Abstract.

1 Context

Autonomous robots are involved in more and more complex tasks, among which exploration is a must-have skill. Exploration is indeed a regular common component of map building or search and rescue missions, in which the environment is only partially known prior to the mission, and must then be discovered and built online. The exploration problem consists in iteratively finding a position to explore and going to this position, while updating its knowledge about the environment. Exploration strategies have been intensively studied in a mono-robot or multi-robot context. Iterative positions are selected either among position candidates generated at random within a local safe zone around the robot [15, 3], or more classically on the frontier of the area known so far [17, 9, 7, 1, 16, 13]. In order to select the most appropriate position among the candidates, all the proposed algorithms [17, 9, 7, 1, 3, 16, 13, 6, 15] rely on the Next-Best-View (NBV) principle: the next candidates are ranked and the best one is selected as the next exploration position. This ranking uses several criteria, such as the amount of information that could be sensed at the candidate position, the length of the path to join it, or the communication availability in case a communication must be performed when sensing an object of interest [3]. These criteria are then, most of the time, aggregated within a global utility function u. For instance, [1] proposes a linear combination (1) of $A(q)$, the area visible from the position candidate q, and $L(q)$ the length of the path to join q from the current position, using a factor λ to weight the relative importance of a criterion over the other.

$$u(q) = A(q) - \lambda L(q) \tag{1}$$

A different combination of A and L is proposed in [9] (2).

$$u(q) = A(q) \exp(-\lambda L(q)) \tag{2}$$

Some authors have however opposed to these ad-hoc aggregation functions, by using a Multi-Criteria Decision Making (MCDM) approach, in which the criteria are aggregated based on a sound and scalable method: [3] uses a Choquet integral to have a fuzzy representation of the several criteria, before aggregating them into a unique utility function; [16] uses the PROMETHEE method to compare candidates two by two based on each criterion, and then build a ranking of candidates from these comparisons.

However, all these exploration algorithms only rely on the NBV principle: they select the next exploration candidate by only looking one step further. At first glance, such an approach could be reasonable in an environment which is mostly unknown, and where it is difficult to reason about the future. Nevertheless we have some other information, especially on the sensor and map evolution models. More precisely, while the model of the environment is unknown, we claim that it makes sense to use at very least some expectation of the information gain and estimation of future costs on the long term using automated task planning techniques; and hopefully obtain better exploration strategies than ones based on a more reactive behavior. A kind-of reasonning on future actions has been proposed in [13], where a complete plan is computed among the candidates by solving a TSP, then selecting the first node of this plan as the next exploration position. This work has been extended to multi-robot exploration [6], and then compared to other exploration strategies [7], with the conclusion that evaluating a complete plan leads to better exploration performances. However this approach only considers one criterion (the distance to frontier points), and evaluates a plan on the frontier known so far, without estimating future costs based on future observations. Another approach is presented in [14] (and extended in [17]). It uses probabilistic modelling of expectation of rewards and effects of actions as its core, but objective function is also a simple aggregation of observed area and travel distance.

Planning in unknown environment

In order to use automated planning techniques in unknown environment, several considerations have to be made. First, we want to anticipate as much as possible, but we also need to be able to integrate new data and replan easily, which advocates for a lightweight planning scheme. Another important aspect is that the state space may be very large and not known in advance, which makes mandatory to work with an implicit state space that can be explicited (or sampled) easily during the construction of the planning search tree. Finally, as we are in the context of autonomous robotics, anytime methods are preferred, in order to make the whole algorithm easily embeddable.

Our choice is to use a general approach that encompasses several algorithms from the well-known UCT [12] in the domain of deterministic decision making to algorithms like RTDP [2] originally designed for Markov Decision Processes. Such approaches have been unified as Monte-Carlo Tree Search [4] in a first step, then extended as Trial-Based Heuristic Tree Search [11]. We will refer to this family of algorithms as trial-based planning.

Such approaches behave well in uncertain environments (at least with probabilistic models of the result of actions), have a natural anytime behavior and integrates sampling mechanisms in their core (as opposed to global optimization over the whole state space). In a nutshell, they extend a search tree in state space starting from current state, exploring possible futures using both the model of the system dynamics and some heuristic that focuses search on more probable and/or more interesting futures. After some exploration is done (generally a trial from start state to some terminal state or until a given horizon is reached), information gathered is backuped upon the search structure from the final state of the trial until the present state.

In our case, there still is a fundamental difficulty to consider: such algorithms generally consider a reliable model of the system dynamics, as well as an accurate model of the reward and costs. In the case of autonomous exploration of unknown environment, the reward and cost models have to aggregate several criteria, among them at least cost of motion (or time) and amount of information gathered. As previously stated, most previous work seek a trade-off between such criterion and consider only the next action. In the case of a whole plan, such simple aggregation lose sense, because a trade-off at every step may lead to poor decision for every criterion: for instance, if we consider that information gathering influences costs of motion (a path planner will give better decisions if the model is better), then emphasizing on information gathering upon minimizing cost of motion on the short term may lead to better results in the long run for both criteria.

Furthermore, while the cost of motion can be well-approximated (depending on the model of the dynamic of the system), the amount of information gathered is much more subject to uncertainty, and aggregating them directly is at least very arbitrary and at worst error-prone.

Contribution

Our aim in this paper is to adapt trial-based planning to the problem of exploration of unknown environments. This paper presents two main contributions among these aspects. First, we propose to use trial-based planning only as an exploration heuristic for building interesting plans and use in a second phase principled MCDM approaches on the built plans. Secondly, we adapt such approach in order to take into account the expectation of gathered information, not relying on a static model of future rewards. In our minds, it opens an avenue for principled planning into unknown and dynamic environment for autonomous robots, at the intersection of automated planning, knowledge modeling and active learning.

The remain of the paper is structured as follows. Firstly, section 2 rephrases the bases of Monte-Carlo Tree Search (precisely in subsection 2.1) then formalizes the proposed contributions on multi-criterion action selection in subsection 2.2 and then on non-static reward model in subsection 2.3. Subsection 2.4 gives the final details of the proposed algorithm. Section 3 concentrates on experimental evaluation by first precising the experimental setup in subsection 3.1. We then evaluate the benefits of multi-criterion based selection of actions and the effects of dynamic reward model. We then conclude and show some perspectives in section 4.

2 Extending Trial-based planning to unknown Environments

In this section, we introduce Monte-Carlo Tree Search and its best known variant: UCT, following [4]. We then extend and discuss such framework for problems in which the models are not reliable at least at the beginning of the mission.

2.1 Trial-based planning

We introduce some notations:

- $\mathcal{S}$ is set of states, $s \in \mathcal{S}$ a state, s_0 denotes the initial or current state of the agent;
- $\mathcal{A}$ the set of actions; $A(s)$ the set of possible actions at state s, a node is *terminal* if $A(s) = \emptyset$;
- $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ the transition function (assumed deterministic in this presentation for the sake of clarity, algorithms can be extended in a straightforward way for probabilistic domains with some $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$);
- $\mathcal{V}$ is a set of nodes of a search tree, $v \in \mathcal{V}$ a node, $d(v)$ denotes the depth of v in the tree, $s(v) \in \mathcal{S}$ the state associated to a node, $Ch(v) \in 2^{\mathcal{V}}$ is the set of children of v, $Pa(v) \in \mathcal{V}$ is the parent of v, $a(v) \in \mathcal{A}$ is the action that *led* to $s(v)$ from $s(Pa(v))$;
- $N(v) \in \mathbb{N}$ is the number of times a node has been visited;
- $Q(v) \in \mathbb{R}$ is the value (or quality) of a node, i.e. the expected or estimated sum of rewards and costs among the future trajectory of the system after $s(v)$ has been visited.

The general MCTS is detailed as algorithm 1. The idea is to create or select a leaf node, using some heuristic (see below), to estimate its value and propagate back this data upon the tree (or at least the path that lead to this leaf). While this procedure generally converges to optimal solution, it is generally used in an anytime fashion, as it focuses on interesting parts of the tree if the heuristic is well-designed.

Algorithm 1: `MCTS(`s_0`)`

```
Input: initial state $s_0 \in \mathcal{S}$
Result: best action for $s_0$
1 create root node $v_0$ with $s(v_0) = s_0$ ;
2 while within computational budget do
3     $v_l \leftarrow$ ChooseLeafNode($v_0$) ;
4     $\Delta \leftarrow$ EstimateReward($s(v_l)$) ;
5     Backup($v_l, \Delta$) ;
6 return SelectAction($v_0$)
```

The *ChooseLeafNode* algorithm (shown as algorithm 2) follows a path from initial state (or current state where the decision is to be made) and returns a leaf node of the tree that is either a terminal state or a new state. While in most variants it may be useful to go as far as a terminal state (i.e. $D = \infty$ in algorithm 2), in our case we prefer to add a limit on the depth of the tree, as the model of the environment is highly uncertain, and thus consider terminal all nodes at depth D.

Algorithm 2: `ChooseLeafNode(`v`)`

```
Input: a node of the search tree $v \in \mathcal{V}$
Data: the max allow depth of the tree $D \in \mathbb{N}$
Result: a leaf node (newly added or terminal)
1 if $v$ is terminal or $d(v) \geq D$ then
2     return $v$
3 if $v$ not fully expanded then
4     return Expand($v$)
5 return ChooseLeafNode(BestChild($v$))
```

Expansion of a node is straightforward, as shown in algorithm 3.

Algorithm 3: `Expand(`v`)`

```
Input: a node of the search tree $v$
Result: a new node
1 choose $a \in$ untried actions from A(s($v$)) ;
2 Ch($v$) $\leftarrow$ Ch($v$) $\cup \{v'\}$ with s($v'$) $= T$(s($v$),$a$) ;
3 Pa($v'$) $\leftarrow v$ ;          // memorization of parent
4 a($v'$) $\leftarrow a$ ;           // memorization of action
5 return $v'$
```

In the popular UCT [12] instance of MCTS, the choice of the path to follow is based on the UCB1 regret bound, shown as algorithm 4 which is a well-designed trade-off between exploitation of previous trials and exploration of promising parts of the tree.

Algorithm 4: `UCT-BestChild(`v`)`

```
Data: a tuning parameter $c$
Input: a node $v$
1 return $\text{argmax}_{v' \in \text{Ch}(v)} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$
```

When a leaf node is chosen, its value has to be evaluated, which can be done using random walk and returning reward of terminal node as in UCT (shown as algorithm 5). For some problems, a direct heuristic evaluation may be available.

Algorithm 5: `UCT-EstimateReward(`s`)`

```
Input: a state $s$
Result: a reward
1 if $s$ is terminal then
2     return reward for state $s$
3 else
4     choose $a \in A(s)$ uniformly at random ;
5     return UCT-EstimateReward($T(s, a)$)
```

Then, upon evaluation of the value of the state associated to a leaf node, the new data has to be backuped along the path of the tree that lead to this state. This is shown as algorithm 6 (which also updates the count of visited nodes).

Algorithm 6: `UCT-Backup(`v,Δ`)`

```
Input: a node $v$
Input: a reward $\Delta$
1 $N(v) \leftarrow N(v) + 1$ ;
2 $Q(v) \leftarrow Q(v) + \Delta(v)$ ;
3 if Pa($v$) $\neq \emptyset$ then
4     UCT-Backup(Pa($v$),$\Delta$)
```

Finally, once the tree has been built, the MCTS algorithm gives the estimated best action. This is generally done using the same criterion used to choose nodes for tree expansion, as shown in algorithm 7 for the case of UCT.

Algorithm 7: `UCT-SelectAction(`v`)`

```
Input: a node for which a decision is asked $v$
1 return a(BestChild($v$))
```

Algorithms 5 and 6 implicitly assumes that rewards are only present at terminal nodes. To generalize for MDPs-like models were rewards may be defined on any transition, we need to introduce a reward function $R : S \times A \times S \to \mathbb{R}$ and use the algorithms 8[1] and 9 (MC for Monte-Carlo). Other backup methods (among other aspects) are discussed and evaluated in [11], particularly if the transition model is probabilistic and available (partial or full Bellman backup).

This algorithmic framework has shown good results in domains where the transition function and the reward function are fixed and well-known, e.g. games and Markov Decision Processes, which is a reason why the models used are not emphasized in this presentation. In the case of exploration of unknown environment, on the contrary, both R and T are not well-known, and subject to change during the mission.

[1] For practical reasons and depending on the models, it may be interesting to limit length of such random walks and/or to sample among them.

Algorithm 8: `MC-EstimateReward(`s`)`

```
Input: a state $s$
Result: an estimated reward
1 if $s$ is terminal then
2     return 0
3 else
4     choose $a \in A(s)$ uniformly at random ;
5     $s' \leftarrow T(s, a)$ ;
6     return $R(s, a, s')$ + MC-EstimateReward($s'$)
```

Algorithm 9: `MC-Backup(`v, Δ`)`

```
Input: a node $v$
1 $N(v) \leftarrow N(v) + 1$ ;
2 $Q(v) \leftarrow R($s(Pa($v$)), a($v$), s($v$)$)$ ;
3 if $v$ is a leaf then
4     $Q(v) \leftarrow Q(v) + \Delta$
5 else
6     $Q(v) \leftarrow Q(v) + \frac{\sum_{w \in \text{Ch}(v)} N(w) \cdot Q(w)}{N(v)}$
7 if Pa($v$) $\neq \emptyset$ then
8     MC-Backup(Pa($v$),$\Delta$)
```

2.2 About multi-criterion rewards for trial-based planning

Considering that the reward model is only a raw estimation of the future rewards, and having in mind that several criterions each one having their own forecast model of different qualities, one of the first thing to do is not to unconditionally rely on the aggregated value Q associated to future state. We thus propose to use such $Q(v)$ aggregation only as a heuristic for plan generation, and take decision using such plans without aggregating criterions.

For doing so, we need to extract plans from the MCTS tree and evaluate them using fair multi-criterion evaluation. We thus use algorithm 10 in place of algorithm 7.

Algorithm 10: `XX-SelectAction(`v`)`

```
Input: the root node $v$
Data: the length of plans to evaluate $D$
Result: Fair multi-criterion evaluation
1 $\mathcal{P} \leftarrow \emptyset$ ;
2 foreach $\{w \in \mathcal{V} / d(w) = D\}$ do
3     $\mathcal{P} \leftarrow \mathcal{P} \cup \{(\ldots,$ a(Pa(Pa($w$))), a(Pa($w$)), a($w$)$\}$
4 return first action of BestPlan($\mathcal{P}$)
```

To evaluate the different extracted plans, we estimate the cost and the gain of information for each of them. For the first criterion, the values are additive and the global cost of a plan is equal to the sum of the cost of each move in this plan. For the second criterion, we consider the gain of information between the map the last node position can provide and the initial map. Once the criteria of each plan are calculated, we can identify the best path thanks to a multi-criteria decision making method. In this paper, we chose to use the method presented in [16].

The PROMETHEE multi-criteria decision making method is based on a two-by-two comparison of all the possible decisions, for each criterion we consider. Then, a preference degree is assigned to each candidate, that allows us to identify the best one. The strength of this method, compared with a classical NBV strategy, is that we aggregate normalized degrees of preference instead of criterion themselves. In our case, the possible decisions are the plans extracted from the tree, and the criteria are the global cost of a plan and its final gain of information. At the end, we will obtain a satisfying plan for both cost and gain criteria.

We introduce the following notations:

- $\mathcal{P}$ is the set of possible plans and p, p' are two plans;
- C is the set of the considered criteria, $c_j \in C$ is the j^{th} criterion, we denote $c_j(p)$ is the value of the criterion c_j for the candidate plan p ($c_j : \mathcal{P} \rightarrow \mathbb{R}$);
- $w_j \in \mathbb{R}$ is the weight associated to the criterion $j \in C$.

The PROMETHEE multi-criteria decision making method has four steps. First, for each pair of possible plans and for each criterion, we calculate the associated preference degree based on the distance between values a criterion gives to two plans. Formally, for some normalisation function $F : \mathbb{R} \rightarrow [0, 1]$, the preference degree $P_j(p, p')$ of p on p' is defined as:

Definition 1. *Preference degree:*

$$P_j(p, p') = F(c_j(p) - c_j(p'))$$

Then, for each pair of decisions, the preference degrees associated to each criterion are aggregated in one global preference index. The global preference index $\pi \in \mathbb{R}$ associated to the pair of decisions (p, p') is defined as:

Definition 2. *Preference index:*

$$\pi(p, p') = \sum_{j \in [1..|C|]} w_j \cdot P_j(p, p')$$

Using preference indexes, we then compute for each plan the outranking flows. For each candidate plan p, we denote $\phi^+(p)$ its positive outranking flow and $\phi^-(p)$ its negative outranking flow:

Definition 3. *Positive and negative outranking flows:*

$$\phi^+(p) = \frac{1}{(|\mathcal{P}| - 1)} \sum_{p' \in \mathcal{P}} \pi(p, p')$$
$$\phi^-(p) = \frac{1}{(|\mathcal{P}| - 1)} \sum_{p' \in \mathcal{P}} \pi(p', p)$$

Finally we are able to establish a final ranking between the candidates. To do so, a net outranking flow $\phi(p)$ is computed for each possible decision p from the positive outranking flow and the negative outranking flow. The best decision will be the one with the higher value of net outranking flow.

Definition 4. *Net outranking flow:*

$$\phi(p) = \phi^+(p) - \phi^-(p)$$

The multi-criteria decision making method applied on the choice of a plan is showed in the algorithm 11.

2.3 Using non-static model of reward

As the environment to explore is unknown, the precise reward gathered may not be straightforwardly modeled. More precisely, the reward is not state-dependant, but path dependant. To understand this,

Algorithm 11: `BestPlan(`$\mathcal{P}$`)`

```
  Input: the set of plans $\mathcal{P}$
  Result: best plan
  Data: the set of criteria $C = \{c_1, \dots, c_{|C|}\}$
  Data: the weights associated to criteria $w_j$
  Data: the normalisation function $F$
1 forall the $(p, p') \in \mathcal{P}^2$ do
2     forall the $c_j \in C$ do
3         compute $P_j(p, p')$ using definition 1
4     compute $\pi(p, p')$ using definition 2
5 forall the $p \in \mathcal{P}$ do
6     compute $\phi^+(p)$, $\phi^-(p)$ and $\phi(p)$ using definitions 3 and 4 ;
7 return $\mathrm{argmax}_{p \in \mathcal{P}}\, \phi(p)$
```

let us consider a simple model of observation where information is gathered at a fixed radius r around the robot. If one path is made of points far from each others of a distance $d \ll r$, then the information gathered at a given point is much smaller than information gathered by the same robot using the same sensor, but along a path made of points far from each other of a distance $d \geq r$, because of the overlapping of observation. One could force the choice of points in path (and then T) to minimize such aspects, but we prefer to update R incrementally during the decision making process, leading to more flexible and robust decision scheme.

To do so, we define R_v the reward function updated for node v and use the algorithm 12 in place of algorithm 3 to update the reward model R along the search tree.

Algorithm 12: `XX-Expand(`v`)`

```
  Input: $v$, a node of the search tree
  Result: a new node
1 choose $a \in$ untried actions from $A(s(v))$ ;
2 Ch($v$) ← Ch($v$) ∪ {$v'$} with $s(v') = T(s(v), a)$ ;
3 Pa($v'$) ← $v$ ;              // memorization of parent
4 $a(v') = a$ ;                 // memorization of action
5 $R_{v'}$ ← updateR($v'$) ;
6 return $v'$
```

This model will be used in the algorithm 9 by simply replacing R with R_v at line 2.

Algorithm 13: `updateR(`v`)`

```
  Input: a node $v$
  Data: current node $s$ = s(v)
  Data: parent node $t$ = s(Pa(v))
  Data: parent's reward model $R_t$
  Data: last action $a$ = a(v)
1 ;
2 ;
3 return newR
```

Présentation de différents modèles de capteurs et leur impact algo de maj du modèle global, intégré à MCTS + variantes (maj à chaque acquisition, discussion sur l'horizon de planif etc...)

In order to update the reward function, we have to simulate the map the robot would produce if the node v is chosen. This new map depends on the knowledge we already have from the current observation point, but also on the sensor properties. The principle is simple:

the map is encoded with whether a value reflecting the propability of the presence or not of an obstacle, or the value -1 if we have no knowledge at all. To simulate the new map associated with the node v, we "scan" the area the sensor would be able to discover, and each cell of the map in this area will be assigned a new probability. If we did not know anything (value -1), a low probability will be assigned to simulate the fact we gained an information but we are not sure of the observation there. In the other cases, the probability will be increased, whether there is an obstacle or not.

To do so, the change of probability depends on the distance between the cell of the map we consider and the position associated to the node v. Indead, in order to have a reaslistic simulation, the certitude will decrease as the considered position will be far from the sensor. As a consequence of this, the gain of probability will directly depends on this distance. In a first time, we simulated this gain with a linear function, the gain decreasing as the distance increase. But this function could be replaced, for example by a sigmoid function, modeling the fact that we can gain a lot of information near the observation position, but that we have a great incertainty as we move away from the sensor.

Regarding the cost estimation, the method we used is more simple. As we only generate candidate positions near the current position in the expansion of the tree, we took the liberty to consider there will be no obstacle on the way to this position. Then, we estimate the cost to go to this position as the euclidean distance to it.

During the expansion of the tree, estimated gain and cost are associated to each created node. The cost is additive, it can thus be added to the cost of the parent nodes to know the global cost of a path. The new map is most of a "mask" which can be added to the precedent map in order to know the new one if we chose the associated node. The gain a node can provide is deducted from this new map.

2.4 Algorithm

Explicitation de T: en particulier l'espace d'état est implicite, et on génère des candidats

Présentation formelle détaillée (suivant ce qui a déjà été présenté plus haut)

To sum up, the algorithm generates a tree thanks to MCTS, using an updated model of reward at each node. In a first time, we only use the best child of the root node. Thus, the robot moves to the associated pose, the map is updated, and the MCTS-algorithm is called again to generate the next pose, and so on. In a second time, we only use MCTS to generate a tree and we extract its different paths. We then choose the best one thanks to the MCDM approach, and the robot moves to the different poses contained in this path. When it is on the last pose, the algorithm loops back.

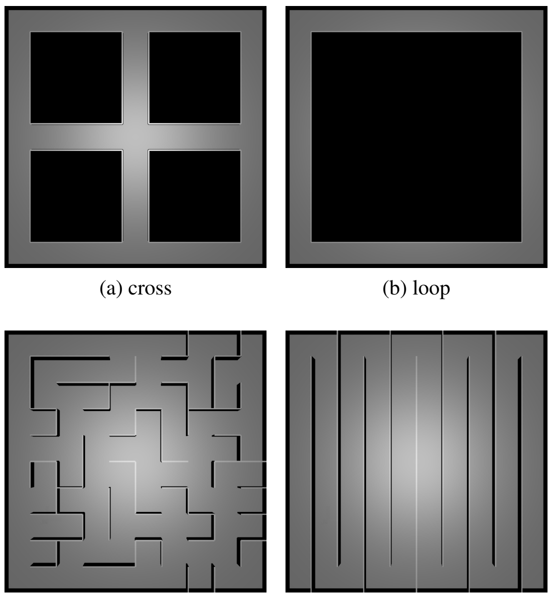
During the expansion of the tree, each node has a maximum fixed number of children. Because we chose to consider the cost of a new pose as the euclidean distance to it, we only generate candidate poses in already known area. Thus, these candidates can be generated at least in he sensor field of view, but also in the already discovered area around the robot.

3 Evaluation

3.1 Setup

The evaluations have been made using the MORSE simulator [5], connected to a guidance and mapping architecture from [8]. The mapping algorithm used in this architecture is GMapping [10].

In order to evaluate our exploration strategies, we have used the benchmark proposed in [18], which consist in four environments with different characteristics, shown in Fig. 1.


(a) cross (b) loop

(c) maze (d) zigzag

Figure 1: Simulation environments

parler de :

- envoie des consignes ; fréquence de replanif
- configuration vitesses de depl. / carto (et capteur)
- position initiale

3.2 Multi-criterion vs. aggregation

We proposed earlier to add a multi-criteria decision making method on the different plans the UCT-tree provides. We used this method aiming at reducing the impact of a criterion aggregating values with different scales, and choosing the next move considering an entire path. We thus made the comparison of these two approaches. In a first time, we tested the Next Best View strategy with the criterion (2) proposed in [9]. Then, we tested the strategy using the MCTS-algorithm and moving the robot to the best child position of the current one. We finally compared these results with the MCDM method applied on the UCT-tree.

In the three situations, in order to have the most reliable comparison, candidates are generated the same way, as the new map simulation and the cost and gain estimations.

The NBV strategy allows us to have new informations very quickly. But its choices of positions are not always wise as it only focuses on the next pose. Consequently, when the robot is in a corridor, it does not notice it and does not give priority to a width exploration before a move forward. Besides, when the robot is in the middle of a fully known area, unless it uses a frontier-based approach (which is not the case here), the robot is stuck and does not know where to go.

3.3 On model update

In order to show the impact of an updated model on the reliability of a plan in a not well known environment, we compared the MCTS stretegy with and without an update of the map estimation during the expansion of the tree.

3.4 Global Algorithm

4 Discussion

Perspectives

Ajout explicite d'un critère "je préfère me déplacer en environnement connu" qui va équilibrer "je veux maximiser la prise d'information" et ainsi focaliser sur la frontière sans avoir à la construire explicitement

REFERENCES

[1] Francesco Amigoni and Alessandro Gallo, 'A multi-objective exploration strategy for mobile robots', in *International Conference on Robotics and Automation (ICRA)*, Barcelona, Spain, (2005).

[2] Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh, 'Learning to act using real-time dynamic programming', *Artificial Intelligence*, **72**, 81–138, (1995).

[3] Nicola Basilico and Francesco Amigoni, 'Exploration strategies based on multi-criteria decision making for search and rescue autonomous robots', in *Proceedings of 10th Int. Conf. on Autonomous Agents and Multiagent Systems*, pp. 99–106, (2011).

[4] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton, 'A survey of monte carlo tree search methods', *IEEE Transactions On Computational Intelligence And AI In Games,*, **4**(1), 1–43, (march 2012).

[5] Gilberto Echeverria, Séverin Lemaignan, Arnaud Degroote, Simon Lacroix, Michael Karg, Pierrick Koch, Charles Lesire, and Serge Stinckwich, 'Simulating complex robotic scenarios with MORSE', in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, Tsukuba, Japan, (2012).

[6] Jan Faigl, Miroslav Kulich, , and Libor Preucil, 'Goal Assignment using Distance Cost in Multi-Robot Exploration', in *International Conference on Intelligent Robots and Systems (IROS)*, Villamoura, Portugal, (2012).

[7] Jan Faigl, Olivier Simonin, and Francois Charpillet, 'Comparison of task-allocation algorithms in frontier-based multi-robot exploration', in *European Conference on Multi-Agent Systems (EUMAS)*, Prague, Czech Republic, (2014).

[8] Nicolas Gobillot, Charles Lesire, and David Doose, 'A modeling framework for software architecture specification and validation', in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, Bergamo, Italy, (2014).

[9] Hector González-Baños and Jean-Claude Latombe, 'Navigation strategies for exploring indoor environments', *International Journal of Robotics Research (IJRR)*, **21**(10–11), 829–848, (2002).

[10] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard, 'Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters', *IEEE Transactions on Robotics*, **23**, 34–46, (2007).

[11] Thomas Keller and Malte Helmert, 'Trial-based heuristic tree search for finite horizon mdps', in *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS)*, (2013).

[12] Levente Kocsis and Csaba Szepesvári, 'Bandit based monte-carlo planning', in *Proceedings of European Conference on Machine Learning (ECML)*, pp. 282–293. Springer, (2006).

[13] Miroslav Kulich, Jan Faigl, , and Libor Preucil, 'On Distance Utility in the Exploration Task', in *International Conference on Robotics and Automation (ICRA)*, Shangai, China, (2011).

[14] Lourdes Muñoz-Gómez, Moises Alencastre-Miranda, Rigoberto López-Padilla, and Rafael Murrieta-Cid, 'Exploration and map-building under uncertainty with multiple heterogeneous robots', in *IEEE International Conference on Robotics and Automation (ICRA)*, Shangai, China, (2011).

[15] Giuseppe Oriolo, Marilena Vendittelli, Luigi Freda, and Giulio Troso, 'The SRT method : Randomized strategies for exploration', in *International Conference on Robotics and Automation (ICRA)*, New Orleans, LA, USA, (2004).

[16] Patrick Taillandier and Serge Stinckwich, 'Using the PROMETHEE Multi-Criteria Decision Making Method to Define New Exploration Strategies for Rescue Robots', in *International Symposium on Safety, Security and Rescue Robotics (ISRR)*, Kyoto, Japon, (2011).

[17] Luis Valentin, Rafael Murrieta-Cid, Lourdes Muñoz-Gómez, Rigoberto López-Padilla, and Moises Alencastre-Miranda, 'Motion strategies for exploration and map building under uncertainty with multiple heterogeneous robots', *Advanced Robotics*, **28**(17), 1133–1149, (2014).

[18] Zhi Yan, Luc Fabresse, Jannik Laval, and Noury Bouraqadi, 'Metrics for performance benchmarking of multi-robot exploration', in *International Conference on Intelligent Robots and Systems (IROS)*, Hamburg, Germany, (2015).