RL-IAC: An Exploration Policy for Online Saliency Learning on an Autonomous Mobile Robot

Céline Craye, David Filliat, Jean-François Goudou

RL-IAC: An Exploration Policy for Online Saliency Learning on an Autonomous Mobile Robot

Céline Craye[1,2], David Filliat[1] and Jean-François Goudou[2]

***Abstract*— In the context of visual object search and localization, saliency maps provide an efficient way to find object candidates in images. Unlike most approaches, we propose a way to learn saliency maps directly on a robot, by exploring the environment, discovering salient objects using geometric cues, and learning their visual aspects. More importantly, we provide an autonomous exploration strategy able to drive the robot for the task of learning saliency. For that, we describe the *Reinforcement Learning-Intelligent Adaptive Curiosity* algorithm (RL-IAC), a mechanism based on IAC (*Intelligent Adaptive Curiosity*) able to guide the robot through areas of the space where learning progress is high, while minimizing the time spent to move in its environment without learning. We demonstrate first that our saliency approach is an efficient tool to generate relevant object boxes proposal in the input image and significantly outperforms state-of-the-art algorithms. Second, we show that RL-IAC can drastically decrease the required time for learning saliency compared to random exploration.**

I. INTRODUCTION

Object localization has received a lot of attention in the recent years. Today, deep learning-based methods [19] provide efficient ways to localize and identify a large set of objects in a wide variety of complex environments, but, they generaly require hours or days of offline training, high GPU resources, thousand to millions of training images, and are not really flexible to novelty. On the other hand, domestic mobile robots are meant to evolve essentially in indoor environments, interacting with a limited amount of objects, for specific tasks and thus do not require such wide scope capacity. Moreover, they should be able to adapt to novelty by quickly updating the representation of their environment, while dealing with limited computational resources. Lastly, the displacement of the robot makes it possible to move to favorable observation conditions in order to improve recognition performances. In that regard, providing the robot a way to explore, actively learn and update online its representation and knowledge is a very desirable property.

Visual exploration of the environment by mobile robots is generally associated with a way to localize areas of interest on which the robot should focus on. This localization mechanism is typically driven by visual saliency maps [10], [16] or, if depth information is available, geometrical segmentation [2], [6]. In the first case, bottom-up saliency maps [11], [21] highlight in the input image stimuli that are intrinsically salient in their context. As interesting elements are not always intrinsically salient, some approach suggest to add top-down modulation of the saliency map in order to further enhance elements related to a given task [10]. In the second case, indoor object segmentation based on depth information usually rely on finding planar surfaces and objects lying on it. Those methods can accurately detect objects on tables or floor, but are limited by the sensor quality, geometrical constraints (size or distance to the objects) and require more computation time than bottom-up saliency maps. In recent years, other approaches have proposed to directly generate bounding boxes around potential objects of interest [1], [22], thus avoiding the traditional sliding window approach for object recognition. Nevertheless, the measure used to generate boxes (such as *objectness*) is often tightly bounded with the concept of saliency. So far, saliency maps are mostly used as a black box and are not learned (although sometimes refined) directly during the exploration of a particular environment. Our first contribution is to provide a method that incrementally learns saliency as the robot observes the environment. The produced saliency maps are therefore dedicated to the environment that was explored, but remain flexible to novelty. We further show that these saliency map can be used to refine object boxes proposals.

Active exploration by robots can be done in many different ways depending on the task and available hardware. On mobile robots aiming at making a semantic cartography of the environment, predefined path plans [16], navigation graphs [13], or frontier-based explorations [12] can drive the robot's displacement. In the context of learning an optimal action policy given visual inputs (typically learning eye saccades to identify objects), reinforcement learning approaches have been proposed [4], [18]. The use of intrinsic motivation (*i.e.* using a reward system that is not related to an external goal, but to the acquisition of competences or knowledge) has also been largely investigated. Oudeyer *et al.* proposed to drive exploration using learning or competence progress [3], [17], while [7] have used the error of prediction of *salient event* to speed up a classical reinforcement learning approach, and [15] have integrated artificial curiosity to deal with high dimensional visual observation in the context of reinforcement learning. Our second contribution is to propose a method, called RL-IAC (for *Reinforcement Learning Intelligent Adaptive Curiosity*, that provides an exploration strategy based on the ideas of the IAC algorithm [17]. One of the challenges for applying such a method to the task of saliency learning is that the cost for actions (displacement of the robot in our case) is not considered in IAC, while it is critical in our framework. We therefore add a reinforcement-

[1]U2IS, ENSTA ParisTech, Inria FLOWERS team, Université Paris-Saclay, 828 bd des Maréchaux, 91762 Palaiseau cedex France celine.craye@ensta-paritech.fr

[2]Thales - SIX - Theresis - Vision & Sensing 1, avenue Augustin Fresnel, 91767 Palaiseau, France celine.craye@thalesgroup.com

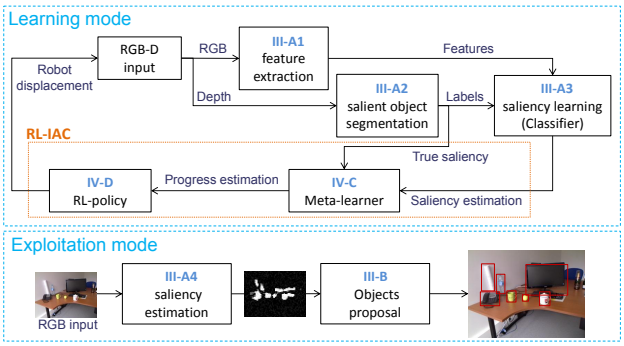


Fig. 1. General architecture of our system

learning module that is retrained after each displacement to determine the action that is the best compromise between learning and displacement. Note that RL here is not used as a final goal (we intend to learn saliency, not displacement), but as a mean to decide at each step the best actions to follow in order to learn saliency.

In a previous work [9], we described the core mechanism of the online learning of saliency based on depth segmentation and demonstrated its efficiency compared to state-of-the-art techniques. We here propose a way to use the generated saliency maps to produce boxes proposals for objects in the environment. Second, we presented in [9] some preliminary results of our exploration mechanism based on the *Intelligent Adaptive Curiosity* (IAC) [17], adapted to the problem of saliency learning. A that time, we succesfully applied IAC in a semi-simulated setup so that an accurate saliency model could be learned with a limited number of relevant visual samples. However, the time spent by the robot in displacements from a position to another one was not taken into acount, so that the time really spent during exploration was not considered. We here propose RL-IAC (for *Reinforcement Learning-IAC*) that provides a way to get a trade-off between the time lost in displacements across the environment, and the acquisition of relevant visual samples allowing a faster and better learning.

II. SALIENCY LEARNING AND OBJECT PROPOSAL

Figure 1 presents the general architecture of our system along with the corresponding section for each block. In a learning stage, the system extracts RGB features (see Section II-A.1) and learns the visual (RGB) aspect of salient elements within their context using a depth-based object segmentation as a supervision signal (see Section II-A.2). Based on the supervision signal and the estimated saliency, we use a meta-learner to obtain an estimation of the local learning progress (see Section III-C). Then, based on the progress estimation in different portions of the environment we evaluate with an RL approach the best action to take between moving to a regions that has a higher learning progress, and keep learning around the current position (Section III-D). Once exploration and learning are finished, we exploit the model to generate environment specific saliency maps using the RGB image (Section II-A.4), and use these saliency maps to generate boxes that isolate objects of interest (Section II-B).

A. Saliency learning

We here recall the key elements of our saliency learning approach. Please refer to [8], [9] for further explanations.

1) Feature extraction : A feature extractor is applied to the RGB image in order to encode the color of each pixel and its neighborhood at different scales, averaged using superpixels. The feature extractor is applied on the whole input frame, and returns a 39 dimensions feature vector for each pixel. Unlike depth segmentation, features are available everywhere in the image. Refer to [8] for more details.

2) Depth-based object segmentation : The object segmenter is based on the depth map and detects objects lying on planar surfaces (typically tables or floor), with a size between 10 and 150 centimeters. We use an adapted version of the method proposed by [6]. The segmentation is based on detecting the floor plane in the depth map, removing walls, and creating clusters with points that neither belong to the floor nor to the walls to create object candidates. Those clusters of points are also used in Section II-B to generate SegBoxes. Then, the segmentation mask is constructed as follows (See Fig. 4, last row for illustrations): Grey areas of the segmentation mask are pixels where depth value is unavailable (dark gray), pixels that either belong to the floor or walls are labeled *not salient* (black). To avoid false positives as much as possible, we categorize as *undetermined* (light gray) the clusters that are either too small, too large, or having a contact with the border of the frame. Other clusters are considered as salient object, labeled *salient* (white).

3) Online learning: The classifier is continuously updated based on the saliency labels provided by the segmentation mask and the corresponding RGB features. Each pixel of the input image is associated with a feature vector from the feature extractor and a label from the segmentation. We train our classifier with the feature-label samples (we only keep those that were labeled *salient* or *not salient*) in order to predict the saliency of a given pixel. The classifier used in our implementation is a modified random forest, designed to re-train a model on-the-fly as new samples arrive (See more details in [9]). After each update, the classifier is able to estimate the saliency of an input based on the model trained with the previous observations, and the RGB image only.

4) Saliency estimation: The saliency maps are constructed by applying the classifier to RGB images. For each pixel, the classifier outputs a score between 0 and 1 that is used as its associated saliency (see Fig. 4, fourth row for examples). Although less accurate than depth segmentation, saliency maps provide a more complete estimation of the saliency, for each pixel of the image.

B. Object bounding boxes proposal

This section presents a new contribution that is a practical application to the saliency learning. The saliency map provides an indication of the interestingness of a given pixel. In order to localize objects in an image, an additional step is

necessary to group salient pixels into object candidates. To this end, we use two types of bounding boxes proposal, and we select or reject each of them based on a score related to saliency. The first bounding boxes are obtained by the EdgeBoxes [22] algorithm: we compute for a given RGB input the 100 most likely EdgeBoxes, and their associated edge score (h_b^{in} in [22]). The second type of bounding boxes are obtained with the segmentation result (called SegBoxes here for simplicity): during segmentation, pixels of the depth map are clusterized in order to create object candidates (See Section II-A.2). Among object candidates, some are labeled *salient*, some should be salient objects but are labeled *undetermined* because they touch a border or are poorly segmented, and some are just artifacts. We define Segboxes as the bounding boxes around all objects candidates (labeled *salient* or *undetermined*) of the segmentation mask.

For both EdgeBoxes and SegBoxes, we associate each box B with a score related to saliency (called here the *saliency consistency score*, or SCscore), representing the ratio of salient pixels in the box:

$$SCscore(B) = \frac{1}{w_B \times h_B} \sum_{i,j \in B} S(i,j) \qquad (1)$$

where $S(i,j)$ is the saliency of the pixel at (i,j) , obtained from the saliency map and w_B and h_B are the width and height of B. The highest the score is for a given box, the most likely it is to contain a salient object. For the EdgeBoxes, the SCscore is multiplied by h_b^{in} . This way, small boxes found within a salient object might be rejected if the h_b^{in} score is low enough. Last we filter out Segboxes and Edgeboxes with a final score below a certain threshold and keep the remaining ones. We found $SCscore = 0.2$ for SegBoxes and $SCscore \times h_b^{in} = 0.01$ to be good thresholds.

III. RL-IAC

In the scope of an autonomous, life-long exploration and learning, the robot must be equipped with an exploration strategy able to drive the robot's actions. This one has a critical impact on the learning quality and efficiency, and should allow the robot to collect interesting examples, while avoiding problems such as *catastrophic forgetting*. Ideally, exploration should be such that the robot focuses on areas where learning is neither trivial nor impossible. This avoids decreasing in the learning quality because of irrelevant samples, and speeds up learning by focusing on appropriate areas first.

Our exploration strategy, that we call *Reinforcement Learning-IAC* (or RL-IAC), uses learning progress as an intrinsic reward as suggested by the IAC (*Intelligent Adaptive Curiosity*) algorithm [17]. In this section, we first recall the main components of IAC. We then explain the major challenges of applying such an algorithm to our problem, and describe the proposed solution to this end.

A. IAC

IAC is originaly designed to learn a sensorimotor mapping. The robot takes actions and learns to predict the consequence of these actions on the environment through sensor feedbacks. In IAC, the robot is learning this mapping by taking actions that maximize the learning progress. Therefore, the algorithm makes the robot focus on cases that are neither too easy nor too hard, so that progresses are constantly made and no time is wasted in unlearnable situations. The key components of the algorithm are:

- a learner that learns a mapping $X \rightarrow Y$ between motor commands X and sensor inputs Y.
- a separation of the motor space into regions. This separation is essential as it allows a local estimation of the learning evolution.
- a meta-learner that monitors the learning evolution in each region and estimates progresses. Progresses are estimated based on the evolution of the error between the learner estimate and the actual sensor input.

To apply the algorithm, the following procedure is used: The robot takes an action (for example, moving its arm with the input motor command X) in a given region R_i. It receives a sensor feedback about the consequence of the action (for example, the 2D position Y of the hand of the robot in the frame of a camera), while the learner tries to predict the sensor feedback $\widetilde{Y}$ based on the X input and previously seen samples. The learner is then updated based on the (X,Y) sample, whereas the prediction error $\|Y - \widetilde{Y}\|$ is added to the error history of the meta-learner region R_i. Last, the progress in region R_i is re-evaluated based on the error history over the last few samples. The next action to be taken from the robot is then randomly chosen in the region that has the highest learning progress.

B. Application to saliency learning on a mobile robot

IAC needs a few adjustments to be applied to our problem. This section lists the main aspects to be considered.

1) Role of the motor commands: The mapping learned in IAC is between motor command and sensor input. Our saliency algorithm learns a mapping between sensor inputs X (the RGB-features) and other sensor inputs Y (the depth-based segmentation). Motor commands in our case are used to move the robot across the environment to a new position and orientation. As a result, we do not use motor command directly as new samples for our learner, but as a way to receive new incoming samples (from the RGB-D input), by showing a different point of view of the environment. Once learning is finished, the model is then completely independent from any motor command.

2) Regions definition: In IAC, the regions are obtained by incrementally splitting the input (X) space of the mapping which is also the motor space. In our case, the X space of the mapping is the 39 dimensions space of the RGB-features which is not well-suited to define regions. The first reason is the dimensionality, much higher than spaces where IAC has been used. The second is the accessibility of examples. In IAC, examples are accessible by sending a motor command X and receiving the sensor feedback Y. In our case, to get a specific X and associated label Y, the robot has to move to a point of view that contains X, which is not feasible in

practice as we do not know the occurences of X before we have seen them. In our implementation, we therefore choose to define regions in the space of positions and orientations accessible by the robot. This space only has 3 dimensions (2 dimensions for the location, 1 for the orientation) and is strongly related with motor commands, that modify the state of our robot in this space.

3) Cost for actions: In IAC, the cost for taking actions is not considered. In that case, a greedy policy that choses the regions with highest learning progress is enough. For a mobile robot moving in a large environment (e.g. a building), the displacements between two regions can be extremely time consuming, making the greedy policy inefficient. We therefore propose to extend the IAC policy with a RL module that estimates the best trade-off policy between progresses and displacement. At each step, given the RL policy, the robot can choose between moving to a region with higher progress, or stay nearby and keep learning.

C. Progress estimation

In our approach, the learner is the classifier that learns saliency from RGB features and depth-based segmentation (see Section II-A.3). The space that is separated into region is the space of position/orientations accessible by the robot. A regions is a subset of positions (x, y, θ) that the robot can reach. Thus, the robot is in region R_i at time t if its current position $(x(t), y(t), \theta(t)) \in R_i$. An action in this space is a displacement of the robot to a given position (x', y', θ'). In our current implementation, the geometry of the robot's environment is supposed to be known, and the regions are defined arbitrarily prior to the experiment.

As in IAC, the meta-learner stores for each region a history of the learning error based on the differences between estimated saliency (from the learner estimation) and observed saliency (from the segmentation algorithm). Each time a new frame is acquired in region R_i, saliency map is estimated and segmentation mask is computed. By keeping only *salient* and *not salient* pixels from the segmentation, we generate the observation set O, and by taking the associated saliency response for each of these pixels, we obtain the estimation set E. Thus, error evaluation $Err_i(t_i)$ is obtained after a new frame is acquired in region R_i after t_i observations in the region by the following formula:

$$Err_i(t_i) = 1 - F_1(O_{t_i}, E_{t_i}) \tag{2}$$

where $F_1(.,.)$ is the F_1 score [1].

An estimation of the learning progress in region i, is obtained by a linear regression of the error rate history Err_i over the last τ samples ($\tau = 10$ in our case):

$$\begin{pmatrix} Err_i(t-\tau) \\ \vdots \\ Err_i(t) \end{pmatrix} = \beta_i(t) \times \begin{pmatrix} t-\tau \\ \vdots \\ t \end{pmatrix} + \begin{pmatrix} \epsilon(t-\tau) \\ \vdots \\ \epsilon(t) \end{pmatrix} \tag{3}$$

[1] $F_1 = \frac{2tp}{2tp+fp+fn}$, where tp, fp and fn are the true positives, false positives and false negatives. We use the F_1 score as our error metrics for Err_i, because *not salient* pixels are representing more than 80% of the samples, making accuracy inappropriate for error estimation.

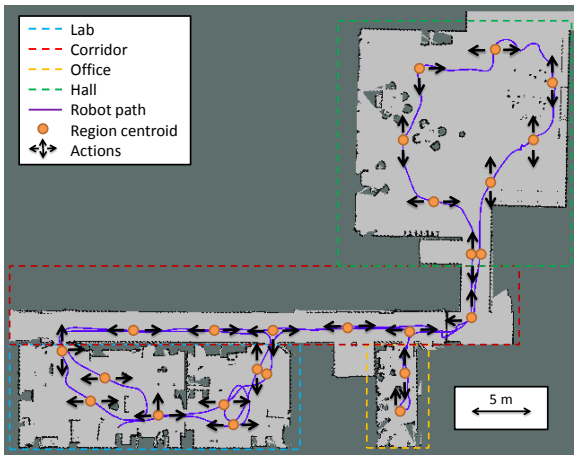


Fig. 2. Navigation graph used for displacing the robot. The path followed by the robot to record the sequence is represented by a purple line. Nodes of the navigation graph are represented by the region centroids, and available actions to move to adjacent regions are represented by arrows.

with $\epsilon(t)$ the residual error. The learning progress LP_i in region i is defined as the derivative of the learning curve (or the opposite of the error rate) in that region:

$$LP_i(t) = -\beta_i(t) \tag{4}$$

D. Exploration policy

Because of the displacement time, the exploration strategy that only follows the highest learning progress is not well-suited as is. Therefore, we propose to use Q-learning [20] to constantly re-estimate the best policy to find a good tradeoff between moving and learning, given the progress in each region. The idea is to simulate future displacements of the robot, and to determine the policy that optimizes progress (or reward). The next displacement is taken by following this policy, the progress in regions is updated, and a new policy is re-estimated.

The problem is modeled as a navigation graph where states are the regions in which progress is evaluated. Adjacent regions are connected by edges in the graph, and the cost for moving to an adjacent region is defined by the time a robot would take to move between the centroids of the two regions. See Figure 2 for an example of navigation graph. The robot can move to adjacent regions or stay within the current region by taking a displacement action in $M = \{up, down, left, right, stay\}$, or $learn$ in the current region. If the action $learn$ is selected, the robot grabs an RGB-D input, process it and updates the saliency learner as well as the meta-learners. If the action is in M, the robot moves to the corresponding adjacent region, randomly selects a new position (x', y', θ') in it and stops at this position.

Suppose that at time t the robot is in region R_i. Each region R_j of the environment is then associated with an estimated learning progress $LP_j(t)$. To decide the next action to take, we simulate 1000 episodes for an horizon of time $N = 3600s$ (*i.e.* from t to $t + N$) in which the robot takes action in the environment and collects reward based on learning progress. The time spent for displacement depends

on the distance between centroids of two regions, whereas the time spent for learning is set as 1s (See Section IV-B for more details about those values). For each episode, the initial state is always R_i. The learning progress is considered as constant during the whole episode, and the reward received when taking an action a in region R_j is as follows:

$$r(R_j, a) = \begin{cases} LP_j(t) \text{ if } a = learn \\ 0 \text{ otherwise} \end{cases} \quad (5)$$

Based on the simulated episodes, we determine an optimal policy by updating a Q-matrix according to the following rule

$$Q(s_k, a_k) = r(s_k, a_k) + \gamma \max_{a' \in M \cup \{learn\}} Q(s_{k+1}, a') \quad (6)$$

with γ the discount factor (0.1 in our implementation), s_k the region where the robot is after k actions, a_k the action to take next, and s_{k+1} the region after taking action a_k. Once all the episodes have been simulated, we select the next action a_t taken by the robot such that

$$a_t = \underset{a' \in M \cup \{learn\}}{\text{Argmax}} (Q(R_i, a')) \quad (7)$$

After this action is taken, learning progress is re-estimated and a new batch of episodes is simulated to train a new Q matrix and decide the next action to take.

Note that each Q-learning policy is obtained by considering the reward as constant in time. This assumption is wrong in practice, as each new $learn$ action influences the learning progress (and therefore, the reward) that would eventually decrease to 0 when the learner cannot be any better. However, the assumption is accurate enough to estimate the next action to take. As the Q matrix is re-estimated before each new action, this approximation does not introduce a significant bias. Moreover, to force the robot to quickly get a first estimation of the progress in each region, we forced the progress in a given region to be very high as long as less than three samples were collected in that region. This additional constraint has the same effect as the R-MAX [5] exploration policy. Last, we used an epsilon-greedy strategy to move to a random region 10% of the time.

IV. EXPERIMENTAL RESULTS

For experimental results, we used two different Kinect datasets composed of RGB-D data with associated camera positions (x, y, θ) obtained by a SLAM algorithm. The saliency and associated object proposal were evaluated on the images of the dataset without considering the position of the camera (see Section IV-A). The exploration strategy was evaluated by simulating the displacement of the robot to one of the position/orientation available in the dataset and getting the associated RGB-D data(see Section IV-B).

The first dataset was collected from a pioneer 3DX robot, with a Kinect RGB-D camera mounted at 1 meter from the ground and tilted slightly downward. We manually controlled the robot in an office building in order to visit coridors, laboratory, hall and offices. We recorded a 15 minutes length video sequence at 20Hz with the robot moving at a 0.3m/s average speed, in which a large variety of views were captured (See Figure 2). The second one is a publicly available dataset called *RGB-D scenes dataset* [14]. The dataset consists of 8 video sequences of indoor scenes of everyday-life objects lying on tables. For both datasets, we manually labeled 100 frames to obtain a ground truth bounding boxes and masks to create an evaluation set. Those frames were removed from the dataset and used for evaluation only.

A. Object proposal

Before presenting the saliency learning progression, we analyze the performance reached when enough samples are used to train the classifier. The evaluation in this section is done with a saliency model that was learned from the entire training sequences and evaluated on the testing datasets. In [9], we demonstrated that our saliency model outperformed several state-of-the-art saliency techniques when trained and evaluated on a similar environment. We now demonstrate that these saliency maps can be used to improve the relevance of the EdgeBoxes [22] and SegBoxes (See Section II-B). To this end, we ran the EdgeBoxes algorithm for each frame of the evaluation set, we collected the 100 best ranked bounding boxes along with their h_b^{in} score. We calculated the SCscore for each box based on Eq. 1 and used it to produce a new ranking for the bounding boxes. We also generated the SegBoxes and filtered out the one with an SCscore above 0.2. When combined with the EdgeBoxes proposal, the SegBoxes bounding boxes were ranked before the EdgeBoxes, as they are much more likely to actually contain objects.

The evaluation metrics is the detection rate versus the number of proposal, based on the *intersection over union* measure (IoU=0.7 in our case) to count the number of detections. This measure is used by [22] to evaluate their performance over state of the art approaches. Results are displayed in Figure 3 for the two datasets. The curves represent the performance for :

- EdgeBoxes alone (*EdgeBoxes*),
- EdgeBoxes re-ranked by the SCscore (*EdgeBoxes+SCscore*),
- EdgeBoxes re-ranked by the SCscore produced by a bottom-up saliency map method called BMS [21](*EB+BMS SCscore*),
- SegBoxes filtered by the SCscore (*SegBoxes+SCscore*)
- the combination of re-ranked EdgeBoxes and SegBoxes (*EB+SegBoxes+SCscore*).

As expected, the use of the SCscore on EdgeBoxes allows a much better detection rate on both datasets. Moreover, using a bottom-up saliency map such as BMS instead of our saliency slightly improves the performance on the *RGB-D scenes dataset*, but performs worse on the other one. The SegBoxes are often very relevant, but the number of proposal is low (less that 8 in any cases), and they do not cover the entire image (reflective objects are poorly detected, objects more than 4 meters away from the Kinect are not processed). For that reason, the SegBoxes alone have a limited detection rate, but they provide a set of boxes that is complementary to the EdgeBoxes. When SegBoxes proposal are combined with EdgeBoxes, the detection is significantly improved. Figure

4 shows sample results to compare the ground truth (first row), the top 5 EdgeBoxes proposals (second rows), the top 5 EdgeBoxes+SCscore proposals (third row, blue) as well as the SegBoxes proposals (third row, yellow). The SegBoxes almost always provides relevant boxes, but it also misses a lot of objects, either because they are too far to be segmented (sample 5, 7), or because segmentation failed (sample 4). In this case, the remaining objects location are recovered by the EdgeBoxes. Moreover, the use of the SCscore to modify the ranking of the EdgeBoxes allows to favor boxes that surround salient elements while removing distractors such as windows or cables (sample 2, 7, 8). Last, it is possible to cope with frames that does not contain any salient object (sample 6) by filtering boxes with an SCscore above a certain threshold (0.01 in our case).

B. Exploration strategy

We now look at the evolution of the saliency quality during incremental learning. We use again the two datasets, and associate a navigation graph to both of them. The navigation graph of the sequence recorded in our building (displayed on Figure 2) has been manually designed based on the robot path during the sequence. The centroids were chosen to be roughly equally spaced. For the RGB-D dataset, the navigation graph represents a building in which each room contains a sequence (8 in total) of the dataset. Each room is divided into 6 regions, and some regions are connected to other rooms (as if there was doors between rooms). To simulate the displacement of the robot and the acquisition of new frames, we associate a set of frames to each region of the navigation graph, based on their recorded (x, y, θ) positions. When the exploration policy sends a command to move the robot to a region R_i, we randomly select a frame that belongs to R_i, and calculate the time the robot would take to move to this position based on the euclidean distance at a 0.3m/speed. The action *learn* consists in grabbing an RGB-D input and launching the saliency learning procedure that contains the following steps and execution time [2]:

- feature extraction: 60ms
- depth segmentation: 20 to 1000ms based on the geometrical complexity of the input
- saliency estimation: 30ms
- meta-learner update: 10ms
- learner update: 10 to 5000s based on the number of samples from the begining of the experiments. The learner is trained in a separate process that keeps running while the robot is moving.
- calculate a new RL policy: 100ms

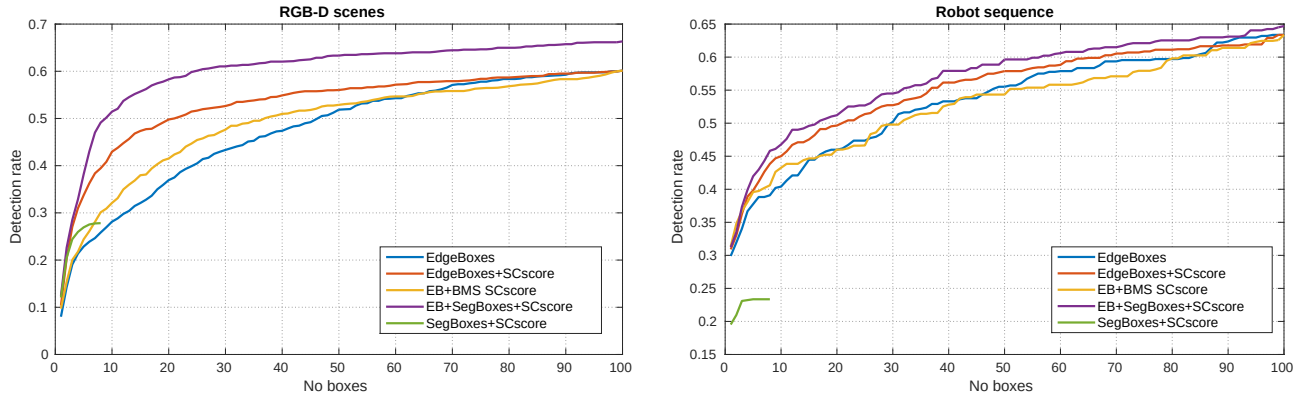
To demonstrate the benefits of exploring the environment using RL-IAC, we compare the evolution of the saliency when exploring the environment with different strategies. Each exploration strategy was tested 10 times on each dataset and considers the average and variance over those experiments. The performance of the system was evaluated using the evolution of the *overall error rate* of the system: based on the reference frames on which a ground truth is available, we compare the estimated saliency map for all of these frames with the available ground truth. We then use the formula provided by equation 2 on each frame and take the average error. Note that the *overall error rate* is an extrinsic metrics used to evaluate the performance of the system. It then differs from the *region error rate*, the intrinsic metrics (based on segmentation rather than ground truth) used to get an estimate of the error in each region in Section III-C. Figure 5 shows the evolution of the *overall error rate* in time on both environments, for 4 exploration strategies:

- RND: Random selection of a region and a position in it. Find the fastest path in the navigation graph to reach this position. Move to that position. Once arrived, take action *learn* that updates the saliency learner and meta-learner. After learning, select a new position to reach.
- IAC: Same as RND, except that the region selected is the one with highest learning progress.
- RND+learn: Same as RND, except that during displacement, the action *learn* is taken once in each visited region, rather than just in the region of destination.
- RL-IAC, as described in Section III.

On both datasets, RL-IAC is the one with the fastest decreasing error. IAC is, with RND, the slowest approach. Note that RND and IAC were both evaluated in [9], but the evaluation was done by drawing the error rate vs the number of observations. Here, we simulate the time the robot would actually take to move in the environment and learn, thus making the experiment more realistic. The *RND+learn* experiment provides a more realistic exploration scenario where the robot keeps learning while moving, the error rate then decreases much faster, but still slower than RL-IAC.

To get a better insight of the way exploration is done by the robot, we divide the building of our dataset in 4 main areas, namely *lab*, *office*, *corridor* and *hall* (See Figure 2). In these area the difficulty to learn saliency is not the same. For example, the corridor does not contain any salient element (Fig 4, sample 6), whereas the hall is a very large room with many salient items and many distractors (Fig 4, sample 8,9). We compare in Figure 6 the average percentage of time spent in each area when using RL-IAC and when using random (*RND+learn*) exploration strategies. In the case of random exploration, the time spent in each area is roughly the same all along the sequence. 40% to 50% of the time is spent in the corridor, whereas 10% is spent in the office. With RL-IAC, the time spent in the corridor (the less 'interesting' area) oscillates between 30% and 20%, except at the beginning, and almost 20% is spent in the office. Moreover, the time spent in exploring each area is evolving in time: The time spent in the office finally decreases to 0%, because the model does not make any more progress in there, in the middle of the sequence, most of the time is spent exploring the lab, while most of the time is spent in the hall at the end of the exploration.

[2]Our implementation was tested on Ubuntu 14.4 with an Intel Core i3-3240, CPU at 3.4GHz quadcore processor



(a) (b)

Fig. 3. Detection rate vs number of boxes proposal comparison on the *RGB-D scenes* dataset (a) and on the robot sequence (b).

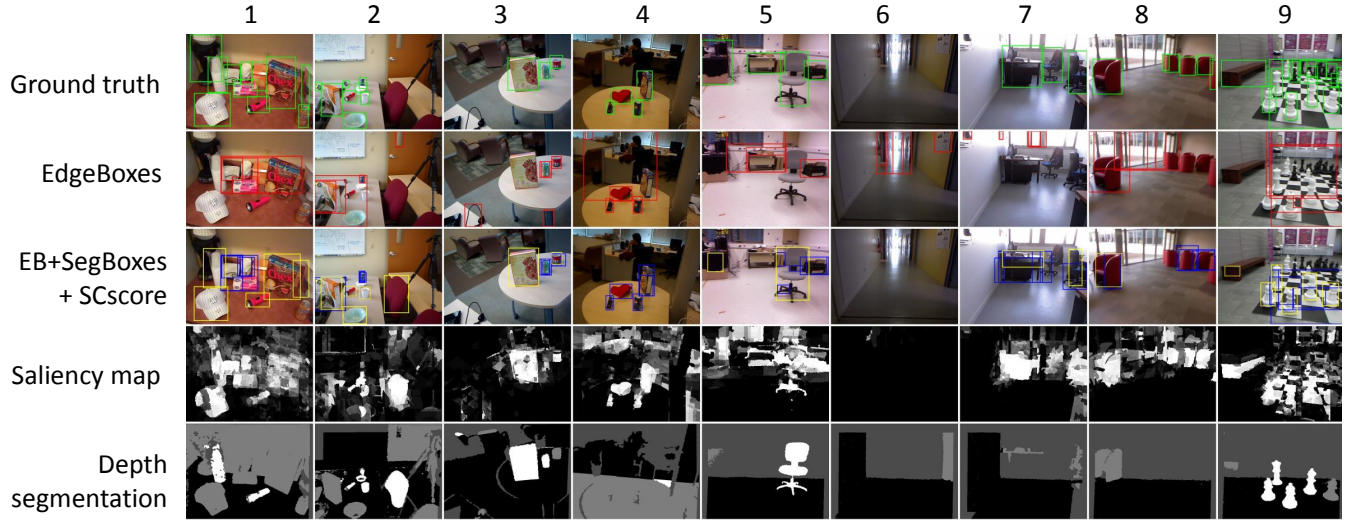
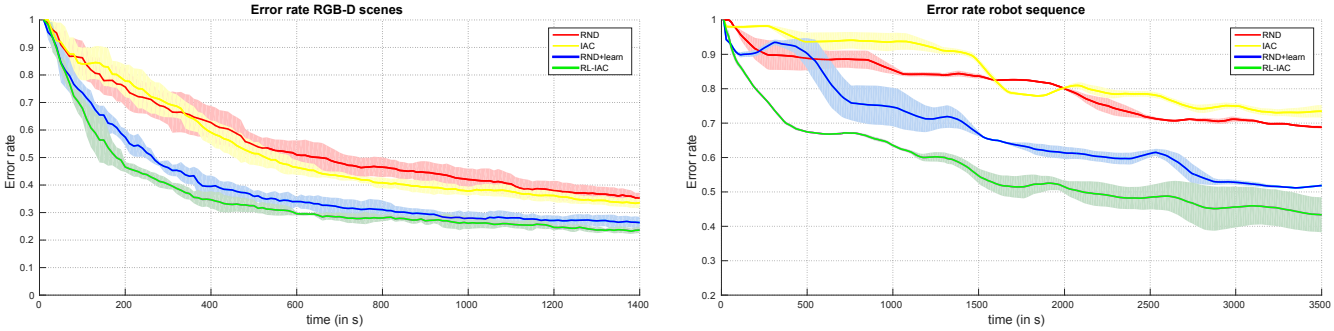
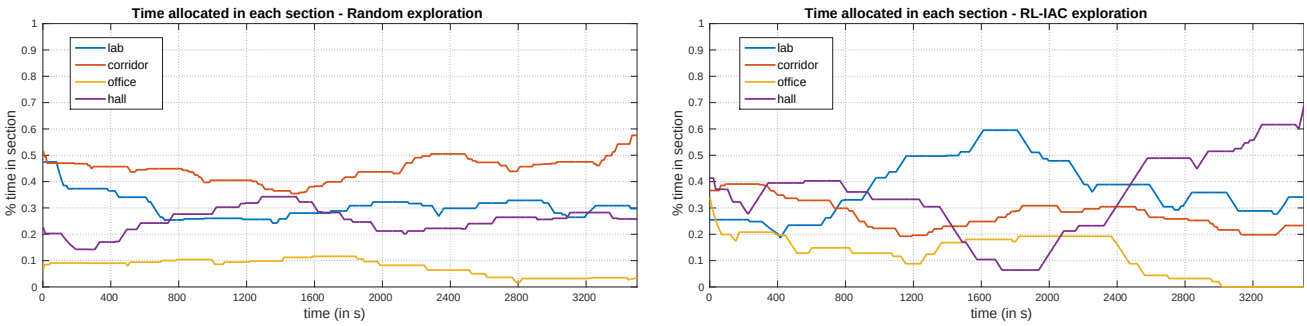


Fig. 4. Sample results for boxes proposal. First row: the RGB-input along with the ground truth bounding boxes. Second row: top 5 bounding boxes proposed by EdgeBoxes. Third row: in blue, the top 5 boxes proposed by EdgeBoxes+SCscore. In yellow, the SegBoxes. To get a better insight of the saliency bias and the SegBoxes, the corresponding saliency and segmentation are displayed in rows 4 and 5.



(a) (b)

Fig. 5. Overall error rate and variance evolution versus simulated time for a few exploration strategies. Results are obtained both on (a) *RGB-D scenes* artificial building and (b) our building



(a) (b)

Fig. 6. Time spend for (a) random position selection (b) RL-IAC

V. Conclusion and Future Work

In this paper, we have presented an approach to incrementally learn visual saliency, using an exploration strategy based on learning progress and reinforcement learning. We show that this method could be used to produce relevant bounding boxes around objects of interest that could be further used for recognition. To allow the robot to autonomously discover and learn about its environment, we proposed a method called RL-IAC, that finds the best compromise between spending time moving to an area with higher progress, or keep learning in a less progressing area nearby. We show that this type of exploration strategy makes learning faster and better than it would be with random exploration.

In a future work, we would like to investigate the use of other types of intrinsic motivation (such as novelty or uncertainty) to drive the robot's actions. The use of navigation graphs is also pretty restrictive as it forces an operator to manually determine regions and the robot's trajectories. An exploration that would incrementally generate regions and transitions would be more appropriate. We could also apply this framework with other definitions of saliency. Instead of a generic object segmentation, we could for example use objects detectors and specialize our saliency to find those objects within their environments.