Unknown Area Exploration with an Autonomous Robot using Markov Decision Processes

Simon Le Gloannec, Laurent Jeanpierre, Abdel-Illah Mouaddib

Unknown Area Exploration with an Autonomous Robot using Markov Decision Processes

Simon Le Gloannec and Laurent Jeanpierre and Abdel-Illah Mouaddib

***Abstract*— This paper addresses the problem of exploration of an unknown area by an autonomous robot. The robot decides at each step where it has to move, without any human intervention. The Goal is to gather the maximum information in a minimum time. This work is based on the Markov Decision Processes framework. We focus on the decision problem : the robot must automatically select points where it can maximize its information gain. Even if the goal of the robot is to produce a map of the environment, the localization aspect is not treated in this paper. We divide the environment into three layers. One Layer where the robot moves, another where the map is constructed and a last layer where decisions are taken. As this model is suitable for small robots with cheap sensors we present some experiments on real robots, with very good results.**

I. Introduction

We are considering an exploration mission with an autonomous robot in an indoor unknown area. The robot decides at each step where it has to move, without any human intervention. The main objective of exploration is to maximize the knowledge about the environment. The robot has to move towards locations that maximize the information gain. Controlling a robot for exploration consists of defining exploration actions.

Actually, one cannot plan a static and deterministic path from the actual position to another location, because one cannot predict exactly the next position. Actions are not deterministic and thus we have to account for uncertain outcomes. For example, a robot may slip or stall during an action, leading to a position longer or shorter than expected. Furthermore, it is not always feasible to get the exact position for the robot. The Markov Decision Processes' (MDP) framework produces a policy that tells the right action to use from any situation in order to maximize the exploration gain while considering the uncertainty of the actions.

The exploration model is fully subsumed by the (PO)MDP (Partially Observable MDP) framework and applied in many domains such as planetary exploration [1], search and rescue[2] and abandonned mine mapping [3], [4]. However, exploring using POMDPs meets the curse of dimensionality and limits its application to exploration domains because it considers uncertainty on observations, on outcomes of actions and on its own state. Recent research has focused on algorithms that scale up. The most popular algorithm is QMDP, which transforms a POMDP into an MDP of beliefs which has the same complexity as an MDP [5].

{slegloan,laurent,mouaddib}@info.unicaen.fr
All authors are with GREYC, Université de Caen Basse-Normandie, UMR 6072, 14000 Caen, France

In POMDPs, beliefs represent the distribution probability on states and their set is huge. To reduce this set, particle filter versions of the POMDP algorithms approximate beliefs by a finite set of states. This class of techniques is named Monte-Carlo POMDP [6]. Other techniques based on specific structure or augmented MDP allow to reduce the complexity of POMDP. Recently, a new algorithm named TOP [7], which is one of the most competitive algorithms, based on topological organization of the space to better organize the resolution, has been developped.

However, even state-of-the-art algorithms like TOP are not able to cope with the complexity of realtime exploration because of the combinatorial induced by the unknown environment. Therefore, in this work, we will consider that the precise location of the robot is known thanks to any possible way: Odometry, GPS, Scan-matching, ... Then, a simple MDP is able to compute a relevant policy to choose exploration actions. If the localization module is able to produce a belief instead of a single position, a QMDP step can use this information along with the MDP values to provide an even better choice.

The approach we present can be extended to multiple robots that explore the area collaboratively. The use of several robots in this context shows several benefits such as speeding up the exploration and improving the robustness and the fault tolerance of the team. MDPs and POMDPs have been extended to Decentralised MDPs (Dec-MDP) and Decentralized Partially Observable Markov Decision Processes (Dec-POMDP). Dec-(PO)MDPs are a smart way to address this kind of problems using multiple robots but it has an even higher complexity for computing a solution[8]. This typically limits the scalability in terms of number of agents and of the size of the area to explore.

However, new coordination mechanisms for several robots have been developed recently using MDPs instead of Dec-MDPs. One of these models, Vector Valued Decentralized Markov Decision Process (2V Dec-MDP) [9], is based on a two steps mechanism: the first one localizes areas with a high information gain, while the second computes the coordination with local observations and interactions limited to these areas. In another model, Opportunity-Cost Dec-MDP (OC-Dec-MDP) [10], the coordination is based on the constraints between task executions, these tasks being spread among lots of agents by the algorithm. These two approaches are very interesting because we can compute solutions for larger problems while keeping the Dec-MDP formalism.

In this paper, we work on a partially known environment, in which a robot that knows its exact location has to decide

where to go to gather as much information as possible as fast as possible. Therefore, the policy must be computed frequently on-board with limited resources. Each time the robot aquires new data, it has to find new goals and compute a policy to reach them. Thus, the policy computation has to be quick to avoid long pauses and wasting time while waiting for the policy to be ready for use.

The outline is the following. We first present the model and the algorithm the robot uses to explore its environement with almost no knowledge of its stucture. This algorithm consists in three steps: the data acquisition, the search for new goals and finally the computation of the policy and its execution. In section V, we show the experimental conditions of the exploration and the results that we obtained with a single robot. Finally, we present in section VI the model extension to multiple agents.

II. Context and target application

This work is a part of the French national challenge *"Cartographie par un Robot d'un Territoire"* for exploring and mapping an unknown area (CAROTTE), funded by the *"Agence Nationale de la Recherche"* and the *"Direction Générale de l'Armement"*. In this project, a robot or multiple robots must produce a map of the environment and must detect instances of objects like chairs, books, furnitures, boxes, etc. The mission lasts thirty minutes at most; during this time robots work autonomously to map an area included in a 20 meters width square and to return to its starting position. Several competitors will participate to this challenge where the goal is to maximize the covered space, the precision of the map and the number of detected objects.

The six wheels robot we use is equiped with a Led Range Finder that detects obstacles in the environment as a vector of distance measurements. This kind of robot has several advantages as far as exploring is concerned. Its main benefit is its moving pattern. It can go forward or backward, rotate without moving or even rotate while moving forward or backward. Additionally, it can climb over small obstacles or cross over small holes in the floor. However, its main drawbacks are a quite high energy consumption and classically poor odometry.

The environment is static: several boxes, objects and furnitures encumber the rooms to explore but do not move during the experiment. The objective for the robot is to produce a map of its environment, with as many details as possible. In this article, we will focus more precisely on the autonomous decision making and on the exploration strategy rather than on the localisation. This part is left for future work. Although the challenge includes taking pictures and locating objects, we will not address this task in our current presentation.

III. Background on Markov Decision Processes

An **MDP** [11] is a fully observable Markov Decision Process represented by a tuple $\langle S, A, T, R\rangle$ where: S is the finite set of states that represent the environment for an agent; A is the finite set of possible agent actions; $T : S \times A \rightarrow \prod(S)$ is a state transition probability distribution, $T(s, a, s') = Pr(s_t = s' | s_{t-1} = s, a_{t-1} = a)$ is the probability of transitioning from state s to state s' after doing action a; $R : S \times A \times S \rightarrow \Re$ is the reward function mapping $S \times A \times S$ to a real number that represent the agent's immediate reward for making action a while being in state s. To solve an MDP, we calculate a policy π that maps each actual state of the system to the optimal action that maximizes the long-term expected reward $\pi_{MDP} : s_t \rightarrow a$.

IV. Our model and exploration algorithm

In this part, we explain how our model is built upon the MDPs framework to quickly build an exploration policy.

The main algorithm for exploration is an infinite loop which consists of:

- acquiring data from the environment, typically with a Laser Range Finder sensor,
- finding interesting locations to explore,
- computing a policy to reach one (or more) of these points,
- executing the actual policy.

Each time the robot acquires new data, it has to find new points of interest and it must compute a new policy to reach them. These points are defined as poses, which are oriented locations where it is best to gather new information. This technique is classically named "Next Best View" (NBV). We based the policy computation on Markov Decision Processes (MDP), which are renowned for their ability to cope with uncertainty. Additionnally, very efficient algorithms based on Dynamic Programming are available to solve them quickly.

Our representation is based on an occupancy grid. More precisely, we divide the model in three layers (see Figure 1). The real environment is represented in the first layer. It mainly includes the interface with the physical robot. The map is stored in the pixel layer, which is a human-readable picture. This intermediate layer is both close to the real world and to the decision layer. The MDP computation occurs in the third layer that we have divided into hexagons. Since hexagons have six natural orientations, the robot has a smoother behaviour while applying the movement actions in our model. Moreover, the transition matrix, which models the effects of the robot's actions, is easier to define.

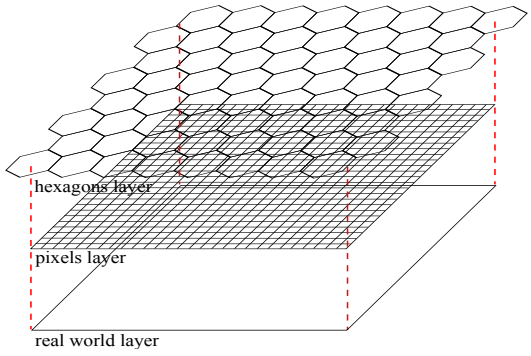

Fig. 1. Hexagonal occupancy grid. The real world features are projected simultaneously onto a pixel map and onto a hexagonal decision map.

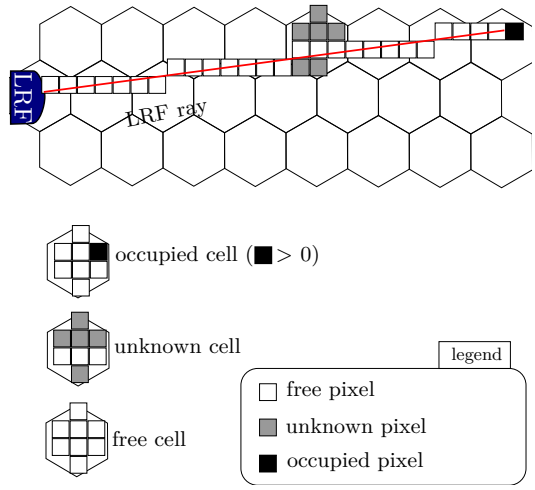

Fig. 2. Occupancy update with Bresenham's algorithm. Pixels update from a single laser ray.

A. Data acquisition

Data are acquired with a ten meters Led Range Finder sensor (LRF). This cheap version of a Laser Range Finder measures 102 distances, every 0.7 seconds approximately. Data come from the real environment and are stored into the pixel layer according to Figure 1. Since we base our approach on the occupancy grid technique, each pixel has a value $\phi \in [0..255]$ with a starting value equal to 128: fully unknown. The LRF throws rays that may hit features in the environment. Therefore, the robot receives a measured distance for each ray cast. The pixels related to a particular ray are updated with the help of Bresenham's algorithm [12]. Figure 2 describes this operation. Each time a ray passes through a pixel, its value increase up to 255: totally free. Each time a feature is hit in the environment, an obstacle is detected. On the pixel layer, the associated pixels' value decrease down to 0: totally occupied. Any pixel will be considered as occupied if its value is lower than a Threshold T_o. On the contrary, it will be considered as free if the value is higher than the threshold T_f. Other pixels are considered as unknown ($T_f < \phi < T_o$).

While the data acquisition process updates the pixel layer, it also updates the hexagon layer as soon as a given pixel crosses one of the thresholds. This handles the uncertainty of the perception by delaying the update of the decision layer until the new information are confirmed.

The robot uses the hexagon layer to make decisions (see IV-C). An hexagon is composed of a set of pixels. It will be considered as occupied if at least one of its pixels is occupied. This represents the obvious risk for colliding with objects. If the ratio of unknown pixels in a given hexagon is less than a threshold T_r, the hexagon cell will be considered as free. Other cells will be considered as unknown.

The LRF data acquisition is done repeatedly, almost 1.5 times per second. When the hexagon layer has been updated, the algorithm has to compute the next interesting points to visit based on those new data.

B. Searching for interesting points

The robot has to explore the whole environment. Each time it receives new data, it has to look for new interesting destinations. The best view pose is a location that is close enough and an orientation that allows gathering lots of new information about the environment.

In this part, we work on the hexagon layer which is used to make decisions. We forbide the robot to move directly to unknown cells, because it may be dangerous. We also forbid the robot to hit the walls, because this is clearly dangerous. However, it is possible to go through free cells.

The frontier between unknown cells and free cells are considered as interesting, because looking at unknown cells well probably provide the robot with new information. However, it is not necessary to reach such a cell to obtain the information. It is generally sufficient to look straight at it. Additionally, unknown cells neighbouring with obstacle cells would not be interesting because the robot cannot move to it without collision. Therefore, we put some positive reward values on free cells that are in the unknown cells neighbourhood. This way, the robot comes into free cells that will provide new information when facing the right direction.

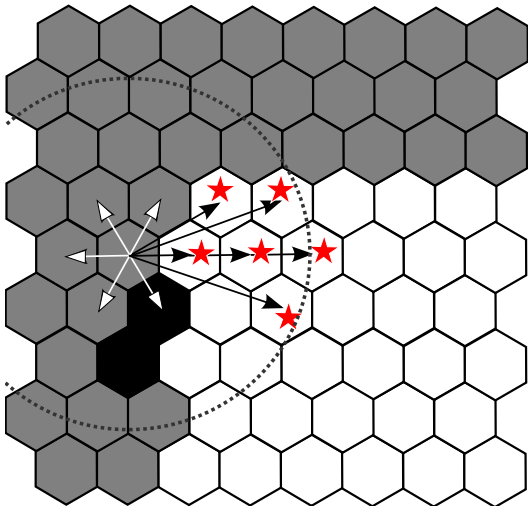

Fig. 3. Propagating the reward into the environment. Here, an unknown cell propagates its reward over a radius. White arrows show impossible propagations whereas black ones represent active propagations. Stars are the resulting rewards.

Figure 3 shows the reward propagation mechanism we implemented. For any unknown cell the reward is propagated in a neighbourhood with a given radius (the grey dotted circle int he figure). Starting from a cell, the algorithm propagates rewards if the line of sight is free (black arrows) and it stops the propagation if it encounters an occupied or an unknown cell (white arrows). The reward value of each free cell is increased each time we add a reward to it. With this method, free cells in the neighbourhood of unknown cells have a high amount of reward and free cells in front of walls does not have reward. This update is done all over the map, but it could be narrowed to the immediate neighbouring of the robot.

We can note that rewards will never be gained by the robot. In fact, as it comes close enough, its LRF will gather new information and unknown cells will hopefully become known before they are reached. Therefore, the rewards will disappear before the robot can claim them. The side effect of this is that the orientation of cells is not necessary as far as rewards are concerned. Formally :

$$R(s) = R(x_H, y_H, \theta) = R(x_H, y_H)$$
$$= \begin{cases} r > 0 & \text{if } (x_H, y_H) \text{ is free} \\ & \& \ (x_H, y_H) \text{ is near an unknonw cell} \\ 0 & \text{otherwise} \end{cases}$$

The fact that all unknown cells are simultaneous sources of reward removes the obligation of choosing Best View Points (BVPs), and then computing a path to reach all of them before choosing the most interesting one. The Value Iteration algorithm, which is described in the next section, computes simultaneously the paths to all the frontier cells, along with their costs. As a matter of fact, it even chooses which is the most interesting BVP for all the known positions and orientations.

Once we have computed the rewards for all the cells, the robot is able to compute its policy, so that it will be able to act optimally in the discovered environment.

C. Computing the robot's policy

The decision making process is based on the Markov Decision Processes (MDP) framework. Here, the MDP is a tuple $\{S, A, T, R\}$ where :

- the states space S represents all possible poses (oriented locations) for the robot,
- the actions set A contains available actions (described in Figure 4),
- the transition function $T : S \times A \times S \rightarrow [0, 1]$ describes the probability to be in the state s' after having executed action a in state s,
- the reward function $R : S \rightarrow \mathbb{R}$ indicates the value the robot gains in each possible state.

The policy is computed with the Value Iteration algorithm [13], that applies Dynamic Programming to solving MDPs efficiently. A policy is a mapping from S to A; it indicates the best action for the robot in each possible state. This policy maximizes the value function, which is the solution of the Bellman equation system.

When the policy is calculated, the robot executes the action that correspond to its oriented position (state). Then it reads the new position, and performs the corresponding action until a new policy is ready.

1) The state space: The state represents the robots pose. It includes the coordinates of the robot center and its orientation in the hexagonal grid: $S = \{(x_H, y_H, n), n \in [0..5]\}$.

The choice of the hexagon size really matters in this application. If we chose too small hexagons, we would have a very large state space. This would have increased the computing time since the complexity of Value Iteration is $S^2 \cdot A$. On the other hand, if we chose too large hexagons, they would cover a lot of real space. Since we consider that the robot cannot cross a cell if there is at least one pixel occupied, it would severly reduce movement possibilities. For example, with a 50 centimeters hexagon configuration, it would happen that hexagons overlap an open door on both sides. In this case, an open door would be seen as a wall. For these reasons, we choose 15cm large hexagons.

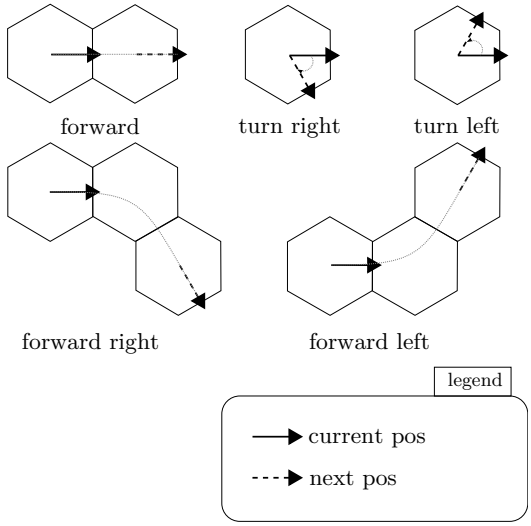

Fig. 4. The robot actions. Solid lines represent current pose while dashed lines figure the pose resulting from the actions.

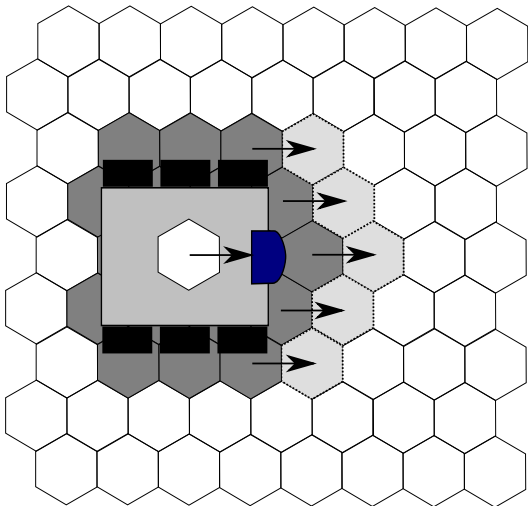

Fig. 5. The robot mask. Dark cells show the robot's current position. Gray cells show the required free cells for the "forward" action to succeed.

2) The 5 actions: The 5 actions we have implemented are described in Figure 4. They model the movement of the robot's center.

3) The transition function: An action's outcome depends on the cells that the robot will cross during it's movement. Since the robot can be so large that its body overlaps several cells, we have to test if any of the robot's mask cell will hit an occupied or unknown cell during its movement to decide if the action is likely to succeed or not. The robot we use covers 18 hexagons (see Figure 5).

We introduce an action mask for each of the actions. As an example, the mask for the "forward" action is depicted in the light grey cells of Figure 5. If there is at least one occupied or unknown cell into the action mask, we consider that the action will fail: the robots stays in place. Otherwise the robot goes to its intended new position and orientation. Considering all these assumptions, we obtain a factored transition function representation with a sparse transition matrix.

This function is updated as soon as the map is modified due to the robot exploration. When new cells are uncovered as free, relevant transitions are updated to allow the robot to go through. When new obstacles are discovered, relevant transitions are updated so that the robot avoids them and chooses an alternative path.

Since the transition matrix is really sparse, it is straightforward to reduce the complexity of Value Iteration from $S^2 \cdot A$ down to $B \cdot S \cdot A$, B being the maximum branching factor, that is the maximum number of states reachable from a single start state.

This brings up another possible enhancement. If we suppose that the transition function is deterministic, we can drastically reduce the complexity of the algorithm since $B = 1$. However, it must be understood that this may degrade the policy since the robot would not consider anymore that its actions may fail. One possible way to overcome this limitation would be to increase the actions' mask to provide more safety. However, this increases the complexity of computing the transition function, and does not account for slipping or drifting, for example.

4) The reward function: The reward function is computed according to the section IV-B. This reward is negative when the robot encounters walls, and high rewards are located around frontier cells, between free and unknown cells.

Since the robot has to return at its starting point, we added a reward to the initial position. This reward grows exponentially with time so that it become more interesting as the deadline approaches. this prevents the robot from exploring a new path to the end point. However, this allows for a simpler scheme, with exploration as the only goal until it decides leaving the area. This avoids tuning the usual exploration/exploitation problem stated in [14]

5) Choosing the horizon: The last two parameters of any MDP are the discount factor, and the horizon. The discount factor γ is a weighting factor that reduces the value of future events. The horizon H is the number of actions that are considered when making the decision.

In our model, the discount factor has little influence on the model. It weights the attractiveness of far rewards with respect to close ones. However, if gamma is set to a value too close to 0, the robot may not be able to reach rewards if they are too far.

The main parameter of interest is the horizon. When the policy is computed with Value Iteration, H actions are considered before making a decision. Any reward that is farther will not be considered. This is not a big problem since new rewards are generated when the robot uncovers fresh unknown areas. Actually, the frontier between known and unknown areas, which is the source of the rewards, goes back as the robot moves toward it.

In general, there are lots of rewards all around the robot because it goes toward them. When the robot moves closer of the frontier, new unkwown areas are uncovered, and the state space has to be updated. Therefore, a new policy will be computed after a few actions. Anyway, any increase of H also increases the computing time because another iteration must be computed for each new action. Thus, H should be kept as low as possible, so that policies may be computed more quickly.

However, when the robot has reached a dead end, for example a corner of the room, the frontier goes behind the walls. At this time, it does not generate new rewards anymore. Then, there are two distinct situations. If there are some reachable rewards around the robot, the new policy will automatically bring the robot to them. If there is none, the robot may be trapped in a area where no action if better than the others.

To account for this situation we dynamically change the horizon H depending on the value function. Let's remember the value function is the discounted sum of all the rewards the robot expects to gain when following the policy. Therefore, if the value function measured at the current location of the robot is below some threshold, H is increased so that rewards generated earlier become reachable. At this time, a lengthy policy computation occurs and the robot obtains a policy that allows it for exploring unknown distant areas.

When the robot finally comes back to an area where rewards are close enough, H can be brought back to a lower value to save computation time and to increase reactivity while computing short-term policies more frequently.

V. Experiments

During the challenge, the robot must explore an area of 120 m^2 in less than 30 minutes. It must also take pictures of the environment to detect some objects. We use is a Koala robot equiped with a Led Range Finder PBS-03 from Hokuyo Automatic LTD. The LRF spans a measure of the environment every 1.8 degrees. Each measure ranges up to 10 meters but experiments show it is only reliable under 5 meters. The room in which the robot evolved in this experiment was partially closed. Some doors (on the left of the map and also in the picture) were partially closed so that the robot cannot cross through them. The robot can still detect some free space in these rooms but cannot reach them. Some boxes had been scattered in the environement to make walls, objects and such other things (see Figure 11).

We instanciate our three-layer architecture as follows: The first layer is a 20×20 m square. The robot moves autonomously in this square. The second layer contains a 1024×1024 pixels map. Each pixel represent 2×2 cm square in the real world. This map is one of the challenge expected result. The third layer contains 183×156 hexagons. Each hexagon is 15 cm width (between two opposite edges) and contains approximately 40 pixels.

We obtain the map depicted in Figure 6. The grey area represents the unknown space. Walls, boxes and other obstacles are painted with black dots while free space is filled with white. On the left of the map, the robot has seen some free space, but it is unable to reach it because doors are not opened large enough for a robot.

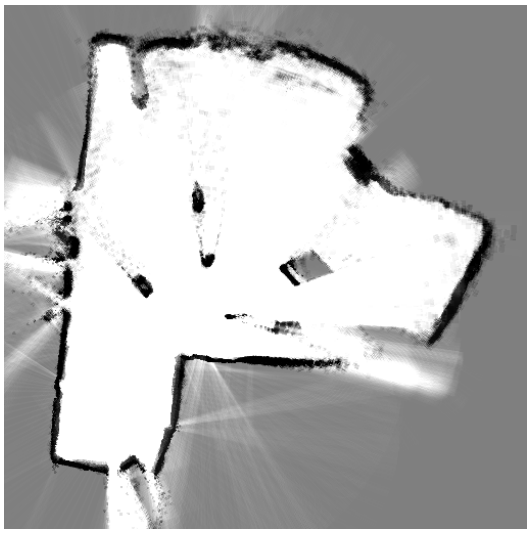

Fig. 6. Resulting map. Pixels' colour ranges from Black (obstacle) to White (free). Gray shades represent the uncertainty.

Fig. 7. Decision layer. Each hexagon is labelled (coloured) as Free (White), Unknown (Gray) or Occupied (Black).

In the lower-right corner of the map, there is a fontier between unknown cells and free cells that is still reachable. As a result, the reward function in this area is high. Rewards are depicted in Figure 8. White hexagons correspond to negative rewards, black hexagons represent zero reward cells and grey hexagons are cells with positive rewards. The robot has computed a new policy with these data. The corresponding value function is depicted in Figure 9. This is a far-reward situation with $H > 100$. We can see that

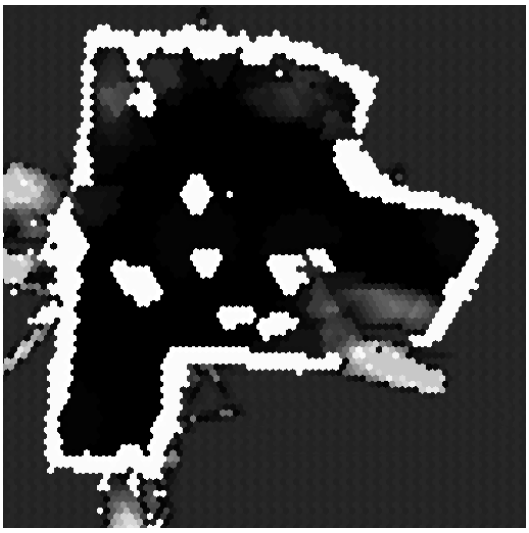

Fig. 8. Reward function. Each hexagon is coloured by the reward the robot gains for being there. Pure White is really bad (walls), Black is neutral, Grays are positive rewards (the brighter the greater).

Fig. 9. Value function (1 angle out of 6). Each pose is coloured by the value the robot would gain following the optimal policy starting from there. Brighter shades represent greater values.

the expected value becomes greater and greater as the robot comes closer the the unknown area. This effectively brings the robot to this area so that it gets explored.

In this experiment, the explored area is approximately six meters large and seven meters long. It represents approximately 100000 pixels. We measure the time that is necessary to discover the whole area. The results are depicted in Figure 10, it represents the number of explored pixels in the map versus time. Each time step is during $1.8s$. We have made several experiments in this room, that show similar results. In the two runs shown here, we separate free pixels from occupied pixels. We observe that the robot has explored the whole area in less than 10 minutes (300-350 time steps). After 10 minutes no new areas are discovered except in the second experiment : the robot unfortunatelely gather some

"new information" through the windows and half-opened doors. This information is not accounted for, since it does not belong to the room to explore.

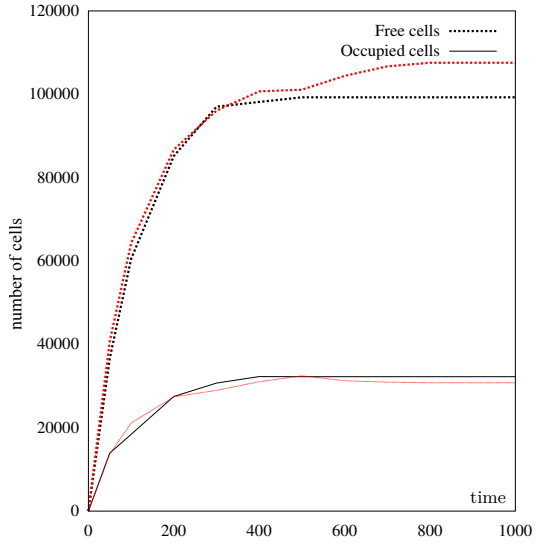

Fig. 10. The number of cells discovered during exploration

Fig. 11. Picture of the koala robot when the exploration starts

VI. Future work

The exploration mechanism currently works with a cheap odometric system. In this paper, we didn't focus on the localisation aspect but more on the autonomous decision aspect. The localisation can be improved using a Simultaneous Localization And Mapping module, before integrating new data into the pixel layer. With a SLAM module, the robot will be able to explore larger areas with less positional drift. We could also consider using active localization [15], that is choosing actions so that the localization can be improved.

We propose an extension to multiple robots. Multiple robots exploration is more robust. It also permits to distribute the exploration task among the robots. Thus, we expect a quicker exploration. We have to decentralize the decision. We think about a model where any robot shares its position and its map from time to time. This model can be view as a mixture between QMDP and Dec-MPDs.

VII. Conclusion

In this paper, we adress the problem of exploring an unknown indoor area by an autonomous robot. The robot must gather information with its Led Range Finder and produces a map of the environment. We are facing the problem of autonomous decision: the robot must maximize the information gain by moving to interesting points in the environment. We use Markov Decision Processes, that are renowed for planning and accounting for uncertainty outcomes. Our Model is divided in three layers: the physical layer, the acquisition layer and the decision layer. We don't focus on the localization aspect but more on the decision aspect. We experimented with real robots to proove the feasability and obtained good results: Even using only its odometric sensor, the robot actualy maps its environment within a few minutes, with no prior knowledge of the environment but its size. Now, we want to improve the localization system and extend the approach to multiple robot with a decentralized approach.

References

[1] E. Gat, R. Desai, R. Ivlev, J. Loch, and D. Miller, "Behavior control for robotic exploration of planetary surfaces," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 4, pp. 490 – 503.

[2] R. Murphy, "Human-robot interaction in rescue robotics," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews,*, vol. 34, no. 2, pp. 138 – 153.

[3] S. Mahadevan and N. Khaleeli, "Robust mobile robot navigation using partially-observable semi-markov decision processes," 1999.

[4] S. Thrun, S. Thayer, W. Whittaker, C. Baker, W. Burgard, D. Ferguson, D. Hähnel, M. Montemerlo, A. Morris, Z. Omohundro, C. Reverte, and W. Whittaker, "Autonomous exploration and mapping of abandoned mines," *IEEE Robotics and Automation Magazine*.

[5] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling, "Learning policies for partially observable environments: scaling up," *Readings in agents*, pp. 495–503, 1998.

[6] S. Thrun, W. Burgard, and D. Fox, "A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. San Francisco, CA: IEEE, 2000.

[7] J. S. Dibangoye, G. Shani, B. Chaib-Draa, and A.-I. Mouaddib, "Topological order planner for pomdps," in *IJCAI'09: Proceedings of the 21st international jont conference on Artifical intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2009, pp. 1684–1689.

[8] D. S. Bernstein, S. Zilberstein, and N. Immerman, "The complexity of decentralized control of Markov decision processes," in *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, Stanford, California, 2000, pp. 32–37. [Online]. Available: http://rbr.cs.umass.edu/shlomo/papers/BZIuai00.html

[9] A.-I. Mouaddib, M. Boussard, and M. Bouzid, "Towards a framework for multi-objective multi-agent planning," in *Autonomous Agent and Multi Agent Systems (AAMAS)*, 2007.

[10] A. Beynier and A.-I. Mouaddib, "An iterative algorithm for solving constrained decentralized markov decision processes," in *the twenty-first National Conference on Artificial intelligence (AAAI)*, 2006.

[11] M. L. Puterman, *Markov decision processes*. John Wiley and Sons, INC, 1994.

[12] J. E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems Journal*, vol. 4, no. 1, pp. 25–30.

[13] R. Bellman, "Dynamic programming : Markov decision process," *Princeton University Press, Princeton N.J.*, 1957.

[14] S. Thrun, "The role of exploration in learning control," in *Handbook for Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, D. White and D. Sofge, Eds. Florence, Kentucky 41022: Van Nostrand Reinhold, 1992.

[15] D. Fox, W. Burgard, and S. Thrun, "Active markov localization for mobile robots," *Robotics and Autonomous Systems*, vol. 25, pp. 195–207, 1998.