithms with goal at $G1$.




**(b)** Convergence rates of the algorithms with goal at $G2$.




**(c)** Convergence rates of the algorithms with goal at $G3$.




**(d)** Convergence rates of the algorithms with goal at $G4$.

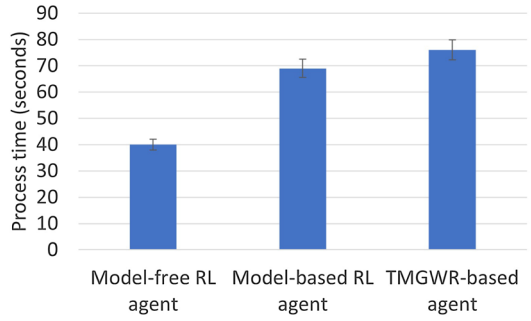
**Fig. 6** Result of scenario I: sample complexity

7c. The model-based RL stays trapped forever when the environment dynamics change in a manner not captured by the experimenter—because of this, no time is recorded for it in this experiment. While the model-free agent manages to find an alternative path to the goal after the optimal path has been blocked, it does that more than three times slower than the TMGWR-based agent. This is because while model-free RL needs to relearn every aspect of its knowledge, the TMWGR-based agent only has to adapt an aspect of its sensorimotor map that associates with the current experience to correctly solve the task. Figure 7d summarises the times taken by each of the algorithms in the three scenarios. The result demonstrates that the TMGWR-based agent has the lowest average time. The model-based agent has no value recorded for it, because it takes infinity to recover from a change in the environment.

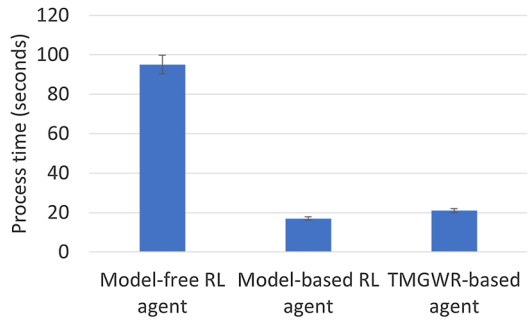
## 7 Conclusions

This paper proposes an autonomous agent architecture that employs the TMGWR network for continuous sensorimotor map learning and uses value iteration to plan to attain a goal using the sensorimotor map. The proposed method computes rewards or motivation potentials in the sensorimotor space and learns the transition model of its environment using the TMGWR network as the sensorimotor map learning algorithm. For the agent to be goal-directed, value iteration helps to propagate the motivation potentials associated with the goal in the sensorimotor space. A modification has been made to the original TMGWR algorithm to enable the agent to encode reliability and discourage the selections of actions that can compromise its progress.

The model-free RL, traditional model-based RL, and the TMGWR-based agents have been evaluated on the basis




**(a)** Times required for training each algorithm.




**(b)** Time required for each algorithm to adapt to a new goal state.


**(c)** Time required for each algorithm to successfully respond to a change in the environment. **Note:** There is no value for the model-based agent because it never recovered from a change in the environment.


**(d)** Average of the times for the three scenarios.

**Fig. 7** Time-based comparisons of the algorithms

of their abilities to self-adapt when the goal changes, to cope when the domain knowledge about the environment no longer applies, and their computational efficiencies. Since the traditional model-based RL requires that the environment dynamics be hard-coded, it fails to recover from a sudden change in the environment unlike the TMGWR-based agent and the model-free RL agent. This is because the sensorimotor map learning potential of the TMGWR-based agent enables it to keep track of changes in the environment, while the model-free agent can also adapt by modifying its state-action space. On the contrary, the model-free agent does not always adapt to a change in goal state, often revisiting the previous goal, whereas TMGWR and model-based RL are both able to avoid previous goal states and adapt immediately to the new goal. The experiments demonstrate that only the TMGWR-based agent has the potential to self-adapt efficiently in dynamic contexts, for both a change in goal or a change in environment.

Furthermore, the experiments examined the computational efficiencies of the algorithms—both their sample complexities and the time-based evaluations of their training and self-adaptivity, for a change in goal state or a change in domain knowledge. The results demonstrate that the TMGWR-based agent has a similar sample complexity to the traditional model-based RL agent, and significantly better sample complexity than the model-free RL agent. However, the model-free agent has a quicker training time than the other two algorithms, because it is lightweight and does not factor in the model of the environment during its learning, an attribute that impacts its ability to self-adapt and cope efficiently with a change in the environment. The model-based RL agent cannot adapt at all to a change in the environment, because it depends on hard-coded domain knowledge and needs that it be manually modified to suit the current domain requirements. The experiments demonstrate that only the TMGWR-based agent has the potential to self-adapt efficiently in dynamic contexts, for both a change in goal or a change in environment and to do so with the lowest overall computational cost.

**Acknowledgements** This work is funded by the Tertiary Education Trust Fund (TETFund) scheme of the Federal Republic of Nigeria.

**Data availability** The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request.

## Declarations

**Conflict of interest** The authors declare that they have no conflict of interest.

## References

Bellman R (1952) On the theory of dynamic programming. Proc Natl Acad Sci USA 38:716

Belousov B, Abdulsamad H, Klink P, Parisi S, Peters J (2021) Reinforcement learning algorithms: analysis and applications. Springer, New York

Berridge KC, Robinson TE, Aldridge JW (2009) Dissecting components of reward:'liking",wanting', and learning. Curr Opin Pharmacol 9:65–73

Bozkurt AK, Wang Y, Zavlanos MM, Pajic M (2021) Model-free reinforcement learning for stochastic games with linear temporal logic objectives. In: 2021 IEEE International Conference on Robotics and Automation (ICRA), IEEE, pp 10649–10655

Chaput H.H (2004) The constructivist learning architecture: A model of cognitive development for robust autonomous robots. Ph.D. thesis

Dai P, Goldsmith J (2007) Topological value iteration algorithm for markov decision processes. In: IJCAI, pp 1860–1865

Drescher GL (1991) Made-up minds: a constructivist approach to artificial intelligence. MIT Press

Dulac-Arnold G, Levine N, Mankowitz DJ, Li J, Paduraru C, Gowal S, Hester T (2021) Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. Machine Learning, pp 1–50

Ezenkwu C.P, Starkey A (2019a). Machine autonomy: Definition, approaches, challenges and research gaps, in: Intelligent Computing-Proceedings of the Computing Conference, Springer, New York, pp 335–358

Ezenkwu CP, Starkey A (2019b) Unsupervised temporospatial neural architecture for sensorimotor map learning. In: IEEE transactions on cognitive and developmental systems

Ge J, Liu B, Wang T, Yang Q, Liu A, Li A (2021) Q-learning based flexible task scheduling in a global view for the internet of things. Trans Emerg Telecommun Technol 32:e4111

Grosnit A, Cowen-Rivers AI, Tutunov R, Griffiths RR, Wang J, Bou-Ammar H (2021) Are we forgetting about compositional optimisers in bayesian optimisation? J Mach Learn Res 22:1–78

Guerin F, Starkey A (2009) Applying the schema mechanism in continuous domains. In: Proceedings of the Ninth International Conference on Epigenetic Robotics, pp 57–64

Hakimzadeh A, Xue Y, Setoodeh P (2021) Interpretable reinforcement learning inspired by piaget's theory of cognitive development. arXiv:2102.00572

Huitt W, Hummel J (2003) Piaget's theory of cognitive development. Educ Psychol Interact 3:1–5

Irpan A (2018) Deep reinforcement learning doesn't work yet. Online (Feb. 14): https://www.alexirpan.com/2018/02/14/rl-hard.html

Kakade SM, et al., (2003) On the sample complexity of reinforcement learning. Ph.D. thesis. University of London London, England

Kohonen T (1990) The self-organizing map. Proc IEEE 78:1464–1480

Liu C, Goel P, Kaeser PS (2021) Spatial and temporal scales of dopamine transmission. Nat Rev Neurosci 22:345–358

Marcus G (2018) Deep learning: A critical appraisal. arXiv preprint arXiv:1801.00631

Maroti A (2019) Rbed: Reward based epsilon decay. arXiv preprint arXiv:1910.13701

Nguyen PD, Georgie YK, Kayhan E, Eppe M, Hafner VV, Wermter S (2021) Sensorimotor representation learning for an "active self" in robots: a model survey. KI-Künstliche Intelligenz 35:9–35

Oudeyer PY, Kaplan F (2009) What is intrinsic motivation? a typology of computational approaches. Front Neurorobot 1:6

Oudeyer PY, Kaplan F, Hafner VV (2007) Intrinsic motivation systems for autonomous mental development. IEEE Trans Evol Comput 11:265–286

Parisi GI, Wermter S (2013) Hierarchical som-based detection of novel behavior for 3d human tracking, in: The 2013 international joint conference on neural networks (IJCNN), IEEE. pp. 1–8

Pelikan M, Goldberg DE, Cantú-Paz E et al.,(1999) Boa: The bayesian optimization algorithm, in: Proceedings of the genetic and evolutionary computation conference GECCO-99, Citeseer. pp. 525–532

Piaget J, Cook M (1952) The origins of intelligence in children, vol 8. International Universities Press, New York

Pierris G, Dahl TS (2017) Learning robot control using a hierarchical som-based encoding. IEEE Transactions on Cognitive and Developmental Systems 9:30–43

Razavian AS, Azizpour H, Sullivan J, Carlsson S (2014) CNN features off-the-shelf: an astounding baseline for recognition. CoRR abs/1403.6382. arXiv:1403.6382,

Saba D, Sahli Y, Maouedj R, Hadidi A, Medjahed MB (2021) Towards artificial intelligence: Concepts, applications, and innovations, in: Enabling AI Applications in Data Science.Springer, pp. 103–146

Schmidhuber J (2006) Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts. Connect Sci 18:173–187

Sermanet P, Xu K, Levine S (2016) Unsupervised perceptual rewards for imitation learning. arXiv preprint arXiv:1612.06699

Strickert M, Hammer B (2005) Merge som for temporal data. Neurocomputing 64:39–71

Sutton RS, Barto AG (2018) Reinforcement learning: An introduction. MIT press

Szepesvári C (2010) Algorithms for reinforcement learning. Synthesis lectures on artificial intelligence and machine learning 4:1–103

Tamar A, Wu Y, Thomas G, Levine S, Abbeel P (2016) Value iteration networks, in: Advances in Neural Information Processing Systems, pp. 2154–2162

Tokic M (2010) Adaptive $\varepsilon$-greedy exploration in reinforcement learning based on value differences, in: Annual Conference on Artificial Intelligence, Springer. pp. 203–210

Toussaint M (2004) Learning a world model and planning with a self-organizing, dynamic neural system, in: Advances in neural information processing systems, pp. 926–936

Toussaint M (2006) A sensorimotor map: Modulating lateral interactions for anticipation and planning. Neural Comput 18:1132–1155

Tsou JY (2006) Genetic epistemology and piaget's philosophy of science: Piaget vs. kuhn on scientific progress. Theory & Psychology 16:203–224

Vamvoudakis KG, Antsaklis PJ, Dixon WE, Hespanha JP, Lewis FL, Modares H, Kiumarsi B (2015) Autonomy and machine intelligence in complex systems: A tutorial, in: 2015 American Control Conference (ACC), IEEE. pp. 5062–5079

Wang S, Chen J, Guo W, Liu G (2019) Structured learning for unsupervised feature selection with high-order matrix factorization. Expert Systems with Applications , 112878

Yang Y, Kiumarsi B, Modares H, Xu C (2021) Model-free $\lambda$-policy iteration for discrete-time linear quadratic regulation. IEEE Transactions on Neural Networks and Learning Systems

Yosinski J, Clune J, Bengio Y, Lipson H (2014) How transferable are features in deep neural networks?, in: Advances in neural information processing systems, pp. 3320–3328