Building a Curious Robot for Mapping

Lucas Lehnert
McGill University
lucas.lehnert@mail.mcgill.ca

Doina Precup
McGill University
dprecup@cs.mcgill.ca

Abstract

Curiosity towards exploring new objects in one's environment is a key driver of intelligent agents. We explore the problem of mapping in environments which are non-stationary, and where areas may exhibit different change patterns. This is an important challenge for potential "domestic" robots, which would have to perform tasks in houses. We propose a computational approach to building a curious agent which constructs a map of its environment and estimates how often a specific location in this map is changing. Using these estimates, the agent will actively try to reach rapidly changing areas more often than static areas. As a result, the behavior becomes more focused towards learning about significant changes in the environment. We present experiments in a robotic navigation framework implemented using ROS. The results show the utility of this framework for map acquisition and maintenance.

1 Introduction

Intelligent agents are often faced with the problem of *exploring* their environment, in order to learn how to achieve different goals. For example, an autonomous robot living in someone's home may have a map of its initial environment, but may need to adapt quickly to changes, such as a new piece of furniture or toys left on the floor. This type of situation requires the agent to explore its environment in such a way as to identify quickly any changes, and to adapt its representations accordingly. Efficient exploration has been a big focus in reinforcement learning, as well as in robotics (especially in map building). However, many existing algorithms equate "exploration" with "randomization". While randomized action selection is crucial in order to gather diverse data, it does not provide the type of exploration behavior seen in animals and humans, which tend to explore *persistently* only the aspects of the environment that have changed significantly (rather than exploring uniformly everywhere). For example, a child will pick up a toy that she has never seen before and inspect it for a long time, until she has figured out how to use it. Meanwhile, other existing toys will be ignored. Simple randomization in the action selection will not lead to this type of directed exploration, and instead will appear more chaotic. Our goal in this paper is to present a computational approach for exploration which mimics *curiosity*, i.e., taking actions in a focused way to detect novelty in the environment.

We use as a testbed the problem of robot mapping, in which a robot has to discover which parts of its environment contain obstacles. We provide the robot with two initial exploration strategies: recency-based and change-based exploration. Recency-based exploration was introduced by Thrun [8] and drives a robot towards parts of the state space that have not been visited in a very long time, under the assumptions that they may have changed since then. Change-based exploration explicitly drives a robot towards parts of the environment in which change is known to happen (even if the robot might have been there recently). Specifically, we propose an approach that estimates change in different parts of the environment using Poisson processes. Processes with high rate indicate rapid changes. We present an adaptive algorithm which learns how to choose the best exploration policy (recency or change-based), using reinforcement learning. The approach is generic and could be used with a larger set of exploration strategies as well. We experiment with the proposed approach on a faithful

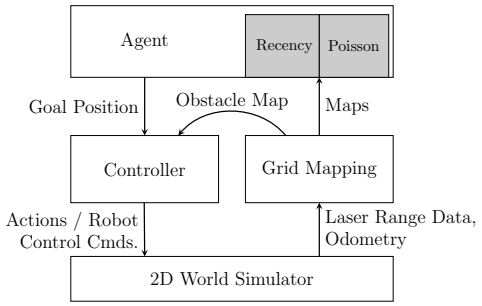

Figure 1: Overview of the learning agent

robot simulator, implemented in ROS [4]. The results show that the proposed algorithm identifies changes in the environment efficiently, and is superior to any of the individual strategies on its own.

2 System overview

Figure 1 presents a general overview of the learning agent, which is split into three modules. The *grid mapper* takes in sensor data from the simulator and maps an occupancy grid map. We experimented both with a grid world simulator and a Stage robot simulator. If the Stage simulation is used, the mapper first converts the laser range reading into a point cloud using the laser_geometry ROS library. Based on the changes that a sensor data update causes to the obstacle grid map, the Poisson rate maps and all other maps used by the exploration strategies are updated in this module as well. The obstacle grid map is sent over to the controller module and all other maps are sent over to the agent. We describe this module in more detail in Sec. 3.

The *controller* takes a goal position from the agent, navigates the robot to this position and explores locally around it, using a local controller. In the grid world setup, this controller takes in the obstacle data from the mapping module, and the ROS tf-transform from the simulator (which lets the user keep track of multiple coordinate frames over time). Once a goal position is received, the controller plans a path to it from the current position, using the most recent map. The planner is a simple wave front planner, i.e. a wave front starting from the goal position is created and then a path is computed using hill climbing from the current position to the goal. If the goal is blocked or hidden by walls, the planned path takes the robot as close as possible to the goal. From the path (which is a sequence of positions), an action sequence is computed using the known transition model of the robot, and the actions are sent to the simulator. Once the goal state is reached, the local controller takes over to perform local exploration. For this, the cells that were changed during the last sensor update are recorded, their centroid is computed and then used as the next goal position. If no map changes were made, one step is made randomly. The number of steps the controller can make is bounded by a constant, and the number of consecutive random steps is also bounded. Once the maximum number of steps is reached, a termination signal is sent to the agent and the controller waits for a new goal position.

The *learning agent*'s task is to first decide an exploration strategy. From this, it creates a probability distribution over the robot's map and samples the next goal position for the controller. Sec. 4 describes in detail the exploration strategy.

3 Mapping

The mapping module takes in laser range data and tf-coordinate transforms to update its obstacle grid map. It keeps a list that stores if and how a cell in the obstacle map is changed. This list is used for learning the rate of change at each cell the obstacle map.

To implement change-based exploration, two Poisson processes at every grid cell in the obstacle map, to estimate the rate of change. One Poisson process models the change of a cell from blocked to free, the other models the change from free to blocked. It is necessary to distinguish the change of a cell into a blocked-to-free and a free-to-blocked event because a cell may only spend a short amount

of time in a blocked state and a long amount of time in a free state (i.e. if a block moved through the world, it will appear at one cell only for a short amount of time). This cannot be represented using only one Poisson process. As shown in Figure (2), these events are strictly alternating.

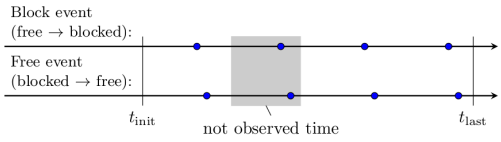

Figure 2: Two Poisson processes model used to learn the rate of change at each grid cell.

To compute a rate estimate at a grid cell, the amount of time a cell is observed and the number of observed events is stored. The average time interval between events is computed for each process at each grid cell c as:

$$\mathbb{E}[T^c_{\text{block}}] = \frac{t^c_{\text{last}} - t^c_{\text{not obs.}} - t^c_{\text{init}}}{n^c_{\text{block}}} \text{ and } \mathbb{E}[T^c_{\text{free}}] = \frac{t^c_{\text{last}} - t^c_{\text{not obs.}} - t^c_{\text{init}}}{n^c_{\text{free}}},$$

where t^c_{init} and t^c_{last} are the first and last time cell c was observed, and $t^c_{\text{not obs.}}$ is the amount of time the cell was not observed between the first and last time (i.e. when the robot has explored a different area). $t^c_{\text{not obs.}}$ is important to factor into this computation, otherwise the interval estimates will have a large bias. n^c_{block} is the number of block events observed, and n^c_{free} is the number of free events observed. Since these estimates tend to be biased towards larger intervals, and as both Poisson processes are coupled (and therefore must have the same rate), the estimate T^c used at a grid cell c is:

$$T^c = \min(\mathbb{E}[T^c_{\text{block}}], \mathbb{E}[T^c_{\text{free}}]). \tag{1}$$

Let E be the number of change events (either a blocked-to-free or a free-to-blocked event) that happened at c since the last time it was observed. E will have a distribution $\text{Po}(\lambda = 1/T^c \cdot (t^c_{\text{now}} - t^c_{\text{last}}))$ and the probability of an event at c is therefore

$$P_c(E > 0) = 1.0 - e^{-(t^c_{\text{last}} - t^c_{\text{init}})/T^c}. \tag{2}$$

To sample the next goal position, $P_c(E > 0)$ are considered as weights at every grid cell, and use to construct a probability distribution over the whole map. This distribution is discretized more coarsely by grouping square patches of change probabilities together and averaging their values, in order to smooth out noise. This distribution is then used to sample the next goal position, if the Poisson based exploration policy is used.

The Recency Based Exploration strategy navigates the robot into areas that were not observed recently. At every time step, each cell c that is observed is flagged with a timestamp t^c_{last} (this includes the current cell). For sampling a goal position, a recency weight is computed at every cell as: $r^c = t_{\text{now}} - t^c_{\text{last}}$ where t_{now} is the current time. These weights are then used to construct a probability distribution over the whole map. from which to sample a goal position.

4 Proposed method

The robot first decides on an exploration strategy and then uses it to sample a goal position. The strategy choice is based on two real-valued, non-negative features:

$$\mathbf{p} = \sum_{c \in M} P_c(E > 0) \text{ and } \mathbf{r} = \max_{c \in M} r^c.$$

Intuitively, a high $\mathbf{p}$ suggests that more changes have happened that were not observed, which in turn favours Poisson exploration. A high $\mathbf{r}$ suggests that a cell has not been visited for a long time, which should favour recency-based exploration. Reinforcement learning is used to determine the strategy choice based on these two features. Since the features are continuous, a tile coder with five overlapping tilings is used to approximate the value function. The tilings have random displacement; each tile has a size of 0.5×100.0, i.e. the $\mathbf{p}$ value is scaled up (divided by 0.5) and the $\mathbf{r}$ value is scaled down (divided by 100.0). The algorithm for learning which strategy to use is summarized below.

Pick a strategy S uniformly at random.
loop
  Determine the next goal position using strategy S and send to controller module.
  Let $\mathbf{T} = \{\}$ be an empty sequence of feature pairs $(\mathbf{p}, \mathbf{r})$. ▷ Also called trajectory.
  while Controller module navigates robot **do**
    Update all maps and parameters in mapping module and receive maps.
    Compute $(\mathbf{p}, \mathbf{r})$ according to Equations (4).
    Append $(\mathbf{p}, \mathbf{r})$ to $\mathbf{T}$.
  end while
  Evaluate $\mathbf{T}$ with a reward $R = \frac{\text{number of map changes}}{\text{length of } \mathbf{T}}$.
  for each $(\mathbf{p}, \mathbf{r}) \in \mathbf{T}$ **do**
    $V_S(\mathbf{p}, \mathbf{r}) \leftarrow V_S(\mathbf{p}, \mathbf{r}) \cdot (1 - \lambda) + R \cdot \lambda$ ▷ $\lambda = 0.9$ is a parameter.
  end for
  Choose the next strategy S with probability $P(\text{select } S) \propto \exp(V_S(\mathbf{p}, \mathbf{r})/\tau)$.
  ▷ $\tau = 2.0$ is a parameter.
end loop

The whole exploration process is subdivided in time, with each loop iteration representing the exploration of a specific area. The agent does not update and re-decide after each time step, as there are long periods during exploration when nothing new is observed. As the agent tries to maximize the number of map updates, these time steps can cause it to unlearn previously learned values, due to the rewards being zero for many time steps. The algorithm we adopted does not have this issue, as it waits for the exploration of a local area to complete and then gives a reward with uniform weighting over the whole trajectory of feature pairs.

Further, the fact that the learning agent only has to choose a goal state simplifies the learning task. Computing actual velocity commands, handling obstacle avoidance, and executing recovery behaviors if the robot gets stuck, will all be handled in this controller, so the learning task itself is simplified.

5 Experiments

The Curious Exploration Algorithm was tested first in a grid world simulator, on different map configurations, and its performance was compared with a version using only the Recency-based or only the Poisson-based exploration strategy. In all experiments the robot started in the center of the map (position $(0, 0)$) and every experiment was repeated five times. We present results for the sample map in the left panel of Figure 3, which has the property that the motion of the objects we hope to discover is fairly wide spread and not clustered in a particular area. The rate of change of the cells in the two areas is also different, which makes the exploration problem harder. The data shown in Figure 3 was collected with a controller that was allowed to make at most 42 steps, which is the length of the longest path of a block in the world (the controller was not allowed to make 2 or more consecutive random steps). This particular grid world is deterministic, but we note that very similar results were obtained in a grid world with stochastic transitions (omitted for lack of space).

Figure 3 shows that the Curious Exploration algorithm learns the rate of change of a grid cell faster than the versions using only the Recency Based and Poisson Based exploration strategies. The Poisson Based exploration strategy is interested only in navigating to changing areas. At the beginning of the exploration process, it will first randomly zig-zags until a first change is detected. This will cause the agent to find a small change interval (high rate of change) at one or a few cells. Then, it will stop moving randomly and only sample these few changing cells as goal states. This strategy locks down on an area and therefore its overall map quality is lower. Recency-based exploration causes the robot to zig-zag through its world , continually, which yields a complete map, but the behaviour is more chaotic.

The configuration of the controller has an influence mostly on the performance of the Poisson Based exploration strategy. If the controller is allowed to follow changes in its surrounding cells for a sufficiently long amount of time, it will (often) follow the moving block. So if only the Poisson Based exploration strategy is used and the agent passes by the moving block, this block will be followed and the changing cells will be traversed. This will help the Poisson Based exploration strategy to extend its map and capture more cells that are changing. As this strategy causes the agent

to stay within the changing area, it learns the rate of changes faster than Recency-based exploration would. So if the robot spends more time on observing changing areas, the rate estimates are learned faster. However, observing an actual change in the occupancy map is not necessary for that.

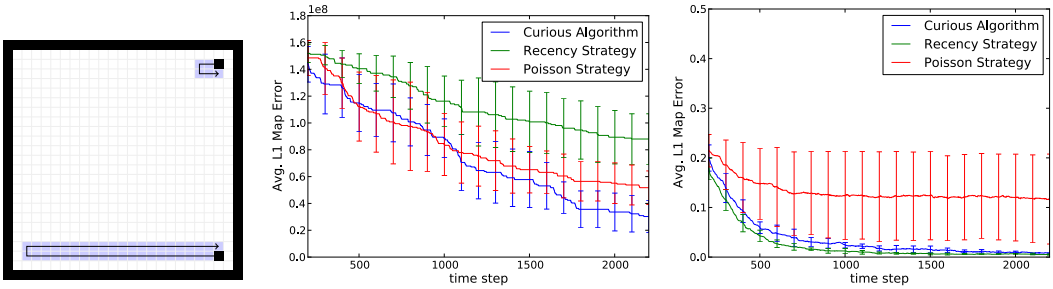

Figure 3: Evaluation Results for the 25x25 grid test map with two moving blocks shown in the left panel. The top block changes position every 7 time steps and moves over 6 positions, the bottom block changes position every 2 time steps and moves on a cycle along the bottom wall. The Curious Exploration algorithm learns the rate of change of each cell in its map faster than a version only using the Recency based or Poisson based exploration strategies (middle plot). In terms of map quality, the Curious Exploration algorithm performs as well as using the Recency Based exploration strategy (right plot).

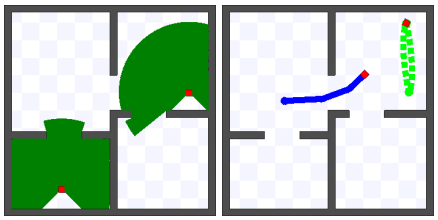

Figure 4: Stage Simulation Configuration. Left: the starting position of the robot that ran the Curious Mapping module (robot in lower left) and a 'dummy' moving robot (upper right). The green shaded areas are the field of view of the laser range sensors. Right: the path the 'dummy' robot (green trace).

Curious exploration was also tested in a Stage simulation, in order to evaluate its performance in a more realistic scenario. In this simulation, a 12×12 meter map was used, which divided the robot's world into four rooms (see Figure 4). A diff-drive robot was configured to move along a wall and go back and forth between two pre-configured coordinates. This robot also had a laser range sensor to detect obstacles in its path. If its path became temporarily blocked, the robot stopped moving until its path became free again. The curious exploration mapping module was then run on a second diff-drive robot with a laser range sensor. This robot was manually operated to move it through the world. The laser range sensor was configured to have a range large enough to completely cover a room. To estimate its position, the mapping module used noise-free odometry data, i.e. the data had an error with zero variance and it did not accumulate observation bias.

Figure 5 shows the maps created in Stage. The left panel shows that the created occupancy map is fairly accurate and contains little noise. Tthe rate map (center map) shows that the path of the moving robot was correctly detected by estimating a shorter change interval at cells that correspond to coordinates on the moving robot's path. The noise in the updates of the occupancy map also caused the mapping module to detect higher rates of change in areas that correspond to static walls in the Stage world. However, this noise can be removed by using the occupancy map. The change probability map on the right in Figure 5 shows that high probabilities are only estimated at cells that correspond to positions visited by the moving robot.

6 Discussion

We presented an approach for tracking changes in an environment which is non-stationary and changes over time. We use reinforcement learning to train a robot choosing between two different

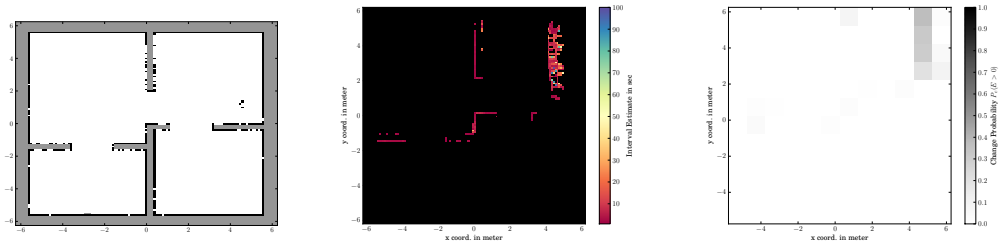

Figure 5: Obstacle and interval map learned in Stage experiment. The red thick stripe in the top right corner of the rate map shows that the trajectory of the moving robot was detected and that a higher rate of change in this area was learned. Due to discretization errors in the mapping, the rate map will have smaller interval estimates for wall cells in the robot's world. However, this noise can be removed using the obstacle map.

exploration strategies: one based on recency, the other based on detecting the rate of change in the environment. Our approach is geared towards continual mapping of a non-stationary environment, in which the map has to be continually updated.

The most related research topic to this work is Simultaneous Localization and Mapping (SLAM) (see Durrant-Whyte & Bailey, 2006a,b[2, 1] and Grisetti et al, 2011 [3], for excellent tutorials on this topic). SLAM is a cornerstone of modern robotics, in which a robot has to simultaneously learn a map of its environment and learn how to localize. In SLAM, most of the interesting information is acquired in the early phases of the algorithm. In contrast, we focus on adapting the robot's knowledge as quickly as possible when changes are detected, and on obtaining behavior that generates persistent exploration patterns. Note that although we focus here on map learning, the proposed approach is generally applicable for artificial agents that have to learn predictions about the states of their environment.

In reinforcement learning, several exploration strategies have been proposed (see Sutton & Barto, 1998 [6] and Szepesvari, 2010 [7] for overviews of these methods). However, the goal there is different, in that the agent is assumed to be in a stationary environment, in which its goal is to maximize a measure of long-term performance. In contrast, we focus here on non-stationary environments. The closest work in spirit to our approach is that in intrinsically motivated reinforcement learning (Singh et al, 2004 [5], and follow-up papers). In that setup, rewards are designed for the agent in order to encourage it to discover "optimal behavior". These rewards may contain a component which encourages exploration. However, the goal of this approach is still (ultimately) to maximize some kind of reward. Our work is more focused on curiosity for the sake of improving the agent's internal representation, without a specific reward maximization goal in mind. Future work will include a direct comparison with these methods.

References

[1] T. Bailey and H. Durrant-Whyte. Simultaneous localization and mapping: Part ii. *IEEE Robotics & Automation Magazine*, 2006.

[2] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: Part i. *IEEE Robotics & Automation Magazine*, 2006.

[3] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2011.

[4] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[5] S. Singh, A. G. Barto, and N. Chentanez. Intrinsically motivated reinforcement learning. In *NIPS*, 2004.

[6] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT Press, 1998.

[7] C. Szepesvari. *Algorithms for Reinforcement Learning*. Morgan and Claypool, 2010.

[8] S. B. Thrun. Efficient exploration in reinforcement learning. Technical report, 1992.