A Hierarchical Maze Navigation Algorithm with Reinforcement Learning and Mapping

Tommaso Mannucci Erik-Jan van Kampen
Faculty of Aerospace Engineering, Delft University of Technology, 2629HS Delft, the Netherlands

***Abstract*—Goal-finding in an unknown maze is a challenging problem for a Reinforcement Learning agent, because the corresponding state space can be large if not intractable, and the agent does not usually have a model of the environment. Hierarchical Reinforcement Learning has been shown in the past to improve tractability and learning time of complex problems, as well as facilitate learning a coherent transition model for the environment. Nonetheless, considerable time is still needed to learn the transition model, so that initially the agent can perform poorly by getting trapped into dead ends and colliding with obstacles. This paper proposes a strategy for maze exploration that, by means of sequential tasking and off-line training on an abstract environment, provides the agent with a minimal level of performance from the very beginning of exploration. In particular, this approach allows to prevent collisions with obstacles, thus enforcing a safety restraint on the agent.**

I. Introduction

Reinforcement Learning (RL) [1] is a bio-inspired branch of machine learning. RL has been chosen for many application [2] due to its simplicity and adaptability, since assigning the reward function is usually simpler than solving the problem beforehand, and since it allows to solve tasks that are time-varying or that present uncertainties. However, one of the well known drawbacks of RL is that, at the start of learning, the policy is usually inefficient since the agent's knowledge of the environment is incomplete. Extensive exploration can be necessary before a task is correctly performed, depending on its complexity: the *curse of dimensionality* dictates that learning times grow exponentially with the number of states, which poses a problem when considering large and discrete, or continuous state spaces. This initial "blind search" can limit the applicability of RL algorithms, for which keeping learning time to a minimum is therefore a strong requisite.

The branch of Hierarchical Reinforcement Learning (HRL) introduces *abstraction* in the learning process: the ability to find common features between states and actions for a given task, as well as between tasks. Although multiple instances of HRL exist in literature, Barto [3], reports of three main approaches: *options* [4], [5] are sequences of actions that can be appended to the action set $\mathcal{A}$, which perform specific subtasks; Hierarchies of Abstract Machines (HAMs) [6] put restrictions on the space of realizable policies through *machines* that learn how to achieve specific subtasks, so that the RL problem becomes optimizing the machines and the transitions between them; MAXQ [7] instead directly divides the complete task into subtasks, and assigns to each its own pseudo-reward. Regardless of the specific method, there are several advantages to learning when abstraction is performed. The agent can individuate the relevant features of the state space to appropriately perform the task. Abstraction can also help the agent when performing sequential tasks by addressing which region of the state space is most relevant at a given time; this is especially useful when considering tasks that share one or more intermediate subtask. Finally, abstraction can be useful in reducing the effects of stochasticities.

Indoor navigation is a field where both autonomous [8] and semi-autonomous [9] platforms can benefit from the application of HRL. In particular, maze navigation exemplifies the advantages of abstraction during learning. Applying classical, "flat" RL algorithms to mazes usually implies very long learning times, due to the typically large state space of this tasks, and due to the difficulty of implementing appropriate function approximators. However, the problem can be simplified by abstraction. For example, an agent that had learned how to avoid one obstacle could reapply this knowledge to avoid similar ones. Higher levels of abstractions can be devised; e.g. if the agent has learned what constitutes a wall, it can learn wall-following behavior, and remember which walls have been previously followed. These and other relatively simple abstractions can be used to speed up learning; however, this abstraction very often originates from the designer itself. For example, to implement wall or corridor following behaviours, the designer needs to properly define which actions adhere to the behaviour and devise an appropriate reward. The reward must be tailored as well to teach the agent when to adopt and terminate a learned behaviour (e.g. when to stop wall following); alternatively, switching conditions between behaviours must be predefined. Parr [6] shows this kind of behavioral-based approach when applied to a maze.

This paper presents a different approach to hierarchy, inspired by the *layered control* philosophy of Brooks [10] and by the top-down hierarchy of Feudal Reinforcement Learning [11]. The strategy is to implement a highly abstracted "lowest-level" controller that needs the least possible amount of task and environment knowledge in order to be applied. This controller can therefore be extensively trained off-line before any actual real-life implementation. The resulting zero-time initial policy might be sub-optimal, but the initial blind-search will be prevented. The properties of such an approach are illustrated in a goal-finding task inside a relatively large maze. Using mapping and RL to promote exploration combined with a hierarchical navigation policy results in an agent that efficiently navigates the unknown environment with minimal

learning time; additionally, undesired occurrences such as collisions are minimized during the actual task.

The remainder of the paper is structured as follows. Section II presents fundamentals for the methods applied, i.e. RL, HRL, and mapping. Section III illustrates in detail the approach. Section IV presents the experimental setup upon which the algorithm is tested, the results of which are discussed in Section V. Finally, Section VI concludes.

II. Fundamentals

A. RL and HRL

Let $\mathcal{S}$ and $\mathcal{A}$ be respectively the set of *states* $\boldsymbol{s}$ and *actions* a. $\mathcal{R}(\boldsymbol{s}, a, \boldsymbol{s}')$ is the *reward function* assigning reward r after transition $\boldsymbol{s}, a \rightarrow \boldsymbol{s}'$, which is given by the probability function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0\,,\,1]$. A policy $\pi(\boldsymbol{s}, a) : \mathcal{S} \times \mathcal{A} \rightarrow [0\,,\,1]$ is optimal if $\forall \boldsymbol{s}$ it maximizes the value function

$$V^{\pi}(\boldsymbol{s}) = E\left(\sum_{k=0}^{\infty} \gamma^k r_k\right),$$

where $\gamma \in (0\,,\,1)$ is the *discount*. These five elements constitute a *Markov Decision Process* (MDP). Among other RL algorithms that can solve MDPs, *temporal-difference* methods approximate the value of states $V(\boldsymbol{s})$ with

$$V^{\pi}_{k+1}(\boldsymbol{s}) = V^{\pi}_{k}(\boldsymbol{s}) + \alpha\left(r_k + \gamma V^{\pi}_{k}(\boldsymbol{s}')\right)$$

upon transition from $\boldsymbol{s}$ to $\boldsymbol{s}'$. The optimal solution is then obtained by *iterating* and *evaluating* the policy.

HRL is a branch of RL that is loosely connected by the concept of *extended temporal actions* or *macro-actions*, i.e. activities extended in time. These introduce *temporal abstraction* into learning, e.g. how to solve subtasks. HRL differs from "flat" RL in that it is not defined on MDPs but the more general *Semi-Markov Decision Processes* (SMDPs) [4], an extension of MDPs that includes extended temporal actions.

B. Mapping and maze exploration

Mapping means using sensor readings to generate an internal representation of the environment. Typically, the agent starts exploration with a small map of its immediate surroundings or with no map at all, but expands on this as more measurements are performed. Two main representations for maps exist. As the name suggests, *grid-based* maps partition the environment into an evenly spaced grid. In theory, each cell can be observed or unobserved, and observed cells can be either empty or occupied. However, in order to cope with reading errors, occupancy of a cell is often treated as a continuous value, e.g. in *evidence grids* [12]. If the value of occupancy of a cell is below a set threshold it is empty, otherwise it is full. Grid-based maps constitute a quantitative representation for an environment, while *topological* maps are a qualitative one. These maps do not use grids but graphs: vertices represent locations, such as rooms, corridors, or landmarks; edges indicate which locations are reachable from one another. Hybrid representations known as "grid-topological" [13] have been attempted as well in the hope of combining the simple interpretation and use of topological maps with the more precise nature of grids.

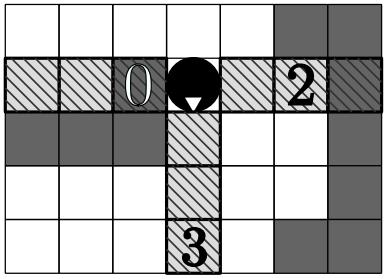

Fig. 1: An example of sonar reading. Sonars are positioned at the front, right, and left of the agent, and detect obstacles in a straight line with a range of up to three squares.

III. Approach description

This section explains the proposed approach. After briefly describing the agent, the two main elements of the approach are illustrated: the hierarchical control for local navigation, and the global exploration strategy, based on mapping and RL.

A. Agent description

The agent considered in this paper represents a ground robot equipped with distance sensors. The robot is assumed to be able to travel at constant speed in the direction it is facing, to brake with negligible braking distance, and to turn on the spot. The discrete primitive actions of the agent are then advancing one square, turning 90° clockwise, or turning 90° counterclockwise. The agent navigates with the aid of a map; this is not provided beforehand but is built online, based on the sensor readings. Three close range distance sensors (e.g. sonars) are positioned at the front, right and left of the robot, detecting the nearest obstacle within three squares of distance in a straight line (see figure 1). A compass detects which of the four cardinal directions the agent is facing at each time. Assuming perfect readings of the sensors, a map is built progressively which is consistent with the environment. It is assumed that the robot performs perfect odometry, so that the agent knows its position in the map at all times.

B. Hierarchical control for local navigation

Local navigation is performed through hierarchical control, which is obtained by repeatedly applying a single, simplified low level controller, on multiple levels of abstraction.

This low-level policy is obtained off-line in a "minimalist" environment, a simple *minigrid* (figure 2) four squares long and three squares wide, which is used to teach basic navigation to a *trainee* agent. The squares of the minigrid can contain fixed obstacles, in which case they are impassable. The trainee will learn how to avoid obstacles to reach the goal positions $p_g{=}p^1_g$, p^2_g, p^3_g, of the fourth row.

The training is divided into episodes, at the start of which the trainee is initialized at a random, empty position p_{st} in one the first three rows. Obstacles are then positioned randomly in the remaining squares. Therefore, a total of

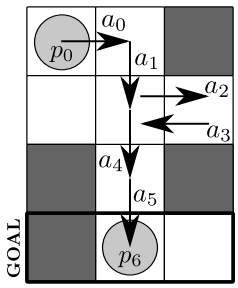

Fig. 2: In this sample episode, the environment contains four occupied squares, included a goal state. The agent starts in p_0 and descends towards goal p_6 in six steps. As the agent performs a deviation with action a_2, the obtained trajectory is suboptimal.

$2^{12} - 2^3 = 4088$ obstacle placements $\mathcal{O}$ are possible. The trainee can then attempt moving from p_{st} to any adjacent square by conventional actions `up`, `down`, `right`, `left`. If the square is empty, the trainee moves with probability one and receives a reward of -1. Otherwise, the action is not applied for this iteration. The trainee chooses a random action with probability 0.1, and otherwise takes the action that, during the episode, has been selected the least amount of times from its current position. An episode terminates as soon as the trainee reaches a goal position or, if including discarded actions, the 100^{th} iteration is reached.

It should be noted that the trainee is not exactly the same agent described in the previous paragraph. It is instead an abstracted version of it, which does not take take orientation into account. This is reflected in the different actions available to the two. Nonetheless, all trajectories simulated in training by the trainee are reproducible in the maze by the agent using primitive actions.

Given k the number of moves in an episode, the trainee performs k trajectories τ=$\{\mathcal{O}, p_i, a_{i:k-1}, R, p_k\}$. Each trajectory is a motion pattern starting from position p_i at time i, all terminating in p_k at time k, after sequence of actions $a_{i:k-1}$. Each trajectory yields an undiscounted return R=$i-k$. If the episode terminates before the 100^{th} iteration, p_k is always one of the goals p_g. The trainee can then use each trajectory as an *option* [4] when in position p_i to get to goal p_k, given obstacles $\mathcal{O}$, receiving a return R. The same motion is also reused whenever possible by applying appropriate transformations to the environment; e.g. consider placement $\mathcal{O}$ and its mirrored image $\mathcal{O}'$ obtained by switching the first and third column of the minigrid. A trajectory that reaches a goal in $\mathcal{O}$ can be used (opportunely mirrored) to reach a goal in $\mathcal{O}'$. Consider again figure 2, in which the trainee successfully reaches the goal in six moves, and therefore six trajectories are observed:

$$\{\mathcal{O}, p_0, a_{0:5}, -6, p_6\},\ \{\mathcal{O}, p_1, a_{1:5}, -5, p_6\},$$
$$\{\mathcal{O}, p_2, a_{2:5}, -4, p_6\},\ \{\mathcal{O}, p_3, a_{3:5}, -3, p_6\},$$
$$\{\mathcal{O}, p_4, a_{4:5}, -2, p_6\},\ \{\mathcal{O}, p_5, a_5, -1, p_6\}.$$

To these, six trajectories are added for right-left mirrored environment $\mathcal{O}'$. Additionally, five trajectories are found for the top-down mirrored environment $\mathcal{O}''$, where goal p_6 is replaced by position p_1. Finally, five more trajectories are obtained for the environment $\mathcal{O}'''$ obtained rotating the obstacles by 180°. In total, 22 valid trajectories are found in four different environments with a single episode of training.

In case the iteration limit of 100 is reached, it is assumed that the goals can not be physically reached, given obstacles $\mathcal{O}$, from starting position p_{st} and all positions encountered afterward in the episode. These trajectories are also used to teach the trainee, with the following modifications. First, R is replaced by -100 for all τ, as an indication that they all fail in reaching the goals. Second, each trajectory τ is replaced by three trajectories with $p_k = p_g^{1,2,3}$. Unsuccessful trajectories will then be in the form: $\{\mathcal{O}, p_i, a_{i:k-1}, -100, p_g^1\}$, $\{\mathcal{O}, p_i, a_{i:k-1}, -100, p_g^2\}$ and $\{\mathcal{O}, p_i, a_{i:k-1}, -100, p_g^3\}$.

Valid trajectories are recorded by means of a table $\mathcal{T}$: $\{\mathcal{O}\} \times \{p_i\} \times \{p_k\} \times \{a_i\} \rightarrow \mathbb{R}$ that maps the combination of environment, starting and ending positions, and first action of each trajectory to the return R. The $4088 \cdot 9 \cdot 3 \cdot 4$ entries are initially empty. At the end of an episode, for each trajectory τ the content of the corresponding entry in $\mathcal{T}$ is compared to the new return R: if it is empty or lower, then it is replaced by R. After convergence, $\mathcal{T}$ indicates if a goal p_k can be reached for given environment and position, and the number of moves necessary to do so when starting with action a_i.

The table $\mathcal{T}$ can then be used to navigate the agent inside a grid, provided that:

- the agent has a map with the surrounding obstacles;
- a local direction of motion is assigned.

Navigation is performed as follows. The agent places a minigrid inside the map, so that the current position of the agent is inside the first three rows, and the goals are in the local direction of motion. Setting aside for the moment the presence of unobserved squares, assume that the map provides the placement $\mathcal{O}$. Then, according to the current position p_i inside the minigrid, $\mathcal{T}$ is consulted to obtain the optimal trainee action `up`, `down`, etc., which is then decomposed into the equivalent primitive actions. Figure 3 illustrates an example of navigation from state A1 to state A9. The agent is initially in A1, and is instructed to go South. It positions the minigrid between A1-D3 an reaches goal D1. Then, it is commanded to proceed East, so it positions the minigrid in D1-F4. This time multiple goals are available, and the agent ends in E4. Following the same identical steps, it then continues East through D4-E7, and finally North through A7-D9, reaching its goal.

As it can be seen, proper maze navigation can be obtained by iterating the policy contained in $\mathcal{T}$. However, determining a direction of motion and positioning the minigrid appropriately is not trivial. For example, the direct route between A1-A9 in figure 3 is blocked by an obstacle. The agent must understand the presence of the obstacle, and then circumnavigate it, in order to avoid unnecessary or even unsafe actions (e.g. collisions).

This can be achieved by applying the policy $\mathcal{T}$ of the minigrid on a higher level of abstraction. Consider partitioning the map into groups of 9-by-6 (or 6-by-9) squares, and each

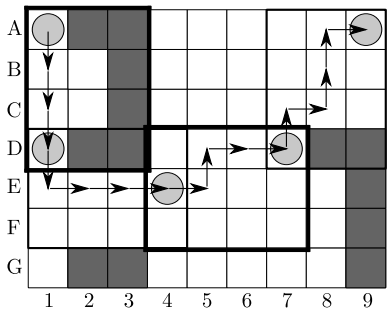

Fig. 3: Sample of maze navigation obtained through iteratively adopting the minigrid logic of table $\mathcal{T}$.

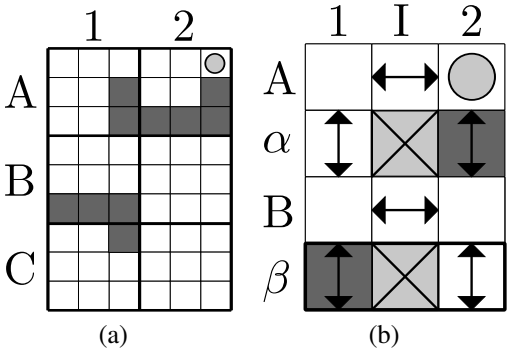

Fig. 4: Transitions between tiles (a) are abstracted into a macrogrid (b), which allows to circumvent obstacles.

group into 6 *tiles*, so that each tile is a group of 3-by-3 squares. Figure 4a shows one such group and its 6 constituent tiles A1, A2, B1, etc. In the same way that the policy $\mathcal{T}$, applied to the minigrid, indicates how to transition between squares, the policy is applied to a *macrogrid* to move the agent between tiles. As with the minigrid, the macrogrid is composed of 4-by-3 cells, which however do not represent squares of the map. Cells A1, A2, B1 and B2 (figure 4b) represent the tiles with the same name. If the agent is in any square of a tile, it is considered to be inside the corresponding cell of the macrogrid. The remaining cells (with the exception of αI and βI) do not represent tiles, but transitions between tiles. If a transition cell is occupied, it is not possible to move between the two adjacent tiles. For example, in figure 4, obstacles prevent paths A2-B2 and B1-C1, so α2 and β1 are occupied. Occupancy of transition cells can be assessed by the agent by positioning a minigrid so that the first three rows overlap one of the two tiles, and then checking $\mathcal{T}$ to see if any goal can be reached. Finally, cells αI and βI represent invalid transitions, and as such are always considered occupied. The use of tiles and macrogrids introduces temporal abstraction, since moving between tiles is an extended temporal action.

Figure 5 shows an application of the combined hierarchical control when the map contains unobserved squares, in which the agent is exploring towards the East. A grid of 3-by-2 tiles is then placed on the map, so that the goal tiles are located in the direction of motion, and the agent is initially in the top-right corner of tile A2. In order to promote exploration,

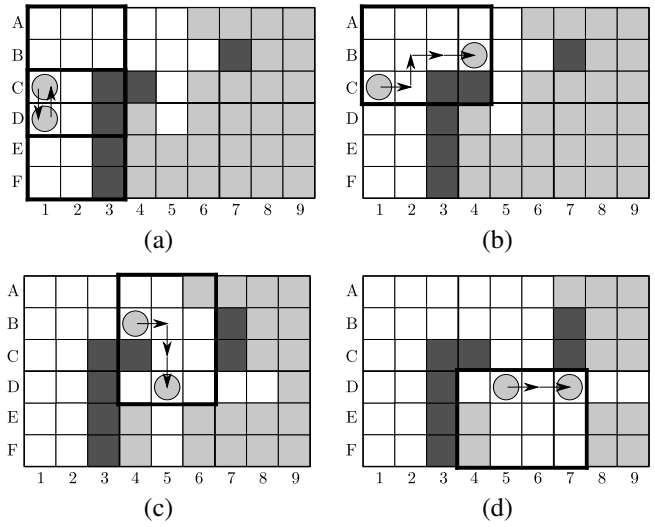

Fig. 5: In (a), the agent receives instructions from the macrogrid to which it cannot abide due to the presence of obstacles. A new path is then issued and executed until in (b), the agent declines the given instruction a second time, as it might not be able to reach goals A7, B7 or C7. The macrogrid computes a third and final path which is then carried on with success in (c) and (d).

all transition cells of the macrogrid are initially considered to be empty. Once a transition is issued by applying $\mathcal{T}$ to the macrogrid, a minigrid is positioned as to include the current tile and the three neighbouring squares of the destination tile, which therefore represent the goals. Table $\mathcal{T}$ is then consulted again to move between squares. If the minigrid contains any unobserved squares, these are considered occupied, in order to prevent collisions with unobserved obstacles. It can be that, due to the presence of occupied or unobserved squares, the transition indicated by the macrogrid cannot be completed. The cell representing the failed transition is marked as occupied inside the macrogrid, which is then consulted again to see if an alternative path is available. If no path is available, a different direction must be explored. For example, in figure 5a, the initial suggested transitions are `South`, `East`, `East`. The `South` transition is immediate. For the following move to `East`, however, the minigrid returns -100, meaning that the transition is impossible. The corresponding transition cell is again marked as occupied; the macrogrid then reroutes as `North`, `East`, `East`. The agent goes up one tile to the original position, and then one tile right (figure 5b). The following transition `East` is again denied by the minigrid, since the unobserved squares might contain obstacles. The macrogrid then reroutes `South` and `East`, which the agent executes in figures 5c and 5d.

In theory, the policy given by $\mathcal{T}$ could be reiterated further, by performing abstraction at increasingly higher levels. Here, the hierarchical structure of control is limited to the two levels of square navigation (no abstraction) with minigrids and tile navigation (one level of abstraction) with macrogrids discussed above. The map-based strategy of the following section com-

pletes the approach by determining which direction to explore.

C. Exploration strategy

In addition to the local navigation policy given by $\mathcal{T}$, a global exploration strategy is necessary to find the goal efficiently. This section illustrates how the current map of the environment is used to select *target sectors* for the agent in order to explore.

The strategy adopted for exploration is to individuate *frontiers* [14]. Frontiers are those portion of the border of the current map that are not occupied by obstacles, and that therefore do not prevent movement. The agent must cross a frontier in order to move into unobserved areas of the map, explore the environment further, expand its map, and eventually find the goal. The frontier approach is combined with RL in order to reduce unprofitable exploration. The procedure is summarized in Algorithm 1.

Algorithm 1 Target sector selection

Given
  $\texttt{map}, \sigma \in \Sigma$
Initialize
  $V \leftarrow 0$
Until *goal not found*
  unexplored $\leftarrow \{\sigma : < 50\%\, observed\, squares\}$
  eligible $\leftarrow \{\sigma : adjacent\, to$ visited$\}\ \cap \{\neg$visited $\cup$ unexplored$\} \cap \{\sigma : \exists\, frontier\}$
  target $\leftarrow \text{argmax}_{\text{sect} \in \text{eligible}} V(\text{current}, \text{sect})$
  counter $\leftarrow 2 \cdot \text{distance}(\text{current}, \text{target})$
  Until *target is reached* $\vee$ counter $\neq 0$
    $navigate\, towards$ target
    counter $\leftarrow$ counter $- 1$
    $update$ map
  If $reached$
    $r \leftarrow -0.1$
    $update\, V(\Sigma, \text{target})$
  otherwise
    $r \leftarrow -1$
    $update\, V(\sigma_{tr}, \text{target})$
  $update\, \Sigma, V$

At the start of exploration, the agent has an initial map. The map is then partitioned into groups of 6-by-6 squares, the *sectors* $\sigma \in \Sigma$. If needed, additional unobserved squares are added to the boundaries of the map until all sectors are exactly 6 squares of side. Initialize the value function $V : \Sigma \times \Sigma \to \mathbb{R}$ as zeros.

All sectors that:

- are adjacent to a *visited* sector, i.e. a sector which contains at least a visited square, and
- are not visited, or contain more than 50% unobserved squares, and
- are separated by a *frontier* from at least one of their neighbours

are *eligible* target sectors. Frontiers are defined as follows: if the agent can reach the candidate sector starting from any side of one of the adjacent sectors, then a frontier exists between the two sectors. Unobserved squares are assumed empty for the sake of determining if a frontier exists. Figure 6 shows an example: neighbour sector **A** certainly does not have a frontier with the candidate, since the common side cannot be reached due to occupied squares; on the contrary, neighbour **B** has a frontier to the candidate.

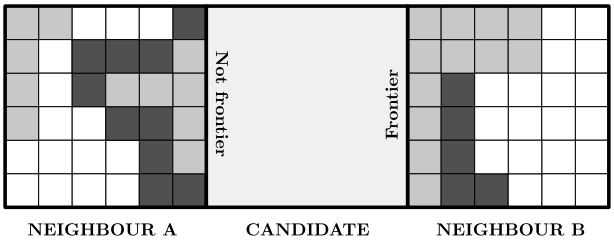

Fig. 6: An example of frontier evaluation.

The actual target among the eligible sectors is then selected according to value function V:

$$\texttt{trgt} = \text{argmax}_{\texttt{elig}} V(\texttt{curr}, \texttt{elig}). \quad (1)$$

If more than one sector has the same value, the nearest one is selected. The use of a value function to select actual targets is motivated by the fact that the agent might not be able to reach the target sector in a reasonable amount of time, e.g. because the unobserved squares turn out to be occupied, or because the agent needs to circumvent a wall in order to reach the target. Therefore, the agent is given a `counter` as a time limit to reach the assigned target. The `counter` starts equal to twice the distance, measured in sectors, between the agent and the target. Each time that the agent is assigned a direction of motion (as in figure 5), the `counter` is reduced by one. If the agent moves to the target before the counter reaches zero, it is assigned a reward $r{=}{-}0.1$. Then $\forall \sigma \in \Sigma$, V is updated according as

$$V(\sigma, \texttt{trgt}) = V(\sigma, \texttt{trgt}) + 0.2 \cdot (r - V(\sigma, \texttt{trgt})); \quad (2)$$

otherwise, $r{=}{-}1$ and, given σ_{tr} the sectors visited during the trajectory to `trgt`, it is

$$V(\sigma_{tr}, \texttt{trgt}) = V(\sigma_{tr}, \texttt{trgt}) + 0.2 \cdot (r - V(\sigma_{tr}, \texttt{trgt})). \quad (3)$$

Note that V is optimistically initialized since the reward is always negative. Therefore, when a target sector is reached, it is unlikely to be selected again due to (2) regardless of the current sector. Conversely, (3) penalizes the choice of target only for those sectors for which a path is not found. Since the map is continuously updated during navigation, the value function is expanded after each update to include any new sector discovered.

IV. Experimental setup

The approach is tested in a maze environment: the mentioned maze of Parr and Russell [6] (figure 7). The maze is a grid of 85 squares of side, of which approximately 3600 are visitable, while the rest are either occupied or not accessible.

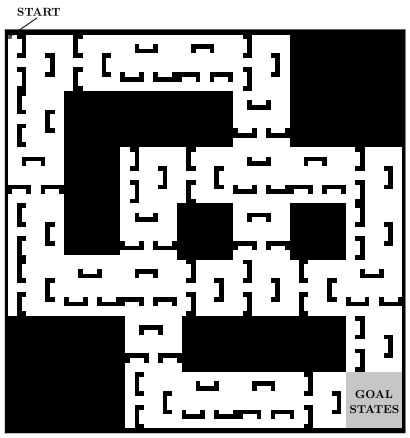

Fig. 7: The maze devised by Parr and Russell for HAM application (adapted from [6]).

The maze contains several identical obstacles which hinder exploration: they create multiple bottlenecks; also, if the agent enters within the "u" shape of an obstacle, it will need to spend actions to return in the open. If the agent collides with an obstacle, it will bump, so that its position remains the same but its orientation is flipped by 180°. The task of the agent is to find the goal states at the opposite side of the maze. Since the agent does not have any prior information on the maze, the task is equivalent to exploring the maze as fast and efficiently as possible.

The agent is initialized in the top-left corner of the map at a random cardinal orientation. The initial map is entirely composed of unobserved squares, except for the current sensor readings. In order to navigate, the agent uses the hierarchical control described in section III: table $\mathcal{T}$ is obtained by teaching the trainee agent in the minigrid environment for 120000 episodes. For the first 100 timesteps, the agent adopts the following stochastic policy: with probability 0.79, the agent advances in the direction of its current orientation; with probability 0.13 the agent turns 90° to its left; the rest of the times it turns 90° to its right. This policy provides the agent with a direction to explore, and is the result of a selection among similar candidate policies, evaluated in a simulated goal-finding task in multiple randomly cluttered environments. Albeit suboptimal, it was semi-empirically devised to provide an incentive to exploration compared to a canonical random policy.

After the 100^{th} timestep, the policy is modified to take into account the presence of the map. With probability 0.2, the agent follows the earlier semi-empirical policy; otherwise, a target sector is identified according to the frontier approach of the previous section. When a target sector is assigned, the agent turns to face the cardinal direction that will bring it closest to the target; e.g. if the target is located at North-North East, the agent would turn to face North. It will then advance in that direction until either its orientation is no longer the closest to target (in which case it will adjust its orientation and continue advancing), until it completes the path (updating value function V with a success, and returning to policy), or until it fails to reach the target before the counter runs out (in which case it updates V with a failure, and again returns to policy for a new target). The agent continues exploring until it comes in sensor range of any goal state, which terminates the episode.

V. Results

Figure 8 shows the amount of steps to goal and the number of observed squares for 30 sample episodes of the algorithm. Each curve represents a different episode and terminates with a square. The number of primitive actions to goal has a mean value of 1917 (indicated by a vertical line), but it varies from a few hundreds to more than five thousands. This is coherent with the experimental setup, since the agent does not know the location of the goal, but only tries to maximize its exploration. Therefore, when the agent explores in the goal direction by chance, the goal states can be found in as little as 620 timesteps. Conversely, if the agent explores away from the goal, completion time can increase by several times.

To provide a reference, the algorithm can be compared with two different hierarchical approaches to the same problem: the original HAMs of Parr [6], and the results of Zhou et al. [15]. Parr defines a-priori a hierarchy of machines, each representing a constraint on the space of possible policies, connected through `call`, `stop` and `choice` states. The agent considered by Parr has a different set of sensors: four short range "sonars" which detect obstacles in the adjacent square, and a long range, high-directed sensor that can spot obstacles far away. Another difference is that Parr's agent does not perform mapping. Under these conditions, the HAMQ-learning achieves an ideal performance after 270000 iterations.

In the work of Zhou et al., a flat Q-learning algorithm is compared to a hierarchical Q-Learning algorithm, using the same agent described in this paper. The higher level of hierarchy of this algorithm uses local sensor readings to estimate the current position of the agent with respect to an internal map, which is updated during and after each episode. This constitutes a *belief macro state*, upon which a `North`, `East`, etc. command is sent to the lower level controller, which navigates avoiding obstacles. The initial performance of Q-learning and Hierarchical Q-learning is approximately the same, with primitive actions in the order of 10^5 to find the goal, which reduces drastically to ~ 5000 primitives for the hierarchical algorithm after 30 episodes.

A direct comparison between the proposed algorithm and the two discussed above should be avoided, due to the intrinsic differences between the three approaches. Nonetheless, it should be noted that the proposed algorithm is able to achieve a comparable level of performance during its first episode, without prior information on the specific environment, but including off-line training. This proves the effectiveness of implementing abstraction together with pre-training as a tool for obtaining hierarchical structured controllers with a satisfactory level of initial performance.

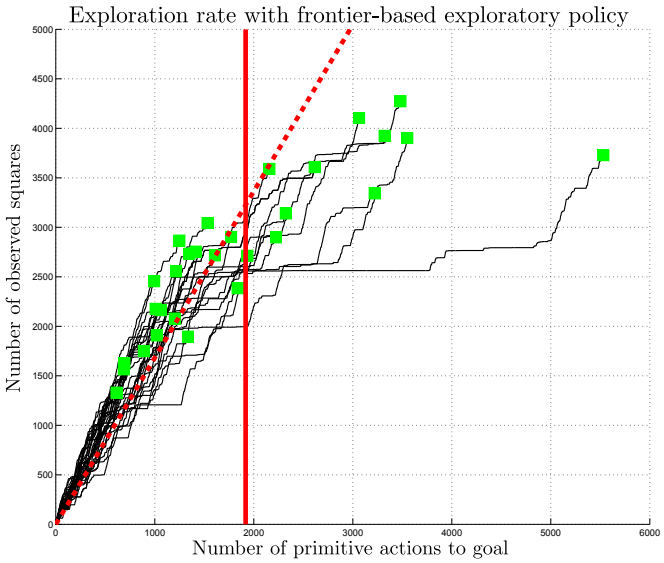

Fig. 8: The amount of exploration performed by the agent in the Parr and Russel maze with the frontier approach. Each line represents one of 30 different episodes, terminating with a square. The vertical line represents the mean number of primitive actions to goal. The dashed line indicates the mean exploration rate.

In the first 1000 steps, the exploration rate is very consistent, with a mean rate of approximately two new observed squares per timestep. In order to appreciate the result, consider that the maximum rate allowed by the sensor reading is of seven squares when advancing, and of three squares when turning, in the event of unobstructed, unobserved surroundings. After the initial exploration, a reduction in rate of exploration can be observed, for which there are two explanations. First, the agent often needs to relocate to another position of the map in order to continue the exploration, navigating through already explored areas of the maze as it does so. Second, the agent often attempts to explore the unobserved states on the other side of walls, treating them as obstacles. These attempts are quickly dismissed by implementing the value function; however they cause the agent to spend actions that don't contribute to the exploration. These explain the peculiar shape of the curves, in which steep rates of exploration are separated by period with little or no new observations. Even when accounting for these effect, the mean rate of exploration is 1.68 new squares per timestep (indicated by the dashed line).

In order to evaluate the exploratory strategy adopted by the agent, figure 9 shows the exploration rate when, instead of adopting the frontier approach, the initial stochastic, semi-empirical policy is adopted for the entire duration of the task. Once again 30 different episodes are presented. It can be seen that the mean amount of steps to completion is significantly higher, approximately 6400 steps, and that the mean exploration rate is also lower, equal to 0.62 new observed squares per primitive action.

Finally, figure 10 shows the final maze and the trajectory of the agent after a typical episode. The agent starts with 100

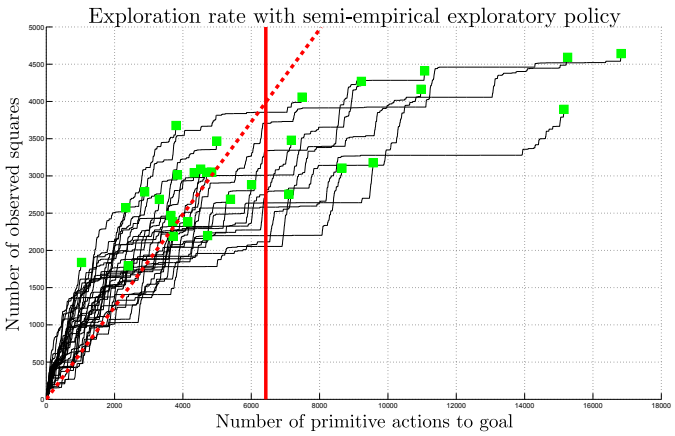

Fig. 9: The amount of exploration performed by the agent in the Parr and Russel maze with the semi-empirical policy.

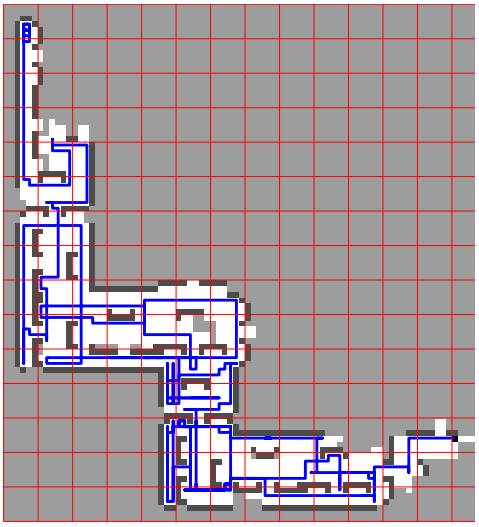

Fig. 10: The final map of the agent after one sample episode. The agent start at an extremity of the map, in the top-left corner, and follows the trajectory until it finds the goals by exploring efficiently its surroundings.

primitive actions suggested by the semi-empirical stochastic policy. As a result to this, the first "room" between the starting position and the first bottleneck is only partially explored. Conversely, all remaining areas through which the agent successively navigates are almost completely explored. The agent's trajectory covers most of the open areas, only occasionally repeating its steps. As an exception, the agent spends a few timesteps in the proximity of the wall at the bottom-left of the maze. As mentioned, this is due to the agent trying to access the area behind the wall itself, which causes a small delay in exploration.

As a concluding observation, the algorithm presented here is designed to maximize the starting exploration, preventing the

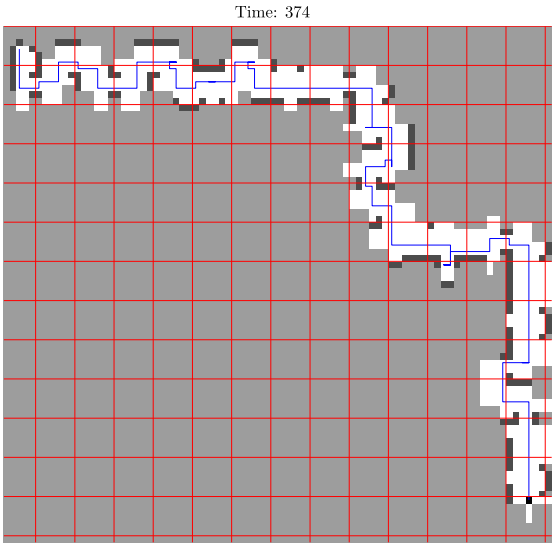

Fig. 11: A sample follow-up episode with the algorithm. Using the previously gained map and the discovered location of the goal, this can be reached with less than 400 primitive actions.

inefficiencies that RL and HRL tend to exhibit in the initial phase of learning. As such, the results presented are limited to first episodes, and no knowledge is transferred from one episode to another. Nonetheless the algorithm can also include previous knowledge to solve the task in a lower amount of iterations. Figure 11 shows an example of a follow-up episode of the algorithm, where a goal value function $V_g : \Sigma \rightarrow \mathbb{R}$ is built based on the map obtained in the previous episode. The value of sectors increases as they near the goal, so that the navigation policy of $\mathcal{T}$ can be used (with one more level of abstraction) to control the agent from the starting sector to the one containing the goals, requiring in this case less than 400 primitive actions. Therefore, the proposed strategy performs satisfactorily on its first episode; however, if previous knowledge is indeed available (e.g. a map), the strategy can easily exploit it to further improve its performance, similarly to other RL approaches.

VI. Conclusions

This paper presents a layered, hierarchical control approach. The strategy of the approach consists of developing a simple low-level control, learned off-line by a trainee agent in a simple, highly abstracted environment, without specific information on the task to be performed. The learned policy is then applied iteratively and on multiple levels of hierarchy, providing control for local navigation. The approach is completed by a global exploration strategy obtained through a combination of mapping and Reinforcement Learning. Mapping is used to individuate frontiers in order to discover unexplored areas, while Reinforcement Learning provides flexibility to the exploration, preventing the agent from repeating mistakes when looking for frontiers.

A maze environment is used to test the algorithm in simulation. The agent, representing a ground robot equipped with sensors, is tasked with finding a goal location. The maze, including the location of the goal and the shape and number of the obstacles, is initially unknown to the agent. The primary objective of the agent is then to explore it as efficiently as possible. Results show that the algorithm provides a steady rate of exploration during the initial phase of the task, followed, if the goal is yet to be found, by a second phase with slightly reduced exploration, due to relocation and due to delays from incorrect exploration attempts. Nonetheless, a comparison with similar hierarchical methods shows that the algorithm achieves a satisfactory performance in exploring the environment from the very first iteration. This is confirmed by examining the trajectories obtained during the episodes, which show the agent exploring the environment exhaustively but efficiently.

Summarising, the combination of abstract, off-line training with a map based, adaptive exploration results in a learning agent whose initial performance, albeit suboptimal, is not affected by the conventional "blind search" phase. Furthermore, the implementation of navigation policy through layered control prevents the agent from attempting unsafe actions, such as entering unobserved squares.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An introduction*, vol.1, no.1, Cambridge: MIT Press, 1998.

[2] L. P. Kaelbling et al., "Reinforcement Learning: A survey", *J. of Artificial Intell. Research*, vol.4, pp.237-285, 1996.

[3] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning", *Discrete-Event Dynamic Syst.*, vol.13, num.4, pp.341-379, 2003.

[4] R. S. Sutton and A. G. Barto, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning", *Artificial Intell.*, vol.112, num.1, pp.181-211, 1999.

[5] D. Precup, "Temporal Abstraction in Reinforcement Learning", Ph.D. dissertation, University of Massachusetts, Amherst, 2000.

[6] R. Parr and S. Russell, "Reinforcement learning with hierarchies of machines", *Advances in Neural Inform. Process. Syst.*, pp.1043-1049, 1998.

[7] T. G. Dietterich, "Hierarchical reinforcement learning with the MAXQ value function decomposition", *J. of Artificial Intell. Research*, vol.13, pp.227-303, 2000.

[8] C. Chen et al.,"Hybrid Control for Robot Navigation - A Hierarchical Q-Learning Algorithm", *IEEE Robot. Automat. Mag.*, vol.15, num.2, pp.37-47, 2008.

[9] D. Barzin and G. Nejat, "A hierarchical reinforcement learning based control architecture for semi-autonomous rescue robots in cluttered environments", in *2010 IEEE Int. Conf. Automat. Sci. Eng.*, pp.948-953, Aug. 2010.

[10] R. Brooks, "A robust layered control system for a mobile robot", *IEEE J. Robot. Automat.*, vol.2, num.1, pp.14-23, 1986.

[11] P. Dayan and G. E. Hinton, "Feudal Reinforcement Learning", *Advances in Neural Inform. Process. Syst.*, 1993.

[12] H. P. Moravec and A. Elfes, "High Resolution Maps from Wide Angle Sonar", in *Proc. of the 1985 IEEE Int. Conf. Robot. and Automat.*, 1985, vol.2, pp.116-121.

[13] S. Thrun and A. Bücken, "Integrating Grid-Based and Topological Maps for Mobile Robot Navigation", in *Proc. of the AAAI* 13^{th} *Nat. Conf. Artificial Intell.*, Portland, OR, Aug. 1996, pp.944-951.

[14] B. Yamauchi, "A Frontier-Based Approach for Autonomous Exploration", in *Proc. of the IEEE Int. Symp. Computational Intell. Robotics and Automation (CIRA)*, 1997, pp.146-151.

[15] Y. Zhou et al., "Autonomous Navigation in Partially Observable Environments ", to be presented at the *Int. Micro Air Vehicle (IMAV) Conf. Competition*, Beijing, China, Oct. 2016.