Simultaneous Learning of Spatial Visual Attention and Physical Actions

Ali Borji Majid Nili Ahmadabadi Babak Nadjar Araabi

***Abstract*— This paper introduces a new method for learning top-down and task-driven visual attention control along with physical actions in interactive environments. Our method is based on the Reinforcement Learning of Visual Classes(RLVC) algorithm and adapts it for learning spatial visual selection in order to reduce computational complexity. Proposed algorithm also addresses aliasings due to not knowing previous actions and perceptions. Continuing learning shows our method is robust to perturbations in perceptual information. Our method also allows object recognition when class labels are used instead of physical actions. We have tried to gain maximum generalization while performing local processing. Experiments over visual navigation and object recognition tasks show that our method is more efficient in terms of computational complexity and is biologically more plausible.**

I. INTRODUCTION

Similar to humans and animals, artificial creatures like robots are limited in terms of allocating their resources to huge perceptual information. That is mainly because of the serial processing mechanisms used in the design of such creatures which allows processing of only a small amount of incoming sensory information at any given time. Visual attention mechanisms serve to implement a bottleneck through which only informative and relevant information are allowed to pass to higher level processing centers. Since robotic agents are usually supposed to guarantee a limited response time, attention is an efficient solution in this area as in biological creatures.

To perform a task, agents should be able to percept the environment and perform appropriate physical actions. Perceptual actions are available in several forms like where and what to look in visual modality. The main concern in learning attention is how to select the relevant information, since relevancy depends on the tasks and goals. In this study, we consider task relevancy of visual attention and aim to extract spatial locations which help the agent to achieve its goals faster.

It is important that a solution for learning task-based visual attention control to take into account other relevant and interleaved cognitive processes like learning, decision making, action selection, etc. There are several biological evidences for this. It has been previously shown that attention and eye movements are context-dependent and task-specific [1]. Previous experiences also influence attentional behaviors which indicate that attention control mechanisms can be learned [2]. Some neuropsychological evidences suggest that human beings learn to extract useful information from visual scenes in an interactive fashion without the aid of any external supervisor [3][4]. Instead of attempting to segment, identify, represent and maintain detailed memory of all objects in a scene, there are evidences that claim our brain may adopt a need-based approach [5], where only desired objects are quickly detected, identified and represented. Considering above evidences, in this work, we introduce a model to consider the influences of task, action, learning and decision making to control top-down visual attention.

From another perspective learning top-down attention is highly coupled with learning representations. Therefore the best way to derive visual attention mechanisms is to learn them in concert with visual representations. This is tightly relevant to an area of research known as state space discretization in reinforcement learning. Here we adapt these techniques for deriving spatial visual attention in interactive environments which is like the saccadic eye movements that humans and animals perform to recognize a scene.

Reinforcement learning (RL) is a general framework for modeling the behavior of an agent that learns how to perform its task through its interactions with the environment. The only information that the agent takes in response to its actions is a reinforcement signal instead of being told that its action has been right or wrong. Like many existing learning methods, RL suffers from the curse of dimensionality, requiring a large number of learning trials as state-space grows. But it has the ability to handle dynamic and non-deterministic environments. It is believed that curse of dimensionality can be lessened to a great extent by implementation of state abstraction methods and hierarchical structures. Moreover, incremental improvement of agents performance becomes much simpler due to less number of states.

Several approaches for interactive discretization of state space have been proposed. Techniques using a non-uniform discretization are referred to as variable resolution techniques [6]. The parti-game algorithm [7] is an algorithm for automatically generating a variable resolution discretization of a continuous, deterministic domain based on observed data. This algorithm uses a greedy local controller and moves within a state or between adjacent states in the discretization. When the greedy controller fails, the resolution of the discretization is increased in that state. The G algorithm [8], and McCallum's U-Tree algorithm [9], are similar algorithms that automatically generate a variable resolution discretization by re-discretizing propositional techniques. Like parti-game,

A. Borji is with Department of Computer Science, University of Southern California, Hedco Neuroscience Building - Room 9, 3641 Watt Way, Los Angeles, California, 90089-2520, USA. borji@usc.edu

M. N. Ahmadabadi and B. N. Araabi are with the School of Electrical and Computer Engineering, University of Tehran and also School of Cognitive Sciences, IPM, Tehran, Iran. {mnili, araabi}@ut.ac.ir

they both start with the world as a single state and recursively split it when necessary. The continuous U-Tree algorithm described in [10], extends these algorithms to work with continuous state spaces.

Jodogne et al. [11] have proposed an approach for learning action-based image classification known as Reinforcement Learning of Visual Classes (RLVC). RLVC consists of two interleaved learning processes: an RL unit which learns image to action mappings and an image classifier which incrementally learns to distinguish visual classes. RLVC could be regarded as a feature based attention method in which the entire image has to be processed to find out whether a specific visual feature exists or not in order to move in a binary decision tree. Like RLVC, our approach proposed in [12] extends the U-Tree to visual domain. Our approach tackles weaknesses of the RLVC, the exhaustive search for a local descriptor over the entire image and relying only a single feature which makes it susceptive occlusions, by computing and searching local descriptors at few spatial locations. Once aliasing is detected a spatial location is selected which reduces the perceptual aliasing the most. This local visual processing results in very fast scene recognition because features are extracted in small regions and there is no need for exhaustive search for a feature. In this paper, we extend our previous solution to tackle other aliasings due to not knowing the previous actions and observations. We also show that it could be applied for object recognition. Perturbation of perceptual data while learning is also studied.

II. Proposed Approach

In our method, attention is directed toward spatial circular regions. An attention tree (Saccade-Tree) is incrementally built from the incoming visual inputs. In each node of this tree, visual content at the focus of attention (FOA) is inspected. To encode the visual content at the focus of attention we use the SIFT descriptors which have shown to be very useful for object and scene recognition and image stitching. Before learning Saccade-Tree, SIFT features are grouped into some clusters.

A. Clustering Local Descriptors

Sequential attention in our method shifts the focus of attention toward the discriminant spatial locations. Final output of the algorithm is a scanpath of eye movements. There is no need that the pattern at each FOA to be represented in fine detail, but an approximate characterization also suffices to discriminate among objects and scenes. Let $f: P \rightarrow T$ be a mapping from the set of focuses of attention P to a discrete set of features T. Here, f returns the class of a specific SIFT feature at a member of P which is a circular region with radius r. In order to derive a rough local descriptor representation, SIFT features of some random images $Q = \left\{q_1, q_2, \cdots, q_{|Q|}\right\}$ are extracted and then clustered using standard k-means clustering algorithm. Therefore a set of $|T|$ clusters $T = \left\{\tau_1, \tau_2, \cdots, \tau_{|T|}\right\}$ hereafter called codebooks are generated. The specific feature that f returns its codebook could be the closest SIFT to the center of FOA, f_1, or the SIFT feature with highest magnitude among SIFT features at FOA, f_2:

$$\begin{aligned} f_1 &= argmin_j \parallel SIFT_{FOA,k} - \tau_j \parallel, \\ & \quad k \text{ is the closest SIFT to FOA center} \\ f_2 &= argmin_j \parallel SIFT_{FOA,k} - \tau_j \parallel, \\ & \quad k = argmax_i \parallel SIFT_{FOA,i} \parallel \end{aligned} \tag{1}$$

B. Learning Saccade Tree

An efficient way to implement attention and state space construction is by means of tree data structures. They are readable by humans and are hierarchy structured which makes them a suitable mean for deriving state-space abstraction and inclusion of the available knowledge for an RL agent. Visual discretization is performed via saccade tree whenever aliasing occurs. Such refinement is performed to increase the cumulative reward of the agent. Each internal node of the Saccade-Tree proposes a single saccade toward a specific spatial location. Edges below a node test the codebook of a SIFT feature. Based on the observed codebook, next saccade is initiated until a leaf node is reached. Leaves point to states in the Q-table. Pruning is done when algorithm ends for instance by merging the nodes with the same best actions or replacing nodes with all their leaves having the same best actions.

Saccade-Tree is incrementally built in a quasi-static manner in two phases: 1) *RL-fixed (Tree-update)* phase and 2) *Tree-fixed (RL-update)* phase. The algorithm starts with one node in the Tree-fixed phase. In each phase of the algorithm external feedback of the critic, in the form of a scalar reward or punishment, is alternatively used to update the Q-table or refine the attention tree. Initially a tree with a single node is created and all the images are mapped to that node. Evidently, such a single state is not enough and aliasing occurs. Then, the algorithm breaks the node into a number of leaves based on some gathered experiences under it. In each Tree-fixed phase, RL algorithm is executed for a number of episodes by following a ϵ-greedy or soft-max action selection policy. In this phase, tree is hold fixed and the derived quadruples $(s_t, a_t, r_{t+1}, s_{t+1})$ are only used for Q-table update according to Q-learning update rule:

$$\begin{aligned} Q_(s_t, a_t) \quad = \quad & (1-\alpha)\, Q(s_t, a_t) + \\ & \alpha(r_{t+1} + \gamma max_a Q(s_{t+1}, a)) \end{aligned} \tag{2}$$

Attention control and state discretization occur in the RL-fixed phase. An important point here is that the agent only accesses the environment through its visual sensor (e.g. its CCD camera), therefore in order to determine its state, it has to traverse its saccade tree from the root node down to a leaf, which determines its state s_t at time t. In the current state, the agent performs an action according to its learned policy. At this point, based on the received reward and the next captured image, which leads to state s_{t+1}, the agent calculates perceptual aliasing for this input. After each RL-fixed phase, nodes with aliasing are detected and expanded

TABLE I

ALGORITHM 1: SACCADE TREE MAIN FUNCTION

1: *tree* $\leftarrow$ create a tree with a single node
2: repeat
3: for $i = 1$ to *maxEpizodes* do
4: $I_t = takeImage()$
5: $s_t = traverseTree(tree, I_t)$
6: $[s_{t+1}, r_{t+1}, a_t] = selectAction(s_t)$
7: $\Delta_t = calcDelta(s_t, a_t, s_{t+1}, r_{t+1})$
8: $mem(s_t) = gatherMem(tree, I, a_t, \Delta_t)$
9: end for
10: for $j = 1$ to $|S|$ do
11: if $|mem(s_j)| >$ *memThreshold* and *checkAliasing*(s_j)
12: tree = $modifyTree(tree, s_j)$
13: end if
14: end for
15: for $i = 1$ to *maxEpizodes* do
16: $I_t = takeImage()$
17: $s_t = traverseTree(tree, I_t)$
18: $[s_{t+1}, r_{t+1}, a_t] = selectAction(s_t)$
19: $Q - table = updatePolicy(Q - table, s_t, s_{t+1}, r_{t+1}, a_t)$
20: end for
21: until (*no aliasing*) or (*max iterations*)
22: $postprune(tree)$

in order to reduce aliasing. After expanding aliased states, leaf nodes without patterns in their memory are deleted. It is worth noting that memories of leaf nodes after each RL-fixed phase are deleted too. The whole process of Saccade-Tree is shown in the Table I.

C. Measuring Aliasing

After each RL-fixed phase, algorithm refines leaves with perceptual aliasing. In order to estimate aliasing, a number of items must be accumulated under a leaf node. This is done by the agent performing some episodes running the current policy learned at the previous Tree-fixed phase. An image is captured, saccade tree is traversed in order to find the perceptual state, appropriate action is performed and a reward is received. An efficient measure of perceptual aliasing in a state (leaf node) is the TD error and can be derived from Q-learning formula as follows:

$$\begin{aligned} Q_(s_t, a_t) &= \alpha(r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \\ &\quad + Q(s_t, a_t) \\ &= \alpha\Delta_t + Q(s_t, a_t) \end{aligned} \tag{3}$$

In order to detect aliasing, all memory items under a node are clustered according to their physical actions and then if any of these clusters has a variance in Δ_t greater than a threshold (*aliasingThreshold*), then that node has aliasing at least with respect to one action. Δ_t converge to zero as the RL algorithm converges when there is no further perceptual aliasing. This is when the transition function T and reward function R are deterministic which means that source of TD error is because of misclassification. Therefore, in each step of RL, Δ_t is a measure of perceptual aliasing in a state s with respect to an action a. The function for detecting aliasing (*checkAliasing*) is shown in Table II.

TABLE II

ALGORITHM 2: ALIASING DETECTOR, $checkAliasing(s_t)$

1: for action $a \in A$ do
2: $mem(a)$= memorys items with action a under s_t
3: $\sigma^2(a) = calcVariance(mem(a))$
4: if $\sigma^2(a) > aliasingThreshold$
5: return *true*
6: end if
7: end for
8: return *false*

D. Tree Refinement

When an aliased class is detected, a spatial location must be selected to dissociate items under this aliased class. In order to find the best location an anticipatory mechanism is needed. When an image is classified in state s_t, codebooks in some spatial regions are saved for this node plus the Δ_t and the elicited action (*gatherMem*).

Assume that $M_k = \{m_{k,1}, m_{k,2}, \cdots, m_{k,q}\}$ is the set of q spatial locations for the k-th leaf of the tree. Codebooks at these locations are calculated for every image which ends to this node. Let Mem_k be the matrix of memory items in the k-th leaf:

$$Mem_k = [mem_{i,j}], i = 1 \cdots Mem_k, j = 1 \cdots q + 2 \tag{4}$$

Each item of this memory is represented as:

$$mem_{i,j} = [\tau_{i,1}, \tau_{i,2}, \cdots, \tau_{i,q}, a_t, \Delta_t] \tag{5}$$

where $\tau_{i,l}$ is the codebook of the l-th spatial position of i-th memory item. Note that when a new node is created in the tree, M_k is initialized in some way for this node. Two possible approaches are random selection of spatial locations or some positions relative to the end of saccade at that node (for example in some directions). Every method for saccade generation must satisfy this condition that FOAs along the path from each node to the root must not be the same. Positions are in the image coordinate frame. Whenever size of the memory under a leaf node exceeds a threshold (*memThreshold*) and it has aliasing, then tree is refined to remove aliasing. Tree refinement is then done by selecting the spatial location which mostly minimizes $\sigma(\Delta_t)$ of memory items according to 6.

$$\begin{aligned} [p^* a^*] &= argmin_{p,a} \left(\sigma^2(L) - \sum_{c=1}^{|T|} \frac{|L_{a,p,c}|}{|L_a|} \sigma^2(L_{a,p,c}) \right) \\ &= argmax_{p,a} \left(\sum_{c=1}^{|T|} \frac{|L_{a,p,c}|}{|L_a|} \sigma^2(L_{a,p,c}) \right) \end{aligned} \tag{6}$$

In the above formula, L is the set of Δ_ts of all memory items, L_a is the set of Δ_ts of items with action a. $L_{a,p,c}$ is the set of Δ_ts of items with action a, spatial location p and codebook c. $|U|$ and $\sigma^2(U)$ are the size and variance of a set U. p^* and a^* are the location and the action which reduces

TABLE III

ALGORITHM 3: TREE REFINEMENT, $modifyTree(tree, s_t)$

```
1: for action a ∈ A do
2:   mem(a)= memory items with action a under s_t
3:   for location m ∈ M do
4:     find p* and a* according to equation 6
5:   end for
6: end for
```

variance the most respectively. The winning spatial location p^* is saved for the node and is used for future tree traversals. Tree is expanded based on seen codebooks at location p^*. For the new created nodes, q potentioal spatial saccade locations are randomly generated. Tree modification is shown in the pseudocode of Table III.

III. EXPERIMENTAL RESULTS

In order to evaluate the performance of the Saccade-Tree algorithm, we apply it to two tasks: 1) navigation in a visual gridworld with obstacles and a goal state and 2) object recognition.

A. Visual Navigation

The aim in this task is to reach the goal state in the bottom of the grid marked with letter G (Fig. 1). The agent has a set of 4 physical actions, $A = \{$*move up, move right, move left, move down*$\}$ and has no access to its (x, y) position in the grid. Agent's only perception of the world is through an image of the object underneath its foot. Any movement taking the agent to an obstacle cell or outside the gird brings it -1 punishment. When it reaches the goal state, it is rewarded $+1$. Each cell of the grid is carpeted with a 128×128 image of the COIL100 object database. Saccade-Tree has managed to recognize all the objects in an action-based manner as well as a valid policy by creating 8 distinct perceptual classes for 11 objects. Interestingly some positions with the same best actions are classified under the same leaves.

Fig. 2 shows the results of action-based object recognition and navigation in a grid with obstacles in positions 6 and 11 and goal state in position 16. In such a gird only two best actions are used from the set of 4 physical actions, *move right* and *move down*. Four conditions are considered: 1) all objects for all positions are unique 2) only 3 positions with the same best actions are assigned the same objects 3) seven positions with the same best actions are assigned the same objects and 4) from 13 non-obstacle, non-goal positions, 7 with one action and remaining 6 with the other action as best are assigned the same objects.

Shown by this figure, as the number of distinct perceptions decrease, the number of states and hence the average tree depth also decrease. This is because there is no need to further refine the tree when there is no aliasing. Interestingly, when there are only 2 distinct objects in the grid (condition 4), the resultant tree has always 2 nodes with the average

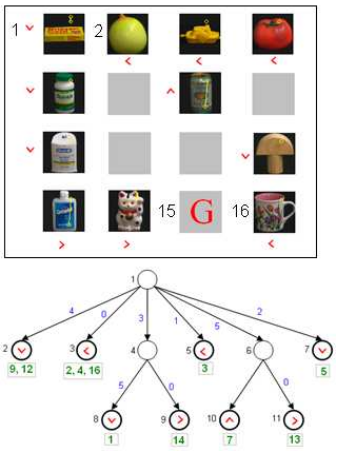

Fig. 1. Navigation in the visual gridworld. Top: visual gridworld with learned best actions. Bottom: Learned saccade tree. Numbers below leaves indicate the positions in the grid, flashes inside the circles show learned actions and blue numbers on the edges are the codebooks seen at the FOA. Eight spatial locations were generated in random i.e. $q = 8$ and $r = 10$.

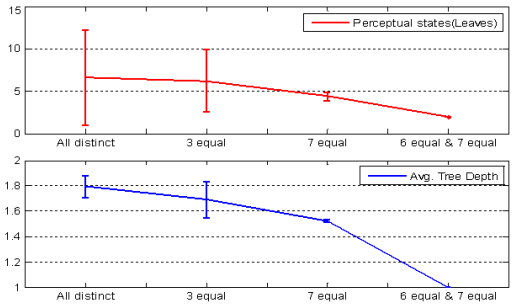

Fig. 2. Top: Number of perceptual states Bottom: Avergae tree depth for a 4×4 gridworld with obstacles at positions 6 and 11 and goal at 16.

tree depth of 1. The variances in diagrams are due to different initial conditions in each run like the spatial locations selected at the beginning. All cases resulted in optimal action selection policy.

Experimenting with another more complex 10×10 gridworld shown in Fig. 3, Saccade-Tree suceeded to derive the optimal policy after 9 phases. Number of generated visual calsses were 68 with the avergae tree depth of 3.6. The same parameters as in Fig. 1 were used with the saccade generation function f_2.

We also compared the traditional RL when agent has access to its (x, y) positions with saccade-tree algorithm. Fig. 4.*a* shows the instantanous and smoothed average reward in 30 epizodes of the RL algorithm. Agent was able to solve the task after these epizodes. Fig. 4.*b* shows the avergae reward of the agent in Tree-fixed phases of saccade tree. Red horizontal line is the avergae reward of the agent derived from Fig. 4.*a*. As it shows both methods converged to the same avergae reward but traditional RL has 16 states while Saccade-Tree generated 8 states.

B. Handling POMDP Cases

When same objects are assigned to the positions with different best actions, algorithm does not converge. This is

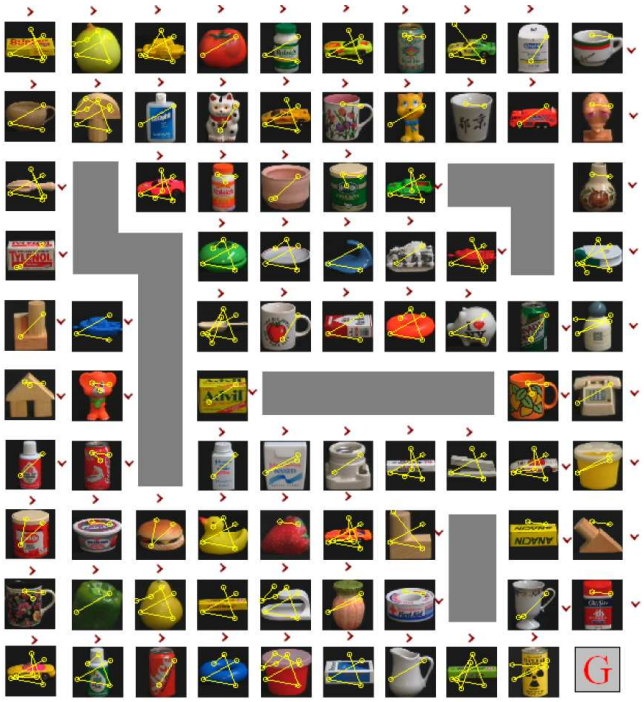

Fig. 3. 10×10 grid with *maxEpizodes* equal to 400.

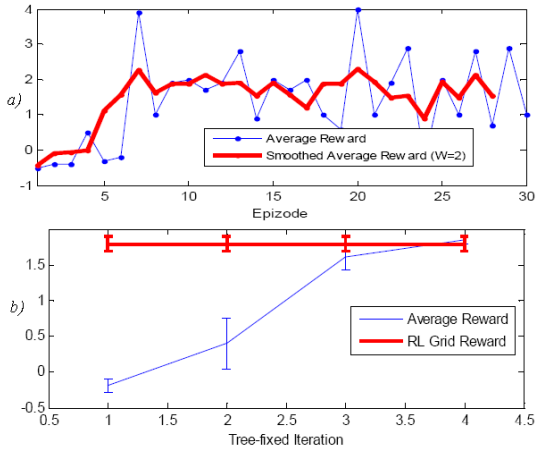

Fig. 4. Comparison with traditional reinforcement learning algorithm.

because there is no aliasing in perception however another kind of aliasing which emanates from not knowing the previous actions and observations exists yet. This has been predicted in original U-Tree algorithm but has not been addressed in the RLVC algorithm. It is important to note that this kind of aliasing frequently occurs in real-world situations. To remedy this, we consider a history for the agent with each item a pair of previous states and actions to a certain depth n. Let $e_{t,j}$ be the j-th pair of the history:

$$e_{t,j} = (a_{t-j}, s_{t-j}), j = 1 \cdots n, a \in \{A \cup Null\}, \\ s \in \{S \cup Null\} \quad (7)$$

Then new representation of a memory item becomes:

$$mem_{i,j} = [\tau_{i,1}, \tau_{i,2}, \cdots, \tau_{i,q}, e_{t,1}, e_{t,2}, \cdots, e_{t,n}, a_t, \Delta_t] \quad (8)$$

History is treated the same as the spatial locations. This way when splitting a node, the history items at that node also compete for reducing aliasing. Applying this new modification, now the Saccade-Tree algorithm is able to solve a grid with positions with different best actions having the same perceptions. In grid with obstacles at positions 6 and 11 and goal at 16, positions 2, 5, 12 and 15 were assigned the same object. As Fig. 5 shows, Saccade-Tree solved this problem by checking the previous action in the internal node marked with blue. Edges below this node check the previous action leaded to the current node.

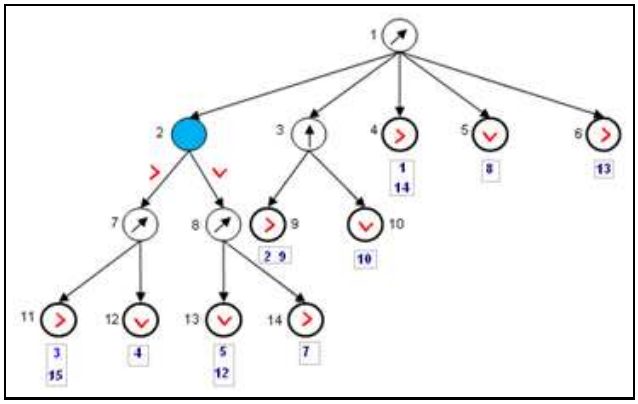

Fig. 5. Saccade tree for a grid with same object assigned to positions with different best actions (history depth of 1, previous state as Null) and spatial locations relative to end of saccade.

C. Object Recognition

Object recognition is a fundamental task of biological mechanisms. It allows an agent to abstract its knowledge and then be able to derive task independent representations. While above experiments showed that Saccade-Tree is capable of visual navigation and action-based object classification, this experiment investigates the capability of Saccade-Tree for classical object recognition. Basics of the algorithm are the same as before but with the distinction that here reward function is a $n \times n$ matrix for recognition of n objects. Diagonal elements of this matrix are $+1$ which shows correct classification of object and off-diagonal elements are -1 for punishing wrong classification. Class labels are assigned to the images instead of physical actions and there is no grid here. Saccade-Tree was able to classifiy 100 objects from the COIL100 database (one sample per image) using 8 random spatial locations and 5 codebooks with $r = 10$.

D. Perturbation Analysis

Learning happens all the time in biological creatures. It will be beneficial for a learning system to be able to adapt itself to dynamics of the environment and perceptual space. In this experiment we altered the image at position 15 in the grid used in Fig. 5 to a new unseen image after convergance and then learning was continued. Saccade-Tree expanded

node 9 in the gird and created 3 new states instead as shown by Fig. 6. Pruning removed node 4 which classified image at position 15 earlier. This result shows that Saccade-Tree is to some extent robust to distortions and perturbations in perceptual inputs.

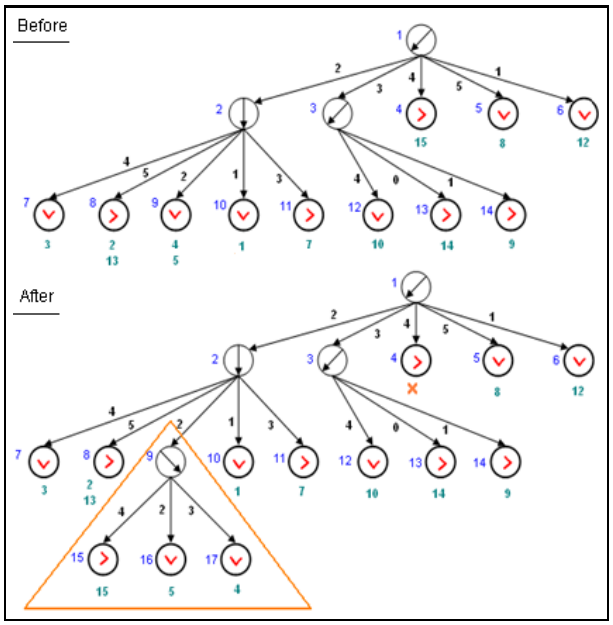

Fig. 6. Perturbation of a percept after saccade tree learning.

E. Invariance Analysis

Previous experiments showed that Saccade-Tree achieved 100% correct policy rate or object recognition rate when only a single image per class or position was used. However, this is not a right assumption for the perceptual space. A learning system while being able to discriminate among samples from different classes (specificity) has to be able to treat all the samples within a class equally (generalization). In this experiment we investigate generalization power of the Saccade-Tree algorithm.

A gridworld with obstacles at positions 6 and 11 and the goal at 16 is used. Each position of this grid is assigned a natural 640×480 scene ($r = 50pixels$). Images are taken from [17] . In each position, the agent captures a scene randomly among 5 possible scenes with major transformation (i.e it does not observe the same image per position but observes images from the same scene under major translation, scale and rotation). The goal here is to learn the correct policy but with minimum number of states or leaves. In all cases agent was able to solve the task. To measure the generalization power an index known as state reduction index is defined as in 9:

$$\begin{aligned} SRI &= \frac{1}{N}\sum_{n=1}^{N}\frac{\sigma^2(\overline{V(n)})}{mean(\overline{V(n)})} \\ \overline{V(n)} &= [v_1(n), v_2(n), \cdots, v_5(n)] \end{aligned} \quad (9)$$

In above formula, $\overline{V(n)}$ is the vector of leaf numbers and N is the number of objects. Numbering is done at the level order when tree is built. Zero value for SRI means that all samples of a scene are classified under the same leaf. The lower the SRI, the more generalization power is. Fig. 7.*a* compares the SRI, number of states and average tree depth (average of depths of all leaves) for two SIFT selection methods in Saccade-Tree (f_1 and f_2) and RLVC. As it can be seen highest magnitude SIFT is more invariant to image transformations than the nearest SIFT feature. RLVC has more generalization because it checks the existence of a SIFT feature anywhere in the image but with the disadvantage of more computation. Lower SRI means that lower number of states in average as Fig. 7.*a* shows. Average tree depth of three approaches is nearly the same.

Fig. 7.*b* shows average number of leaves, number of tree nodes and phases (turns) after convergence of the Saccade-Tree for recognition of 16 scenes (5 views each), where class labels are used instead of actions. Consistent with Fig. 7.*a*, it shows that largest magnitude SIFT leads to more generalization.

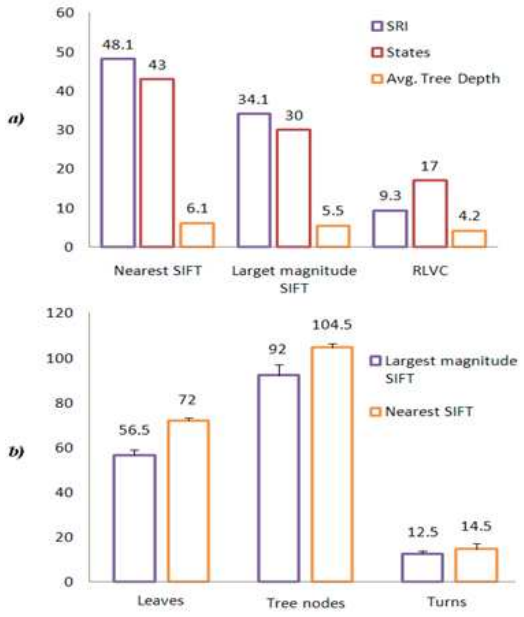

Fig. 7. Invariance analysis a) comparing Saccade-Tree and RLVC over a 4×4 visual navigation task with $r = 50$ and 5 codebooks, b) recognition of 16 objects with random spatial locations with 5 views per each position.

We also applied the algorithm to visual servoing in real-world scenarios. For this purpose, we created the environment shown in Fig. 8.*b* in *Webot* environmrnt [18]. There is a goal box and an obstacle in the middle of the environemt. The aim is to reach the goal whereever the simulated *e-puck* (Fig. 8.*c*) starts its move. The robot captures 320×240 images. We limited it to *turn left*, *turn right* and m*ove forward*. The robot uses its infrared sensors to understand whether it hits the obstacle or the goal at the corner. The Obstacle and the goal have different heights and then infrared sensors return different values. Fig. 8.*a* shows the environ-

ment carpeted with natural scenes. The robot was supposed to map each image it takes through its camera to its actions. The way is by saccading through the image and following the Saccade-Tree algorithm. Examples of some captured images are shown in Fig. 8.*d*. Refering to the generalization issue, while Saccade-Tree converges it generates many states. One solution to remedy the generalization problem of the Saccade-Tree for this task is to use other information like movements of the robot. For example, when robot turns right and goes forward and then turns right again, it is likely that it observess nealry the same scene as it saw first and these two images belong to the same class.

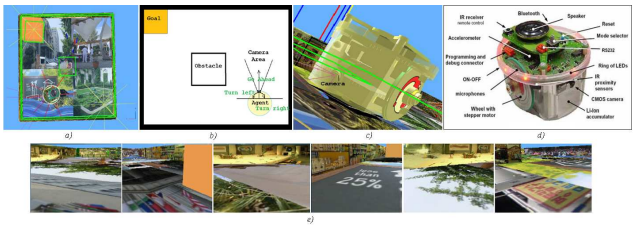

Fig. 8. Simulation experiment a)carpeted navigation environment, b)Goal and obstacle and e-puck actions, c)an image of simulated e-puck in the environment, d)e-puck robot and its sensors and e)sample images robots takes from the environment

IV. Discussions

Results show that Saccade-Tree is able to solve the gridworld and object recognition tasks very fast by only extracting SIFT features at small number of image regions. Compared with RLVC which has $O(kn^2)$ computational complexity where k is the average tree depth and n is the image size, complexity of Saccade-Tree is $O(k^{'}\pi r^2)$ where the average tree depth or saccade length is $k^{'}$ and r is the radius of FOA. Ignoring the constants, Saccade-Tree is $\frac{n}{r}$ times more faster than RLVC. Since, this is quite appealing, a shortcoming arises from this local processing and it is lack of generalization. Since Saccade-Tree algorithm works in spatial domain instead of feature space, it makes the method sensitive to large spatial transformations like translation, rotation and scale.

V. Conclusions and Future Works

An interesting observation is that representations are learned interactively and are expanded adaptively based on the agents needs. They are also as compact as possible and encode the information at the necessary level without unnecessary details. For example, for recognition of a scene, it would be very efficient and conclusive to attend to important spatial locations. Therefore, global image representation approaches although might propose more accurate solutions in some cases seems not to be the best solutions where information bottlenecks exsit. In accordance with these views, our method discretizes the visual world when it is needed and when it helps the agent to perform better by removing perceptual aliasing. The only predefined knowledge supplied to the agent was the clusters of visual features or codebooks. This does not put a big constraint on the method because these clusters could also be learned.

The weakness of Saccade-Tree in generalization is because saccadic movements are initiated in a coordinate frame locked to the image. This causes relocation of visual contents at FOA when an image is transformed. While we tried to remedy this by considering the codebook of the highest magnitude SIFT to some extent, problem still remains to be investigated in future researches. A possible solution is by introducing a coordinate frame which is relative to a stable property for example salient points introduced by the bottom-up saliency based model of visual attention [16] could be promising candidates.

References

[1] A. L. Yarbus, *Eye movements during perception of complex objects*, in Eye Movements and Vision, ed. L. A. Riggs, Plenum Press, New York, ch. 7, pp. 171196, 1967.

[2] V. Maljkovic and K. Nakayama, Priming of pop-out: I. Role of features, *Memory and Cognition*, vol. 22, pp. 657-672, 1994.

[3] E. Gibson and E. Spelke, *The development of perception*, In Flavell, J. H., and Markman, E. M. (Eds.), Handbook of Child Psychology Vol. III: Cognitive Development (4th edition)., chap. 1, pp. 276. Wiley, 1983.

[4] M. Tarr and Y. Cheng, Learning to see faces and objects, *Trends in Cognitive Sciences*, vol. 7, no. 1, pp. 2330, 2003.

[5] J. Triesch, D. H. Ballard, M. M. Hayhoe and B. T. Sullivan, What you see is what you need, *Journal of Vision*, vol. 3, pp. 8694, 2003.

[6] R. Munos and A. Moore, Variable resolution discretization in optimal control, *Machine Learning*, vol. 49, pp. 291323, 2002.

[7] A. Moore and C. Atkeson, The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces, *Machine Learning*, 21, 1995.

[8] D. Chapman and L. Kaelbling, "Input generalization in delayed reinforcement learning: An algorithm and performance comparisons", *In Proc. of the 12th International Joint Conference on Artificial Intelligence (IJCAI)*, Sydney, 1991, pp. 726731.

[9] R. McCallum, *Reinforcement learning with selective perception and hidden state*, Ph.D. thesis, University of Rochester, New York, 1996.

[10] W. Uther and M. Veloso, Tree based discretization for continuous state space reinforcement learning, *In Proc. of the 15th National Conference on Artificial Intelligence (AAAI)*, pp. 769774, Madison (WI, USA), 1998.

[11] S. Jodogne and J.H. Piater, Closed-Loop learning of visual control policies, *Journal of Artificial Intelligence Research*, vol. 28, no. 349-391, 2007.

[12] A. Borji, M.N. Ahmadabadi and B.N. Araabi, "Learning sequential visual attention control through dynamic state space discretization", *in IEEE International Conference on Robotics and Automation*, Kobe, Japan, 2009, pp. 2258-2263.

[13] R. Sutton and A. Barto, *Reinforcement Learning: an Introduction*, MIT Press. 1998.

[14] C. Watkins and P. Dayan, Q-learning, *Machine Learning*, vol. 8, no. 3,4, pp. 279292, 1992.

[15] D. Lowe, Distinctive image features from scale-invariant keypoints, *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91-110, 2004.

[16] L. Itti, C. Koch and E. Niebur, A model of saliency-based visual attention for rapid scene analysis, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 20, no. 11, pp. 12541259, 1998.

[17] http://www.montefiore.ulg.ac.be/ jodogne/phd-database/

[18] http://www.cyberbotics.com/