Improving the Efficiency of Online POMDPs by using Belief Similarity Measures

Joaquín Ballesteros[1], Luis Merino[2], Miguel Ángel Trujillo[3], Antidio Viguria[3] and Aníbal Ollero[1,3]

Abstract— **In this paper, we introduce an approach called FSBS (Forward Search in Belief Space) for online planning in POMDPs. The approach is based on the RTBSS (Real-Time Belief Space Search) algorithm of [1]. The main departure from the algorithm is the introduction of similarity measures in the belief space. By considering statistical divergence measures, the similarity between belief points in the forward search tree can be computed. Therefore, it is possible to determine if a certain belief point (or one very similar) has been already visited. This way, it is possible to reduce the complexity of the search by not expanding similar nodes already visited in the same depth. This reduction of complexity makes possible the real-time implementation of more complex problems in robots. The paper describes the algorithm, and analyzes different divergence measures. Benchmark problems are used to show how the approach can obtain a ten-fold reduction in the computation time for similar obtained rewards when compared to the original RTBSS. The paper also presents experiments with a quadrotor in a search application.**

I. Introduction

Acting robustly in dynamic and outdoors environments requires to cope with the uncertainties inherent to the limitation of sensors, imprecise models, errors, etc. In the last years, decision-theoretic planning methods coping with uncertainties, like Partially Observable Markov Decision Processes (POMDPs) [2] are more and more often used in robotics [3], [4], [5]. However, the broader application of planning under uncertainties methods faces a road blocker due to the curse of dimensionality (both, with respect to the state space and to the action-observation histories).

Two main kinds of algorithms can be found. In one hand, offline POMDP algorithms try to find offline the best action to be performed under all possible situations. Due to the inherent complexity, the first proposed algorithms of this kind were applied to just the simplest problems. Several algorithms have been proposed in the last years to deal with larger state, actions and observation spaces [6], [7], [8]. Most of these algorithms require to recompute the full policy when small changes on the environment dynamics happen.

Online forward-search-based POMDP algorithms have been proposed in the last years as an alternative for planning under uncertainties [9], [4]. In these online settings, the planner tries, for a certain lookahead planning horizon, to search for the best next action from an initial belief. These algorithms create a tree representing the reachable belief space for the horizon considered, and the objective is to find the best route (the one with a highest expected reward) through the tree in order to determine the best action to perform. After deciding and executing the action, a new planning iteration is performed. The problem again is that this search scales exponentially with the planning horizon.

Different strategies are considered to overcome this complexity. In some strategies, offline and online methods are combined: the offline method is used to obtain bounds that are later used in the online phase to prune branches of the belief tree to reduce the computational time [9]. In [10], the authors make use of the structure of certain problems, which allows the automatic construction of macro-actions, reducing effectively the action space for planning.

In this paper, we also consider a forward-search algorithm for online planning under uncertainties. We analyze an additional factor that can be used to improve the scalability of these methods. In general, the algorithms described above do not consider any metric within the belief manifold. Our motivating idea is that planning can be made more efficient by introducing an adequate metric or topological structure in it. In general, the tree representation used in forward-search will be actually a graph, where some belief points are visited more than once through different observation-action histories. Then we could reuse the computations for beliefs that have already been visited.

Here we present a first approach in that direction. We consider the use of similarity measures to determine the difference between belief points in the tree, so that we can determine nodes that represent the same belief point. By setting a threshold on these measures and adjusting it, a balance between complexity and optimality can be achieved. This reduction of complexity is very important for the real-time application of these methods in robotics.

The rest of the paper is organized as follows: the next section describes the general algorithm. Then, the different similarity measures considered are described. In the next section, the algorithm and the different measures are analyzed by using well-known benchmark problems, and compared with the algorithm RTBSS [1]. Experimental results in a quadrotor testbed are then presented. The adaptation to heuristic search algorithms is considered before finalizing with some conclusions and future work.

J.B., M.A.T., A.V. and A.O. are partially funded by the FP7 EC-SAFEMOBIL Project (288082). The work of L.M. is partially funded by the European Commission 7th Framework Programme under grant agreement no. 288235 (FROG)

[1]J. Ballesteros and A. Ollero are with the University of Seville, Seville (Spain) jballesteros,aollero@cartuja.us.es

[2]L. Merino is with Pablo de Olavide University, Seville (Spain) lmercab@upo.es

[3]M.A. Trujillo, A. Viguria and A. Ollero are with the Centre for Advanced Aerospace Technologies matrujillo,aviguria@catec.aero

II. FSBS: Forward Search in Belief Space Algorithm

A. Preliminaries

Formally, a discrete POMDP[1] is defined by the tuple $\langle S, A, Z, T, O, R, D, \gamma\rangle$ [2]. The *state space* is the finite set of possible states $s \in S$; the *action space* is defined as the finite set of possible actions $a \in A$; and the *observation space* consists of the finite set of possible observations $z \in Z$. At every step, an action is taken, an observation is made and a reward is given. After performing an action a, the state transition is modeled by the conditional probability function $T(s', a, s) = p(s'|a, s)$, and the posterior observation by the conditional probability function $O(z, a, s') = p(z|a, s')$. The reward obtained at each step is $R(s, a)$, and the objective is to maximize the sum of expected rewards, or *value*, earned during D time steps. To ensure that the sum is finite when $D \to \infty$, rewards are weighted by a discount factor $\gamma \in [0, 1)$.

As the state is non-observable, a belief function b is maintained by using Bayes rule. The belief obtained if we apply the action a and get the observation z is $b'(s') = \tau(b, a, z) = \eta O(z, a, s') \sum_{s\in S} T(s', a, s)b(s)$. The normalization constant:

$$\eta = P(z|b, a) = \sum_{s'\in S} O(z, a, s') \sum_{s\in S} T(s', a, s)b(s) \qquad (1)$$

gives the probability of obtaining a certain observation z after executing action a for a belief b. As said above, the objective is to determine the policy $a = \pi(b)$ that maximizes the cumulative reward, or value $V^{\pi}(b)$:

$$V^{\pi}(b) = R(b, \pi(b)) + \gamma \sum_{z\in Z} P(z|b, a)V^{\pi}(b^{z}_{\pi(b)})) \qquad (2)$$

where $R(b, a) = \sum_s R(s, a)b(s)$ is the expected immediate reward[2]. The value of the optimal policy is usually denoted by $V^*(b)$ and associated to it is the optimal Q function:

$$Q^*(b, a) = R(b, a) + \gamma \sum_{z\in Z} P(z|b, a)V^*(b^z_a)) \qquad (3)$$

Forward-search algorithms create an AND-OR tree by exploring the next beliefs for all possible actions and for all possible observations starting at an initial belief (see Fig. 1). The branching factor in AND-OR POMDP trees is $|A||Z|$ where $|A|$ is the number of actions and $|Z|$ is the number of observations, and the number of leaves nodes for a tree of depth D is $(|A||Z|)^D$. In [9], a classification of POMDP online algorithms can be found, and also three strategies that are employed to improve the computing time required to choose the best action:

- Monte Carlo sampling algorithms: minimize the branching factor by sampling a subset of observations.
- Heuristic search algorithms: guide the search of the most relevant branch nodes.
- Branch and Bound algorithms: Prune nodes that are suboptimal compared to other that have already been expanded.

[1]In this paper we will limit our analysis to discrete POMDPs, although most ideas can be considered in the continuous state case

[2]$b^z_a = \tau(b, a, z)$ will be used to obtain a more compact formula.

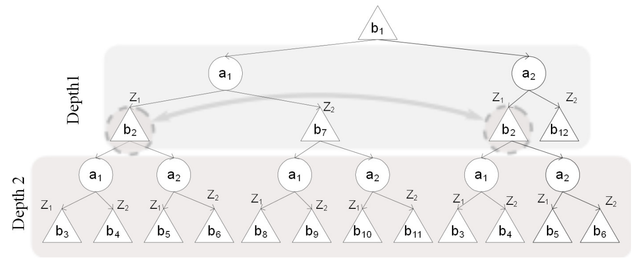

Fig. 1: An AND-OR Belief Tree with 2 actions and 2 observations. The OR-node are represented by triangles and the AND-nodes by circle. If a repeated belief appears at the same depth, it means that it has an identical subtree and the same value (like belief b2).

B. The FSBS algorithm

The algorithm proposed is here is of the Branch and Bound kind, and proceeds via look-ahead search up to a fixed depth d. We use the structure of the RTBSS algorithm proposed by [9] to elaborate the FSBS (see Algorithm 1). The algorithm uses the max-planes lower bound [11] of the optimal value implemented in [12] (line 3). The δ function determines how the FSBS algorithm propagates this lower bound from the leaves up to the root.

$$\delta(b, 0) = LowerBound(b)$$
$$\delta(b, d) = \max_{a\in A}(R(b, a) + \gamma \sum_{z\in Z} Pr(z|b, a)\delta(b^z_a, d - 1))$$

The main idea of the algorithm is to reuse the calculated rewards for the nodes that have been already explored at the same depth. It is the heuristic that allows us not to expand nodes and aims at minimizing the branched nodes as much as possible. Therefore, if the next belief to be expanded is similar to a saved belief at the same depth, that belief will not be expanded because its value is already calculated (see Fig. 1). The similarity between beliefs is computed by using one of the divergence measures described in Section III.

To determine the similarity between beliefs, we keep a node list for each depth d, $nodeList_d$. Every node contains the following items: the belief b; an action; and the accumulated reward if that action is applied to the belief, $\delta(b, d)$. We only keep the beliefs up to depth $D-1$ because the leaf nodes cannot be expanded.

The function *orderbyAccReward* in line 5 is used to find the accumulated rewards that are obtained when we apply each action to the current belief.

For each action at depth d, the *orderbyAccReward* function looks for a similar belief in $nodeList_d$, b', considering a similarity threshold *th*. If a belief is successfully found, the accumulated reward that was already stored, $\delta(b', d)$, is assigned to it, and if not, it is assigned as ∞ in order to

Algorithm 1: FSBS Algorithm

```
1:  function FSBS($b, d, th$)
2:      if $d == 0$ then
3:          return $LowerBound(b)$
4:      end if
5:      $\{st_1, st_2, ..., st_{|A|}\} \leftarrow orderbyAccReward(b, d, th)$
6:      $L_T(b) \leftarrow -\infty$
7:      $i \leftarrow 0$
8:      while $i < |A|$ $AND$ $st_i.AccReward > L_T(b)$ do
9:          $a \leftarrow st_i.IdAction$
10:         $rAcc \leftarrow st_i.AccReward$
11:         $L_T(b, a) \leftarrow -\infty$
12:         if $st_i.isFoundSimilar$ then
13:             $L_T(b, a) \leftarrow Reward(b, a) + \gamma rAcc$
14:         else
15:             $L_Z(b, a) \leftarrow \sum_{z \in Z} P(z|b, a) FSBS(b_a^z, d - 1, th)$
16:             $L_T(b, a) \leftarrow Reward(b, a) + \gamma L_Z(b, a)$
17:             $saveNode(b, a, d, L_Z(b, a))$
18:         end if
19:         $L_T(b) \leftarrow \max\{L_T(b), L_T(b, a)\}$
20:     end while
21:     return $L_T(b)$
22: end function
```

get expanded first. The *orderbyAccReward* function returns a list, sorted by accumulated reward, in which each element contains the following items: the action associated; the accumulated reward if this action is applied to the input belief if exists; and a variable that indicates whether the resulting belief is already in a depth on the tree or not.

In line 12-13 we reuse the accumulated reward if a similar node is found at the same depth. In this case, we stop expanding along that path. If not, we expand and keep the node (line 15-18). To finish we choose the action that maximizes the accumulated reward (line 21).

We calculate the optimal policy as:

$$\pi^*(b, D) = \arg\max_{a \in A}(R(b, a) + \gamma \sum_{z \in Z} P(z|b, a) FSBS(b_a^z, D - 1, th))$$

This is applied in each planning iteration. At every iteration, the optimal action is applied and then a new forward search is performed, in receding horizon fashion.

III. Determining the similarity between belief functions

The key idea of the FSBS algorithm is to define a similarity measure between belief points. For close enough belief points, we can reuse the computed lower bound on the Q function, and it is not required to re-explore the subtree rooted at that point.

There are different measures that can be used to determine the similarity between probability distributions. In this paper, we will analyze three measures from the field of information theory; namely, the Bhattacharyya distance, D_B (4); the Jensen-Shannon (JS) divergence D_{JS} (6), which uses the Kullback-Leibler divergence D_{KL} (7); and finally, the the Rénji divergence, a generalization of relative entropy; in particular the divergence of order 2, D_{R2} (5),. Given two discrete beliefs (probability distributions) $p(s)$ and $q(s)$, they are defined as:

$$D_B(p \parallel q) = -\ln\left(\sum_{s \in S} \sqrt{p(s)q(s)}\right) \tag{4}$$

$$D_{R2}(p \parallel q) = \log E[\frac{p}{q}] = \log \sum_{s \in S} p(s) \frac{p(s)}{q(s)} \tag{5}$$

$$D_{JS}(p \| q) = \frac{1}{2}(D_{KL}(p \| \frac{p+q}{2}) + D_{KL}(q \| \frac{p+q}{2})) \tag{6}$$

where

$$D_{\mathrm{KL}}(p \| q) = \sum_{s \in S} p(s) \ln \frac{p(s)}{q(s)} \tag{7}$$

These three statistical divergences have been used in many areas. Rénji divergence has been used in domain adaptation [13]. The Jensen-Shannon divergence has been used in imaging processing [14] and active learning settings [15]; also, the Bhattacharyya distance has been used in image processing for histogram comparison [16].

For our algorithm, the best choice is a statistical distance with a low computational complexity and that can compare any given couple of probability distribution. The computational complexity of all of them is $O(|S|)$, with $|S|$ the number of states. Among them, the Rénji divergence of order 2 is the fastest to compute. However, the main problem here is that the divergence is only defined if $q(s) > 0$ for all $p(s) > 0$, and therefore it leads to many cases in which the two beliefs are not comparable. Also, it is not symmetric. By contrast, the JS divergence is always defined for two fixed beliefs; moreover, it is bounded between 0 and 1 [17] and symmetric; the drawback is that it requires to calculate the KL divergence twice. Between these both statistical distances, we encounter the Bhattacharyya distance with a medium computational complexity.

Although some of the measures are sometimes called distances, as the KL distance, actually they are not proper distance measures, as the triangle inequality usually does not hold for them. However, the square root of the JS divergence is a metric [18], and thus we will focus the analysis on the JS divergence.

IV. Analysis using benchmarks

In order to evaluate the FSBS algorithm, we have implemented RTBSS [1] as baseline algorithm. We have used Trey Smith's library [12] for the implementations. Moreover, we have adapted the library so that it can be used in ROS (Robotic Operating System) [19]. We have used the POMDP parser and the structure provided by [12] to implement FSBS, and we have also used the lower bound included in it as utility function (Algorithm 1, line 3). For the RTBSS implementation, we have employed the upper bound implemented in it too.

For evaluation, we have considered two well known benchmark problems widely used to test and compare POMDP algorithms: RockSample [20] and Tag [21]. The methodology employed has been the following: the FSBS algorithm is executed, considering different thresholds in the similarity function used to compare belief points. Moreover:

- For RockSample we have executed all possible rock configurations twenty times for each statistical divergence measure of Section III and for each threshold.
- Regarding the Tag problem, we executed five times all the possible start configurations (trivial cases not included), only for the JS divergence and for each threshold.

We also executed RTBSS (using the lower and upper bound provided by [12]) and what we denominate the equal case (the FSBS algorithm in which two beliefs b and b' are marked as similar only if $\forall s \in S \; b(s) = b'(s)$) for the previous two benchmarks and the same configurations.

For each of these executions we have recorded the mean number of branch nodes used by the algorithm, the mean expected reward returned by the algorithm and the total time used per run of the problem. The figures below will show the mean of these values and their standard deviations for all the runs performed[3].

A. Rock Sample Benchmark

As RockSample7_8 is a problem with a branch factor 26, and we are comparing all statistical divergences for a depth of 4, we have therefore 18278 nodes that can be expanded. In Fig. 2 we can see the mean number of branch nodes per problem execution for the different options. All statistical divergences are represented on the upper horizontal axis except Rényi's divergence of order 2 which is represented in the lower horizontal axis. As said above, we compare here different thresholds for each statistical divergence measure. The RTBSS and equal comparison are duplicated for all threshold values for a more user-friendly reading.

As expected, the number of branch nodes decreases if we increase the threshold. The Bhattacharyya distance behaves worse than the JS case as when $p_i = 0$ it cannot differentiate the value for q_i and vice versa, so less nodes are found as being similar. The Rényi's divergence needs a higher threshold to prune because it is not defined in case q_i is equal to 0. It is important, moreover, to notice that even in the case of the removal of equal nodes, a significative reduction of nodes is achieved with respect to the original RTBSS.

More importantly, we also compare the expected reward to check how the reduction of nodes affects the performance of the planner. In Fig. 3 we can see the comparison of

[3]Please notice that the library employed does not use factored POMDP. The rewards obtained here are therefore worse than the ones that can be obtained with factored POMDPs, like in [9]. However, the conclusions are not affected by the particular implementation, as we are interested in the relative gain obtained by incorporating the divergence measures.

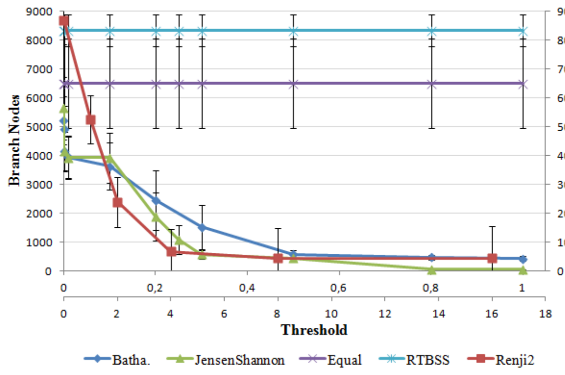

Fig. 2: Number of branch nodes (RockSample): the graphic shows the mean number of nodes and its variance for different divergence measures and thresholds. The RTBSS case and the equal case are shown as well.

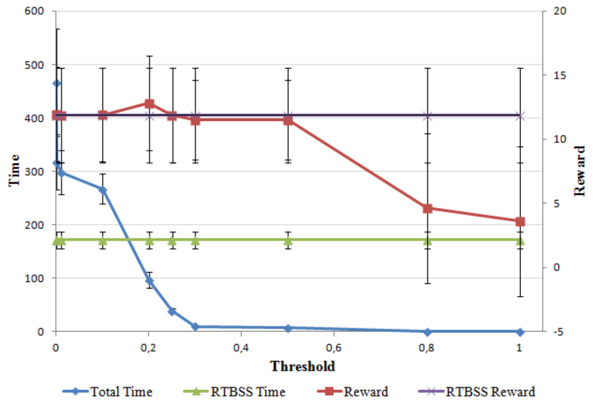

Fig. 3: Expected Reward (right vertical axis) and execution time (left vertical axis) for RTBSS and FSBS using the JS divergence (RockSample).

the expected reward for FSBS using the JS divergence and RTBSS. We observe how the expected reward of the JS case remains very close to the one obtained by the RTBSS for a threshold lower than 0.25. The figure also shows the execution time for both algorithms. Above this threshold our algorithm improves the time performance because it always expands a little quantity of nodes (less than 5%). For lower thresholds than 0.2, the Jensen Shannon divergence computation cost increases and makes the expected time worst than RTBSS. As expected, for larger thresholds the reward obtained is reduced.

In Fig. 4 we can see the comparison for the case of the Bhattacharyya distance. The Bhattacharyya distance has a similar behavior to the JS Divergence, as it remains very close to the expected value of RTBSS for thresholds lower 0.3. However, the expected reward for JS is most often better than for the Bhattacharyya distance. The lower computation cost improves the expected time, although the number of branch nodes is always higher than in the JS case for thresholds higher than 0.1 (see Fig. 2).

Regarding RockSample we can finally see in Fig. 5 the

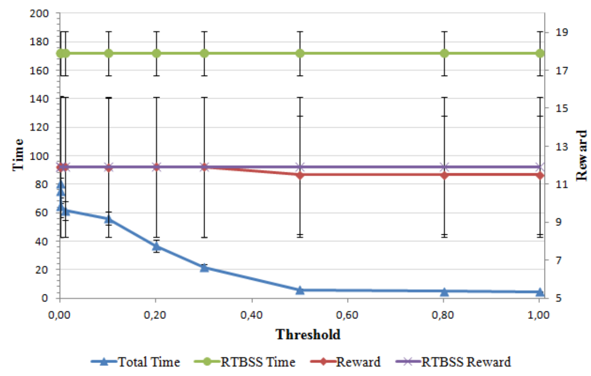

Fig. 4: Expected Reward (right axis) and execution time (left axis) for RTBSS and for FSBS using the Bhattacharyya distance (RockSample).

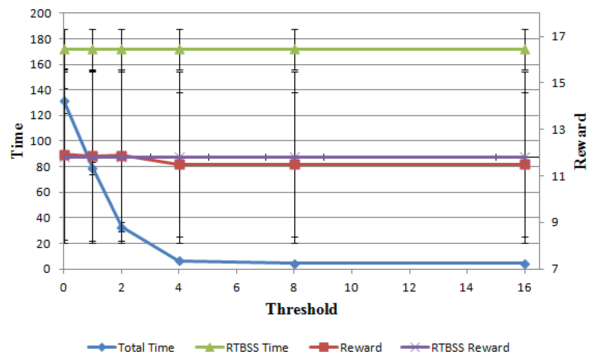

Fig. 5: Expected Reward (right axis) and execution time (left axis) for RTBSS and for FSBS using the Rényi's divergence of order 2 (RockSample)

comparison when using the Rényi's divergence of order 2. The reward also remains very close to the expected for thresholds lower than 2.

When comparing the mean execution time for each statistical distance, using the maximum threshold that maintains the expected reward equal in mean to the RTBSS (0.2 JS, 0.3 in Bhattacharyya and 2 in Rényi), it can be seen that the Bhattacharyya gets the lowest execution time.

B. Tag Benchmark

In the case of the Tag benchmark [21] the branching factor is 145. We carried out a comparison for depth 4, which implies that 3069795 nodes can be expanded, and we focus on the JS divergence in this case, as commented above. In Fig. 6 we can observe the mean of the number of branch nodes per problem execution for the different options. It can be seen how the FSBS algorithm with the JS divergence manages to reduce the number of branch nodes to nearly an 8% of the nodes expanded by RTBSS. As expected, the number of branch nodes decreases if we increase the threshold, and we therefore obtain the same behavior as in RockSample.

In Fig. 7 we can observe the comparison of the expected cumulative reward for FSBS using the JS divergence and

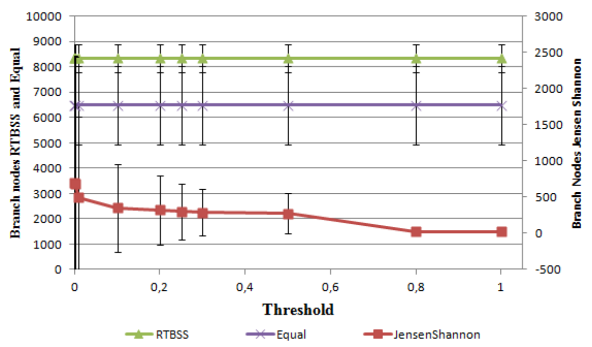

Fig. 6: Number of branch nodes (Tag): the graphic shows the mean number of nodes and its variance for Jensen Shannon divergence and thresholds (right axis). The RTBSS case and the equal case are shown as well (left axis).

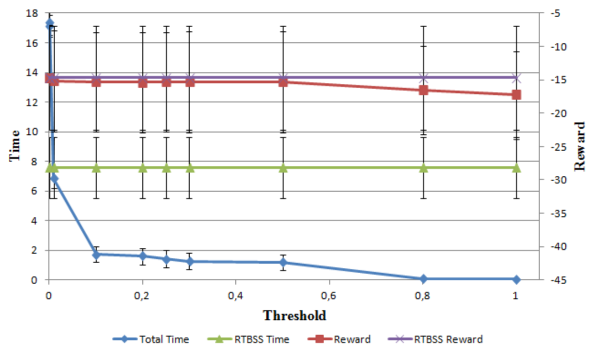

Fig. 7: Expected Reward (right axis) and execution time (left axis) for RTBSS and for FSBS using the JS divergence (Tag).

that of RTBSS. In this case, the expected reward of the JS case is nearly equal to the expected value of RTBSS for thresholds below 0.01. For higher thresholds, the reward remain fairly close to the to the RTBBS reward. However, time improves substantially, like in RockSample, as we increase the threshold, with a 90% reduction of execution time when compared to RTBSS for thresholds above 0.2.

Table I shows the total expected time used per run for different planning depths when using the Jensen Shannon divergence. We have chosen the following two thresholds: on one hand, 0.5 that gets a good compromise between reward and time in the tests shown previously; and on the other hand, 0.8 that gets a better total time against the obtained reward. As expected, the reward obtained in both cases increases with the planning horizon. Another observation is that the total time is more negatively affected when depth is increased by lower thresholds.

V. Experimental Results

We have also tested the algorithms in real time in the quadrotor testbed of the Center for Advanced Aerospace Technologies (CATEC), see Fig. 8. The testbed is equipped with 20 VICON cameras, which allow to know the position and orientation of the vehicles with great precision, and to perform experiments with up to 8 quadcopters at the same time.

TABLE I: Comparative table for different planning depths using Jensen-Shannon

Threshold	0.5		
Depth	5	6	7
Time	3.90	9.64	20.51
Reward	-14.90	-12.78	-12.32
Threshold	0.8		
Depth	5	6	7
Time	0.1	0.12	0.14
Reward	-17.17	-16.90	-16.71

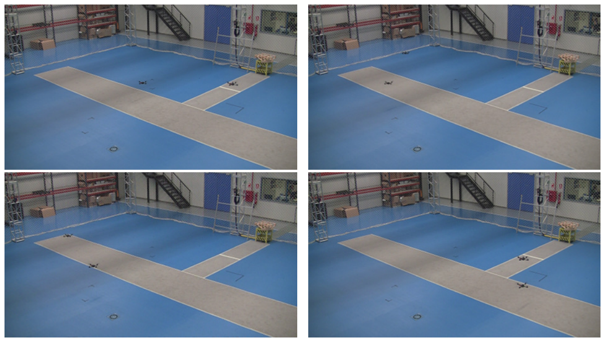
Fig. 8: Four snapshots of the experiments at CATEC's testbed. The last image shows the moment when the intruder is finally detected.

The ROS module implementing our online POMDP has been integrated into the testbed. In the experiments performed, the Tag problem is employed to model a search mission in which one quadcopter has to localize an intruder. The objective is to localize and indicate the position of the intruder.

The space is discretized into 27 cells of 80 cm. each. The pursuer can observe its own location. It also has a camera (simulated using the VICON system) with a field of view limited to the cell below the quadcopter. This camera is able to detect if the intruder is in that cell or not. The intruder knows the position of the pursuer, and tries to separate from it 80% of the time (the other 20% stays at the same cell). The actions that the pursuer can take are staying at the same cell or moving to one of the 4 neighbor cells. In this setup, the online POMDP acts as a planner that sends waypoints to the path planner. These waypoints are the centre of the next neighbor cell to move.

Figures 9 and 10 show the trajectories obtained in one experiment using a planning horizon of 4 and a threshold of 0.1 in the FSBS algorithm (using the JS divergence). In the test, the intruder is successfully detected around the second 260. The planning loop can be executed in real-time (well below 1 second each iteration, as the computers of the testbed are more powerful than the one used in the previous section).

VI. Adaptation to heuristic search algorithms

Our goal in this paper was to show that introducing similarity measurements between belief points in online POMDP algorithms reduces the complexity and minimizes

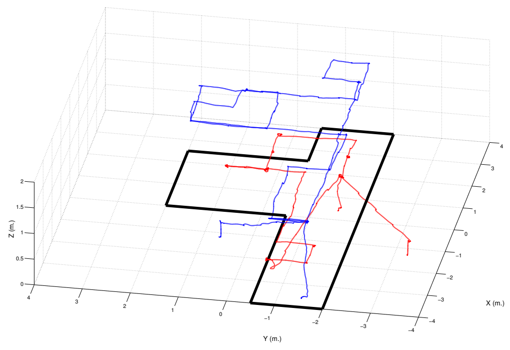

Fig. 9: 3D trajectories of the pursuer (blue) and the intruder (red). The quadrotors are limited to be in the indicated scenario (in black).

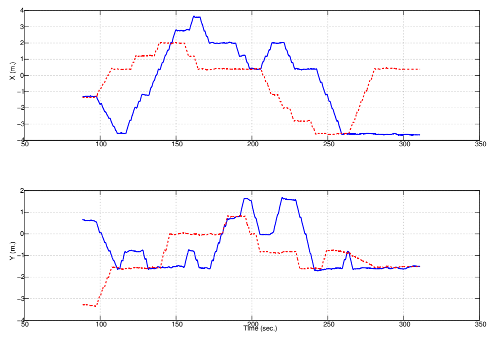

Fig. 10: X and Y coordinates as a function of time for the pursuer (blue) and the intruder (red). Around 260 seconds the pursuer detects the intruder when it is over it.

the number of nodes used. We have employed a branch and bound algorithm as baseline as branch and bound algorithms always explore the same tree for a fixed depth and a same given input, which enables us to carry out comparisons between trees with the same depth.

Of course, heuristic search algorithms can obtain better policies. In heuristic search algorithms, the belief tree is not completely explored for a fixed depth. Instead, the nodes are expanded according to a heuristic function until a timeout is exceeded. The heuristic computation has an additional computational cost and it should be known for all the fringe nodes before the candidate to be expanded is chosen. For this reason, the number of expanded nodes is far inferior than in branch and bound algorithms. However, heuristic search algorithms select only representative nodes, which usually improves the final result.

We implemented a version of AEMS2 [9] by using the measurement divergence in order to compare beliefs at the same depth. We based our evaluation work on the Rock-sample benchmark. We compared our AEMS2 version with the original AEMS2 by using the JS divergence and the Bhattacharyya distance, both with 0.2 threshold and a 2-second timeout.

TABLE II: Comparative table for different distance measurement in AEMS2

	AEMS2		
Distance	None	JS	Bahata
Rewards	14.2	14.24	15.09
Nodes	2758	98	141

The result can be observed in table II. It can be seen how, due to the additional complexity introduced by the comparisons, less nodes are expanded for the same planning time than in the original AEMS2 algorithm. However, the reward obtained is very similar. This leads us to conclude that more efficient implementations of the comparisons, for instance by using other data structures like kd-trees, can lead to enhance the performance of heuristic search algorithms.

VII. Conclusions and Future Work

In the paper, we have presented a forward-search algorithm for online planning under uncertainties. The algorithm considers some divergence measures to determine the similarity between belief nodes of the forward search tree, allowing to discard branches that are similar to branches already visited, and thus reducing the computational cost of the algorithm. The threshold considered for the divergence measure allows to tradeoff optimality and execution time. This allows applying online POMDPs to actual robotic problems with larger domains and/or planning depths.

We have presented comparisons with state of the art algorithms that support this claim using two benchmark problems. It has been shown that if the threshold remains below a certain value for a given statistical distance, the expected reward remains very close to the ϵ-optimal value calculated in RTBSS. Moreover, we have also tested the planner in real-time in a quadrotor testbed in a searching application in which the robot is equipped with very limited sensors. The same ideas can be applied to heuristic search algorithms.

As future work, we will consider more efficient data structures, like kd-trees and hashing tables, to reduce the computational burden of the distance comparisons. Furthermore, the same ideas will be used to build graph-like structures, in which the beliefs are compared at different levels of the tree; this way we expect that better results can be obtained when combining the approach with methods based on heuristic search. Another interesting line is to develop adaptive thresholds depending on the planning time available. We will also analyze the potential applications of results on information geometry [22] to the planning under uncertainty problem, by considering the introduction of proper metrics into the belief manifold.

References

[1] S. Paquet, B. Chaib-draa, and S. Ross, "Hybrid POMDP algorithms," in *Workshop in Multiagent Sequential Decision Making in Uncertain Domains (MSDM). The 10th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2006, pp. 133–147.

[2] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, pp. 99–134, 1998.

[3] S. Ong, S. W. Png, D. Hsu, and W. S. Lee, "POMDPs for Robotic Tasks with Mixed Observability," in *Proc. Robotics: Science and Systems, RSS*, 2009.

[4] R. He, A. Bachrach, and N. Roy, "Efficient planning under uncertainty for a target-tracking micro-aerial vehicle," in *Proc. International Conference on Robotics and Automation, ICRA*, 2010.

[5] J. Capitan, M. Spaan, L. Merino, and A. Ollero, "Decentralized Multi-Robot Cooperation with Auctioned POMDPs," in *The 6th Workshop in Multiagent Sequential Decision Making in Uncertain Domains (MSDM). The 10th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2011.

[6] M. T. J. Spaan and N. Vlassis, "Perseus: Randomized point-based value iteration for POMDPs," *Journal of Artificial Intelligence Research*, vol. 24, pp. 195–220, 2005.

[7] H. Kurniawati, D. Hsu, and W. Lee, "SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces," in *Proceedings of the Robotics: Science and Systems Conference*, Zurich, Switzerland, 2008.

[8] B. Bonet and H. Geffner, "Solving pomdps: Rtdp-bel vs. point-based algorithms," in *Proceedings of the 21st international jont conference on Artifical intelligence*, ser. IJCAI'09. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2009, pp. 1641–1646.

[9] S. Ross, J. Pineau, S. Paquet, and B. Chaib-draa, "Online planning algorithms for POMDPs," *Journal of Artificial Intelligence Research*, 2008.

[10] R. He, E. Brunskill, and N. Roy, "Efficient planning under uncertainty with macro-actions," *Journal of Artificial Intelligence Research*, vol. 40, pp. 523–570, February 2011.

[11] T. Smith, "Probabilistic planning for robotic exploration," Ph.D. dissertation, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, July 2007.

[12] ——, "ZMDP software for POMDP and MDP planning," http://www.cs.cmu.edu/ trey/zmdp/, 2012.

[13] Y. Mansour, M. Mohri, and A. Rostamizadeh, "Multiple source adaptation and the Rényi divergence," in *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 2009, pp. 367–374.

[14] J. Gómez-Lopera, J. Martínez-Aroza, A. Robles-Pérez, and R. Román-Roldán, "An analysis of edge detection by using the Jensen-Shannon divergence," *Journal of Mathematical Imaging and Vision*, vol. 13, no. 1, pp. 35–56, 2000.

[15] M. Aminian, "Active learning for reducing bias and variance of a classifier using Jensen-Shannon divergence," in *Proceedings of the Fourth International Conference on Machine Learning and Applications*, ser. ICMLA '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 43–48. [Online]. Available: http://dx.doi.org/10.1109/ICMLA.2005.7

[16] E. Choi and C. Lee, "Feature extraction based on the Bhattacharyya distance," *Pattern Recognition*, vol. 36, no. 8, pp. 1703–1709, 2003.

[17] J. Lin, "Divergence Measures Based on the Shannon Entropy," *IEEE Transactions on Information Theory*, vol. 37, pp. 145–151, 1991.

[18] D. Endres and J. Schindelin, "A new metric for probability distributions," *Information Theory, IEEE Transactions on*, vol. 49, no. 7, pp. 1858 – 1860, july 2003.

[19] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[20] T. Smith and R. Simmons, "Heuristic search value iteration for POMDPs," in *Proceedings of the 20th conference on Uncertainty in artificial intelligence*. AUAI Press, 2004, pp. 520–527.

[21] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for POMDPs," in *International Joint Conference on Artificial Intelligence*, vol. 18. Citeseer, 2003, pp. 1025–1032.

[22] S. Amari and H. Nagaoky, *Methods of information geometry*. American Mathematical Society, 2000.