FROG

FUN ROBOTIC
OUTDOOR GUIDE

Deliverable D3.3
Person guidance navigation component

Consortium

UNIVERSITEIT VAN AMSTERDAM (UvA)
IDMIND - ENGENHARIA DE SISTEMAS LDA (IDM)
UNIVERSIDAD PABLO DE OLAVIDE (UPO)
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE (ICL)
UNIVERSITY OF TWENTE (UT)

Grant Agreement no. **288235**

Funding Scheme: **STREP**

DOCUMENT INFORMATION

Project

Project acronym:	FROG
Project Full Title:	Fun Robotic Outdoor Guide
Grant agreement no.:	288235
Funding scheme:	STREP
Project start date:	1 October 2011
Project duration:	30 September 2014
Call topic:	ICT-2011.2.1 Cognitive Systems and Robotics (a), (d)
Project web-site:	www.frogrobot.eu

Document

Deliverable number:	D3.3
Deliverable title:	Person guidance navigation component
Due date of deliverable:	M33 - Sept. 30, 2014
Actual submission date:	November 7, 2014
Editors:	
Authors:	UPO
Reviewers:	All Partners
Participating beneficiaries:	UPO
Work Package no.:	3
Work Package title:	Behavior detection and human-aware guidance
Work Package leader:	ICL
Work Package participants:	All Partners
Estimated person-months for deliverable:	4
Dissemination level:	Public
Nature:	Prototype
Version:	2.0
Draft/Final	Final
No of pages (including cover):	40
Keywords:	Guiding, POMDP

Contents

List of Figures

List of Tables

Figure 1: FROG guiding a group of persons at the Lisbon Zoo

1 Introduction

The objective of FROG is to develop a fun robotic guide (see Fig. 1). While robot guides have been developed for more than a decade [52, 45], the project considers as new contributions the development of social behaviors and a wining robot personality by integrating social feedback, as well as the robust operation in outdoors crowded scenarios.

In scenarios involving interaction with humans, human awareness has to be taken into account in the entire robot planning and navigation stack [28], from task planning [1], task supervision and execution [7] to path planning and execution [46, 53, 55].

The project aims to demonstrate the operation of the robot in the Lisbon City Zoo and the Royal Alcazar in Seville. The basic scenario envisaged for the operation of the robot, described in Deliverable D1.1 [10] and depicted in Fig. 2, requires several capabilities from the point of view of the robot navigation stack:

- Waypoint navigation: the robot should wander around the place to attract the people attention.
- Approach people: if some people is detected, the robot should approach the people to inquiry if they may be interested on a guided tour.
- Guide persons between different Points of Interest (POIs), see Fig. 1.

While a description of the navigation stack and the first two tasks are described in D2.2 [12], in this deliverable we will describe the techniques developed for robot guiding of a person or a group of persons. In particular, in this new version, the final models employed and results from the experiments are also presented.

The document is organized as follows. In the next section, a discussion on the guiding task and the related state of the art is presented. Then, a short description of the architecture and sensing required for guiding is described. This is followed by an in depth explanation of the main model behind the guiding component, Partially Observable Markov Decision Processes, as well as the new techniques developed. Finally, and analysis and results obtained in simulation and real experiments are presented.

2 Robot guiding

As commented above, the deployment of robots as guides in museums has a long successful story, with Rhino [4], Minerva [52], Robox [45], Mobot [35], Rackham [8], Robotinho [16], among others. In these remarkable works, efficient navigation strategies were developed. And they raised new challenges on

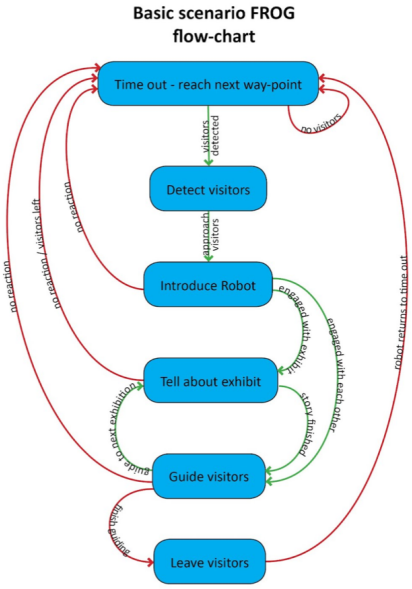


Figure 2: Basic scenario in FROG

human-aware robot navigation. In particular, on person guidance there are still issues that should be solved.

Early in the project, an analysis of human tour guides was performed and documented in D1.1 [10]. From the questionnaires and discussions with the subjects, several relevant comments and relevant aspects were obtained. They confirm what has been also found in the literature. In a guiding task, the way humans follow may vary greatly [37]. Furthermore, it is important to consider human commitment and intentions. Humans may suspend temporarily their commitment in the task (for instance, to make photos of a particular place, or to gather information of an interest place on the way to the destination), or completely give up the task at hand. Also, the motion of the guide can influence the motion of the persons.

Thus, in that direction, several works have been presented in the last years. In [17] probabilistic models of human-interaction are extracted from data for the purposes of person guiding. This work achieves more robust guiding velocities by selecting velocities on the path towards the goal depending on the distance to the goal and the person following. In [19, 20], methods for guiding groups of people by a team of cooperating robots are presented. The robots adapt the trajectories to avoid humans getting lost from the group. The key elements are the ability to predict future persons' positions to locally adapt the position of the robots.

One very important aspect of guiding is that the robot and the persons try to achieve a joint goal. As indicated in [28], this poses several navigation challenges, as the robot should not be over-reactive nor ignorant about the person's activities. Thus, pure reactive models are insufficient for the guiding task. Besides local adaptation, high-level decision making is required: the human commitment and goals should be considered [9, 8]. Thus, it is important that the robot maintains a belief state on the human intentions.

In this sense, the authors in [39] employ Markov Decision Processes (MDPs) to predict the destination of the persons in guiding applications. In [37], a framework has been presented to re-engage a guided person, in the case he/she suspends the guiding process and moves away significantly. In that case the robot re-plans a smooth path to approach the person in a goal-oriented manner so that to exert a kind of social force pulling towards the goal.

However, the person intentions are inherently non-observable. The robot should infer these intentions from measurements on the person positions, velocities and other similar features. Furthermore, it is important to deal with noisy observations, occlusions, etc.

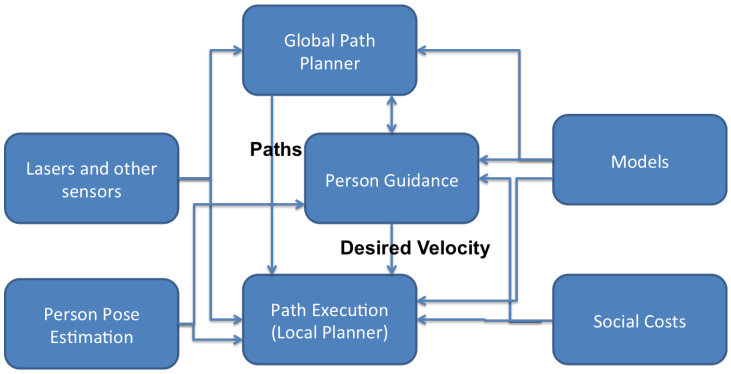


Figure 3: The FROG navigation stack is modified for the guidance application.

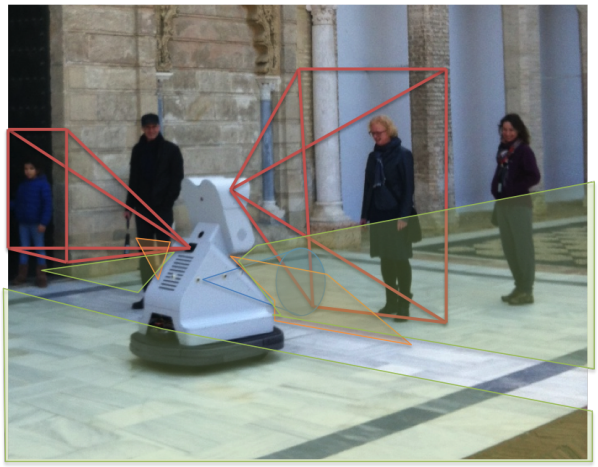


Figure 4: Sensors on board FROG. A front stereo pair provides a 3D information and persons detections and poses between 2.5 and 15 meters (in red). The horizontal lasers (green) are also processed to obtain persons positions estimations surrounding the robot. A (push-broom) tilted laser ahead is able to obtain 3D information below 2.5 meters (orange). Finally, some sonars are used to detect laterally obstacles.

Therefore, here we will employ Partially Observable Markov Decision Processes (POMDPs) as the model of the guiding task. These models allow to consider in a principled way uncertainties on actions and observations, and to reason over non-observable variables. Closest to our work, in [51] a POMDP is also used to infer the intentions of the person for wheelchair navigation. Similar ideas are considered, but in a different scenario. Moreover, offline POMDP models are used, so the system has to be re-planned offline if a different scenario is considered. In FROG, a new online POMDP technique is developed, able to build models on the fly and run in real-time. Furthermore, the proposed system is able to work with a larger scenario in terms of state space and observation outcomes.

2.1 High-level Design

The FROG navigation stack is described in D2.2 [12]. As indicated there, it follows the hierarchical division between a global planner and a local path execution.

The guiding component appears in the middle of the navigation stack for the particular task of person guidance (see Fig. 3). Its objective is to control how the robot follow the path planned to the guiding destination by considering the person position and person intentions.

It does that by controlling fundamentally the velocity of the local planner, which tries to follow the path provided by the global planner avoiding obstacles and avoiding other persons in a social way (see D2.2 [12]). Furthermore, if it is estimated that the person has abandoned the task, then it is able to cancel the current path.

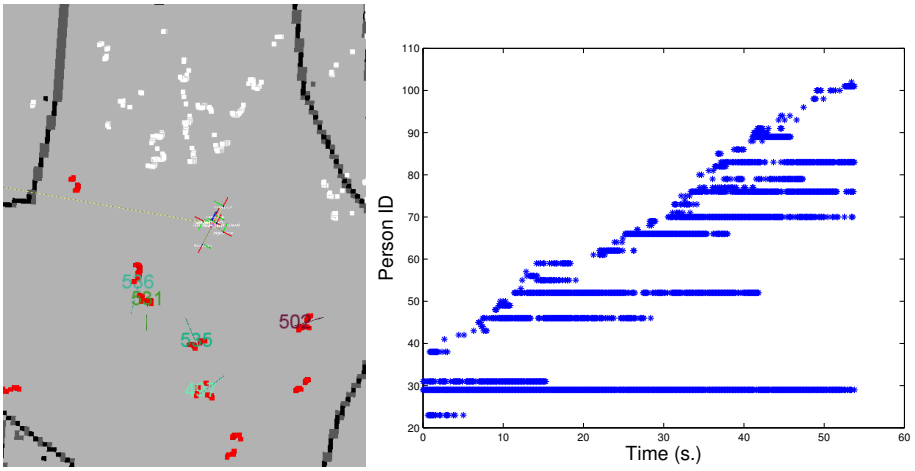


Figure 5: Person detection and tracking. Left: the detected legs are grouped and filtered to obtain person tracks, including velocities and orientations. Right: consistency of track IDs for one of the experiments between 2 POIs at the Royal Alcázar. Most of the persons tracked are not following or engaged with the robot. Person 29 is one of the persons being guided.

As it will be described later, this module receives the path to be followed, as well as the estimations on the persons poses and the robot pose.

3 Human-awareness

For the application of person guidance is key to be able to determine if the person/persons being guided are actually following the robot. Several sensors are used for person detection and localization on board the FROG robot (see Fig. 4).

The frontal stereo cameras provide the position and orientation of persons within their field of view by using the module developed by UvA and described in Deliverable D3.1 [13]. This module provides persons positions and orientations for persons between 2.5 and 15 meters (and the position up to 25 meters) in front of the robot. However, they cannot be used to estimate positions of the persons behind the robot.

A rearview camera was initially planned, to provide tracks of the heads of persons following the robot. However, in the final version of FROG this option was discarded.

Then, in order to have further information about the surrounding persons, and in particular for the guiding application, we also process the information from the 2D horizontal lasers. For that, we leverage the technique developed by Mozos et al. [34]. This algorithm is not as accurate as the UvA's module, but it is reliable enough to provide good estimations of persons around the robot. We use the front and rear lasers as inputs for this algorithm, so we are able to detect persons in 360° around the robot at good frequency (about 10Hz).

As this system provides information about each laser segment indicating if it constitutes a human leg or not, another two layers were developed to complement this information:

- A first layer is in charge of grouping pairs of legs in case of being near enough to be considered as a single person, considering the Euclidean distance between both legs and establishing a maximum threshold. For safety reasons, the remaining single legs are also considered as additional persons, due to situations where people stay with both legs close enough to be identified as a single segment in laser measurement.
- The second layer performs a simple tracking of people, by using nearest-neighbor data association and Kalman filtering to handle with people movement and small miss-detections, as can be seen in Fig. 5, and to estimate the persons' velocities and orientations.

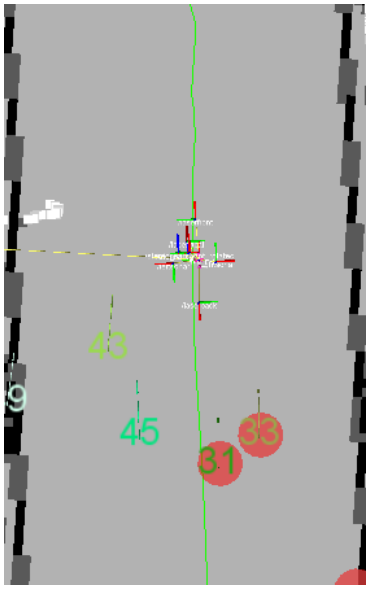


Figure 6: From the list of persons behind or besides the robot, only the candidates that have similar velocity and orientation as the robot are marked as followers (with a circle in the image). In the figure, persons 43 and 45 are overtaking the robot, while 31 and 33 are real followers.

For the particular application of guiding, a third processing layer is considered to filter out the persons that are not being guided by the robot. This layer considers the orientation and velocity of the persons, so that those persons behind or besides the robot moving at a similar velocity and orientation (determined by two thresholds) are marked as followers (see Fig. 6).

The main issue with the laser-based system is that it is not possible to re-identify tracks. We will assume that persons following the robot at its pace are observations indicating that the group being guided is actually following the robot.

4 Modeling the guiding task as a POMDP

In a guiding application, the robot has to guide a person or group of persons towards a common destination. A typical solution to the problem is to plan a path towards the destination and then follow the path by controlling the speed of the robot in a reactive way by using some feedback on the person being guided, like laser, visual information or a combination of them [33].

This requires having an estimation of the position of the person/s. However, imperfect sensors, false positives, occlusions, etc, makes it hard to track robustly a person being guided, and may lead to uncertainties on the person position.

Furthermore, as discussed in Section 2, a pure reactive approach when following the path is not socially adequate. One of the main sources of uncertainty in this problem is that the person may change his/her mind, or decide to stop for a while at a different interest point in his way to the destination. From a social point of view it is important that the robot considers the person goals while guiding the person, in order to wait for the person or even to change its own goal accordingly. However, the robot cannot have a direct observation on this person goal.

Besides that, the actions of the robot can also influence the persons commitment towards the goal, by "pushing" them, or by asking the persons to follow, as the human guide do. Thus, in order to plan the robot actions facing this partially observable variables and uncertainties, reason on the effect of actions and combining different tradeoffs, POMDPs are proposed as model.

4.1 POMDPs in a Nutshell

Formally, a discrete POMDP is defined by the tuple $\langle S, A, Z, T, O, R, D, \gamma \rangle$ [27].

- 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 $z \in Z$.
- After performing an action a, the state transition is modeled by the conditional probability function $T(s', a, s) = p(s'|a, s)$, which indicates the probability of reaching state s' if action a is performed at state s
- the observations are modeled by the conditional probability function $O(z, a, s') = p(z|a, s')$, which gives the probability of getting observation z given that the state is s' and action a is performed.
- The reward obtained for performing action a at state s is $R(s, a)$.

The state is non-observable; at every time instant the robot has only access to observations z which give incomplete information about the state. Thus, a belief function b is maintained by using Bayes rule. The new belief b' obtained if we apply the action a at belief b and get the observation z is given by:

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

The normalization constant:

$$\eta = Pr(z|b, a) = \sum_{s' \in S} O(z, a, s') \sum_{s \in S} T(s', a, s) b(s) \tag{2}$$

gives the probability of obtaining a certain observation z after executing action a for a belief b.

The POMDP model assumes that, at every step, an action is taken, an observation is made and a reward is given. The objective is to determine the actions that maximize the sum of expected rewards, or *value*, earned during D time steps. This actions will depend on the information available, represented by the belief state b. Thus, the objective is to determine the policy $a = \pi(b)$ that maximizes the cumulative reward in time

$$\pi^* = \arg\max_{\pi} E \left[\sum_{t=0}^{D} \gamma^t \sum_{s \in S} b_t(s) R(s, \pi(b)) | b_0 \right] \tag{3}$$

To ensure that the sum is finite when $D \to \infty$, rewards are weighted by a discount factor $\gamma \in [0, 1)$.

The return obtained by following a given policy π from belief b is given by the *value* $V^\pi(b)$:

$$V^\pi(b) = R(b, \pi(b)) + \gamma \sum_{z \in Z} Pr(z|b, \pi(b)) V^\pi(b^z_{\pi(b)})) \tag{4}$$

where $R(b, a) = \sum_s R(s, a) b(s)$ is the expected immediate reward[1].

The value of the optimal policy π^* is usually denoted by $V^*(b)$, and it is the fixed point of the following recursion:

$$V^*(b) = \max_{a \in A} \left[R(b, a) + \gamma \sum_{z \in Z} Pr(z|b, a) V^*(b^z_a) \right] \tag{5}$$

and the associated optimal policy:

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

Figure 7: A topological map of the principal POIs of the FROG tour.

$$\pi^*(b) = \arg\max_{a \in A} \left[R(b,a) + \gamma \sum_{z \in Z} Pr(z|b,a) V^*(b_a^z) \right] \tag{6}$$

The previous recursion is also called a backup operation on b. Associated to the optimal value function is the optimal Q function:

$$Q^*(b,a) = R(b,a) + \gamma \sum_{z \in Z} Pr(z|b,a) V^*(b_{\pi^*(b)}^z) \tag{7}$$

Given this definition, it is clear that $V^*(b) = \max_{a \in A} Q^*(b,a)$.

4.2 The guiding model

In FROG, a predefined tour is considered. Figure 7 shows the tour with the principal POIs considered. At every POI, the robot shows some information and interact with the users. Once a POI is finished, the robot guides the persons towards the next POI in the tour. To guide the persons to the next POI, the global path planner determines a path towards this destination.

The important information to be included in the state space S is the position of the robot r_t and the person p_t within the path. We will also consider as part of the state the intention of the person g_t.

$$s_t = \begin{pmatrix} r_t \\ p_t \\ g_t \end{pmatrix} \tag{8}$$

In order to model the problem by using the POMDP framework described above, the planned path to the next POI is discretized into a set of points with a granularity that can be adjusted. And thus, r_t and p_t are actually the positions of the robot and person within the discretized path. Finally, the person intention g_t is modeled using a binary variable, indicating if the person/s continues engaged in the tour, or has abandoned it.

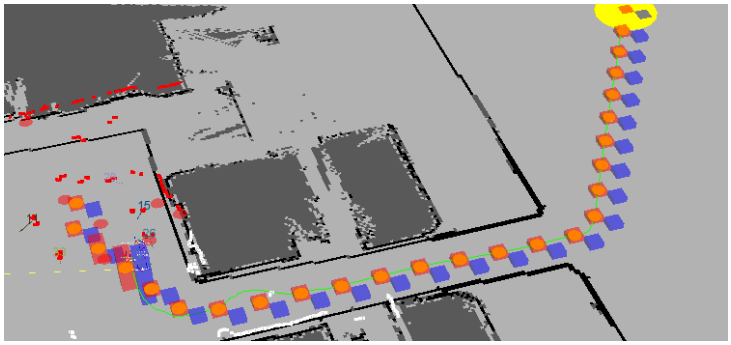


Figure 8: The model employed: the planned path (in green) is discretized. At each point, the bars blue and red indicate the estimated belief on the robot and person positions respectively. Furthermore, the yellow circles indicate the current robot belief on the person goal. The sizes are proportional to the marginal probabilities.

By using its local sensors, and applying the Bayes filter in (1), the robot will maintain a belief state (a probability distribution) over its pose, the person position and the person intention (see Fig. 8), thus reasoning on the person's intention. The belief state is initialized to the initial position of robot and person in the tour, and the person goal/intention is initially set as true (committed towards the end of the path).

The important parts of the model are the observation and prediction functions, described in the next sections.

4.3 Observation model

The robot sensors considered are the output of the localization system of the robot and the sensors used for person tracking.

$$z_t = \begin{pmatrix} z_{l,t} \\ z_{p,t} \end{pmatrix} \tag{9}$$

The localization of the robot $z_{l,t}$ is provided by a map-based Monte-Carlo localization algorithm described in D2.1 [11]. By using this algorithm, the robot can know its position within the path, and thus $z_{l,t} = r_t + \nu_t$, where ν_t models the accuracy of the localization algorithm. The output is discretized according to the discretization of the path.

The accuracy of the algorithm for different parts of the map is known beforehand and can be included into the POMDP observation model. This model assigns some probability of obtaining a robot position measurement in the surroundings of the real position (for instance, in the corridor that can be seen in Fig. 8, the robot may have some uncertainty in its position on the direction of the corridor because of the symmetry of the scenario).

For person guidance, the estimation described in Section 3 is employed. A laser-based person detection and tracking is used. The output of the system is the relative position of the person with respect to the robot $z_{p,t} = r_t - p_t + \epsilon_t$, where ϵ represents the errors from the sensors. This output is discretized according to the discretization of the path. The observation model should also consider the probability of detection when the person is in the field of view of the sensor, while accounting for a small possibility of miss-detections. Furthermore, it can be used to reason on the occlusions due to the static map of the environment.

Both sensors are conditionally independent given the state, and thus, the final observation model is:

$$O(z_t, a_t, s_t) = p(z_t|a_t, s_t) = p(z_{l,t}|r_t)p(z_{p,t}|r_t, p_t)$$

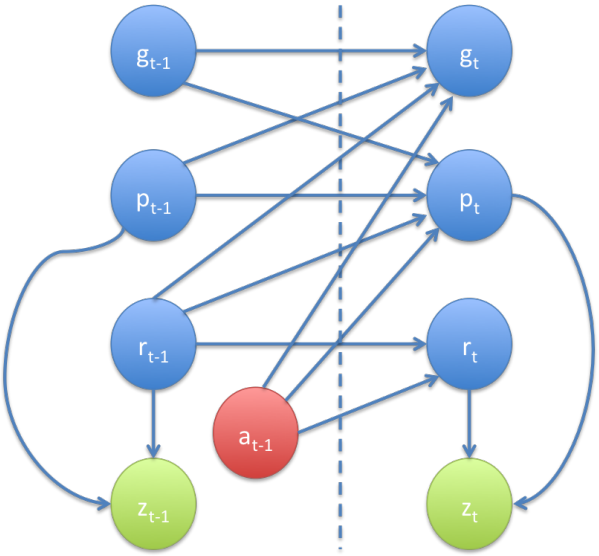


Figure 9: Dynamic Bayesian Network representing the problem.

4.4 Prediction function

The second important aspect is the prediction function, which models the probabilistic evolution of the state according to the actions carried out.

$$T(s_t, a, s_{t-1}) = p(s_t|a_{t-1}, s_{t-1}) = p(r_t, p_t, g_t|r_{t-1}, p_{t-1}, g_{t-1}, a_{t-1})$$

The robot actions a are the velocities controlling the follow of the path. These velocities are discretized. Furthermore, the robot can give voice utterances, asking the persons being guided to "follow the robot". Finally, the robot can abandon itself the task (which should be the case if the persons have abandoned the tour). Fig. 9 shows the Bayesian network that defines the model.

The prediction function depends on different conditionally independent probabilities (see Fig. 9).

$$p(r_t, p_t, g_t|r_{t-1}, p_{t-1}, g_{t-1}, a_{t-1}) = p(r_t|p_t, g_t, r_{t-1}, p_{t-1}, g_{t-1}, a_{t-1},)p(p_t, g_t|r_{t-1}, p_{t-1}, g_{t-1}, a_{t-1})$$

The motion of the robot depends directly on the actions carried out, and do not depend on the rest of the state, $p(r_t|p_t, g_t, r_{t-1}, p_{t-1}, g_{t-1}, a) = p(r_t|r_{t-1}, a)$. The motion of the person is uncertain, though. The person model is separated into the evolution of the person goal, and the motion of the person:

$$p(p_t, g_t|r_{t-1}, p_{t-1}, g_{t-1}, a_{t-1}) = p(p_t|g_t, r_{t-1}, p_{t-1}, g_{t-1}, a_{t-1})p(g_t|r_{t-1}, p_{t-1}, g_{t-1}, a_{t-1})$$

The next position of the person p_t will depend on the intention of the person. The model employed considers that the person will also adapt to the robot pace, and thus it will depend on the robot position and action as well:

$$p(p_t|g_t, r_{t-1}, p_{t-1}, g_{t-1}, a_{t-1}) = p(p_t|r_{t-1}, p_{t-1}, g_{t-1}, a_{t-1})$$

Finally, the conditional probability $p(g_t|r_{t-1}, p_{t-1}, g_{t-1}, a_{t-1})$ models that the actions of the robot can influence the intentions (goal) of the person. For instance, if the robot stays at the same place for a long time the person may be more willing to abandon; also, the other actions of the robot, like asking to be followed, will also affect the person intentions.

From the previous sections, it can be seen that the person goal, g_t, is not directly observable from the sensors. But it will influence the motion of the person, and thus it can be indirectly inferred from the observations available.

4.5 The reward structure

The task to be performed is defined by the reward function $R(s, a)$, which indicate the robot preferences. This reward function will be used to plan the best course of actions. In the particular task of person guidance, the objective is that the person accompanies the robot until the final point in the path.

$$R(s, a) = -\omega_g \|goal - p\| - \omega_d \|r - p\| - \omega_{ann} c_{ann}(a, g) \tag{10}$$

The reward is composed, thus, of several cost functions, combined using different weights. The two first terms penalize that the person is far from the goal and the robot far from the person.

The term $c_{ann}(a, g)$ models the annoyance to the persons given by certain actions of the robot. For instance, if the person is committed, it could be annoying to keep asking the person to follow the robot; or if the person is committed, a very high cost should be incurred if the task is cancelled.

5 Solving POMDPs

5.1 State of the art

The main objective of the guiding module is to solve the previous POMDP model to determine the policy (3) which indicates the actions that the robot should carry out given the current belief state, and that maximizes the expected cumulative reward.

In the last years, POMDPs [27] are more and more often used in robotics [18, 31, 36, 24, 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 to solve POMDPs. 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 algorithms could be applied to just the simplest problems. Several algorithms have been proposed in the last years to deal with larger state, actions and observation spaces [50, 29, 3]. 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 as an alternative for planning under uncertainties [44, 24]. In these online settings, the planner tries, for a certain lookahead planning horizon, to search for the best next action. 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 [44]. In [25], 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.

Here, we consider also forward-search algorithms 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 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 accumulated reward for beliefs that have already been visited and that are in the same depth.

In this deliverable, we consider the use of divergence 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

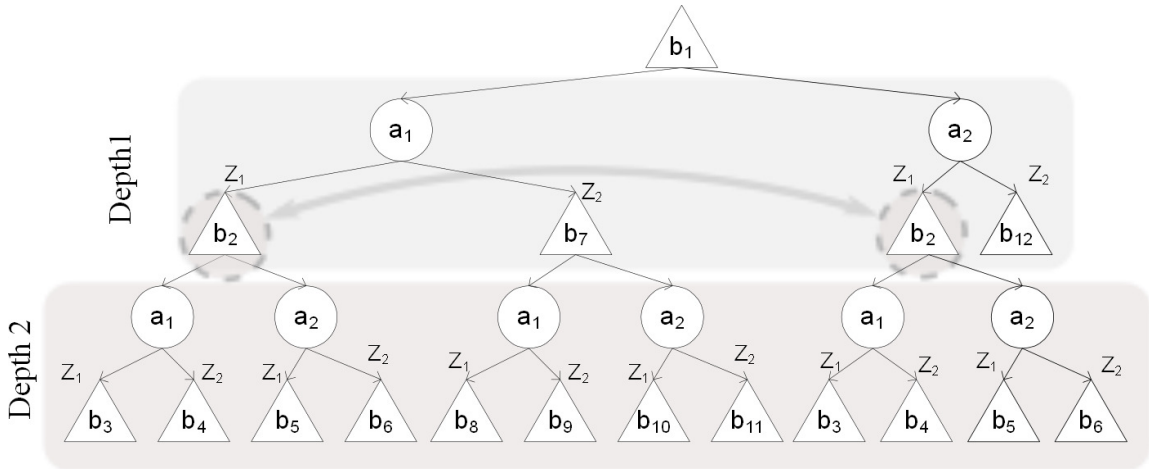


Figure 10: An AND-OR Belief Tree with 2 actions and 2 observations. The OR-node are represented by triangles and the AND-nodes by circle. Notice it is time a belief appear in the same depth it mean that it has identical subtree (like belief b2).

a threshold on these measures and adjusting it, a balance between complexity and optimality can be achieved.

5.2 Online POMDP solvers

Forward-Search based online POMDP 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 and for a planning horizon D (see Fig. 10). The belief nodes are represented by OR nodes, in which an action has to be chosen. The actions are represented as AND nodes, where all potential observations z that can occur has to be considered, each one leading to a new OR belief node.

The value of the root belief node is obtained by propagating the value estimates from the fringe of the tree, using the backup operator (5) at each node. The value V^* on the leaves is typically computed by using an approximate value function computed offline.

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 $(|A||Z|)^D$. Therefore, the number of nodes to be explored grows exponentially with the planning horizon.

In [44], 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.

Here, a new insight is also considered in order to reduce the complexity of online POMDPs, which is the inclusion of topological and metric properties on the belief manifold. The next section presents an initial idea in this sense.

6 The FSBS algorithm

The first algorithm proposed is here is of the Branch and Bound kind, and proceeds via look-ahead search up to a fixed depth d from an initial belief b_0. We use the structure of the RTBSS algorithm

Algorithm 1: FSBS Algorithm

1: **function** FSBS(b, d, δ)
2: **if** $d == 0$ **then**
3: **return** $L(b)$
4: **end if**
5: $\{st_1, st_2, ..., st_{|A|}\} \leftarrow orderbyAccReward(b, d, \delta)$
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: $L_Z(b, a) \leftarrow \sum_{z \in Z} P(z|b, a) FSBS(b_a^z, d - 1, \delta)$
16: $L_T(b, a) \leftarrow Reward(b, a) + \gamma L_Z(b, a)$
17: $saveNode(b, a, d, L_T(b, a))$
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**

proposed by [44] to elaborate the FSBS (see Algorithm 1).

Thus, the algorithm creates a belief tree in a depth-first fashion. At each node of the tree, and upper $U_T(b)$ and lower $L_T(b)$ bound on the optimal value function V^* are maintained. The bounds are propagated to the parent nodes using the backup operator (5):

$$L_T(b) = \begin{cases} L(b) & \text{if } b \in fringe(T) \\ \max_{a \in A} \left[R(b, a) + \gamma \sum_{z \in Z} Pr(z|b, a) L_T(b_a^z) \right] & \text{otherwise} \end{cases} \tag{11}$$

$$U_T(b) = \begin{cases} U(b) & \text{if } b \in fringe(T) \\ \max_{a \in A} \left[R(b, a) + \gamma \sum_{z \in Z} Pr(z|b, a) U_T(b_a^z) \right] & \text{otherwise} \end{cases} \tag{12}$$

where $fringe(T)$ is composed of the leaves of the tree. The algorithm uses the max-planes lower bound $L(b)$ [48] of the optimal value implemented in [49] (line 3). The recursive algorithm is called with the threshold δ used to determine if two belief points are equal.

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 (according to the similarity threshold δ), that belief will not be expanded because its value is already calculated (see Fig. 10). The similarity between beliefs is computed by using one of the divergence measures described in Section 6.1.

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 lower bound on the accumulated reward if that action is applied to the belief, $L_T(b)$. 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 δ. If a belief is successfully found, the accumulated reward that was already stored, $L_T(b')$, is assigned to it, and if not, it is assigned as ∞ in order to 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:

$$\pi^*(b_0, D) = \arg\max_{a \in A}(R(b_0, a) + \gamma \sum_{z \in Z} P(z|b, a) FSBS(b^z_{0a}, D-1, \delta))$$

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.

6.1 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. Here, we will analyze three measures from the field of information theory; namely, the Bhattacharyya distance, D_B (13); the Jensen-Shannon (JS) divergence D_{JS} (16), which uses the Kullback-Leibler divergence D_{KL} (15); and finally, the the Rényi divergence, a generalization of relative entropy; in particular the divergence of order 2, D_{R2} (14). Given two discrete beliefs (probability distributions) $p(s)$ and $q(s)$, they are defined as:

$$D_B(p \parallel q) = -\ln\left(\sum_{s \in S} \sqrt{p(s)q(s)}\right) \tag{13}$$

$$D_{R2}(p \parallel q) = \log E[\frac{p}{q}] = \log \sum_{s \in S} p(s) \frac{p(s)}{q(s)} \tag{14}$$

$$D_{\mathrm{KL}}(p \| q) = \sum_{s \in S} p(s) \ln \frac{p(s)}{q(s)} \tag{15}$$

$$D_{JS}(p \| q) = \frac{1}{2}(D_{KL}(p \| \frac{p+q}{2}) + D_{KL}(q \| \frac{p+q}{2})) \tag{16}$$

These three statistical divergences have been used in many areas. Rényi divergence has been used in domain adaptation [32]. The Jensen-Shannon divergence has been used in imaging processing [21] and active learning settings [2]; also, the Bhattacharyya distance has been used in image processing for histogram comparison [6].

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 Rényi 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 [30] 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.

Table 1: Comparative between statistical distance

Distance	Complexity	Always defined	Range
Bhattacharyya	Medium	Yes	$0 \leq D_B \leq \infty$
Rénji 2	Low	No	$0 \leq D_{R2} \leq \infty$
Jensen Shannon	Hight	Yes	$0 \leq D_{JS} \leq 1$

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 [15], and thus we will focus the analysis on the JS divergence.

6.2 Analysis using benchmarks

In order to evaluate the FSBS algorithm, we have implemented RTBSS [38] as baseline algorithm. We have used Trey Smith's library [49] for the implementations. Moreover, we have adapted the library so that it can be used in ROS (Robotic Operating System) [42]. We have used the POMDP parser and the structure provided by [49] 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 [47] and Tag [40]. 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 6.1 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 [49]) 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[2].

6.2.1 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. 11 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

[2]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 [44]. However, the conclusions are not affected by the particular implementation, as we are interested in the relative gain obtained by incorporating the divergence measures.

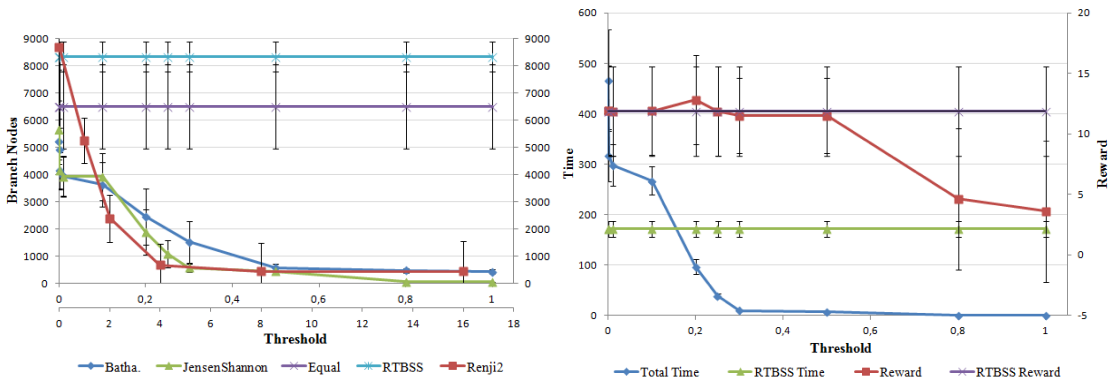
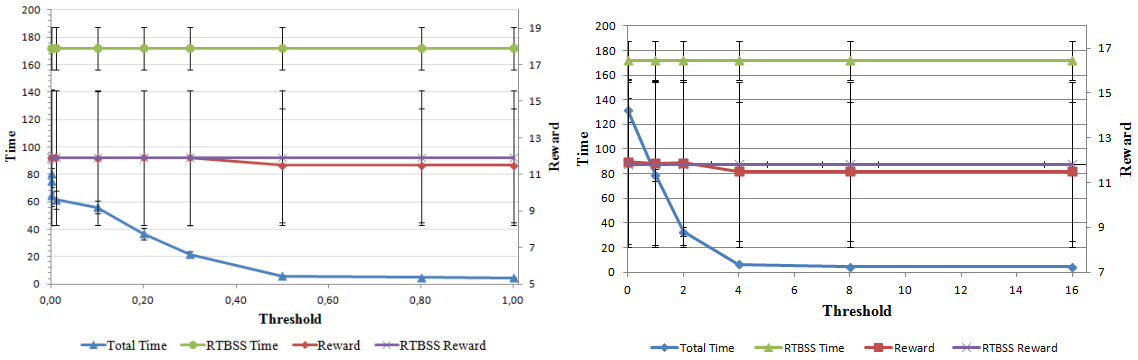


Figure 11: Left: Number of branch nodes (RockSample). The graphic shows the mean number of nodes and its variance for different divergence measures and thresholds. The RTBSS case and the equal case are shown as well. Right: Expected Reward (right vertical axis) and execution time (left vertical axis) for RTBSS and FSBS using the JS divergence (RockSample)



(a) (b)

Figure 12: (a)Expected Reward (right axis) and execution time (left axis) for RTBSS and for FSBS using the Bhattacharyya distance (RockSample). (b) Expected Reward (right axis) and execution time (left axis) for RTBSS and for FSBS using the Rényi's divergence of order 2 (RockSample)

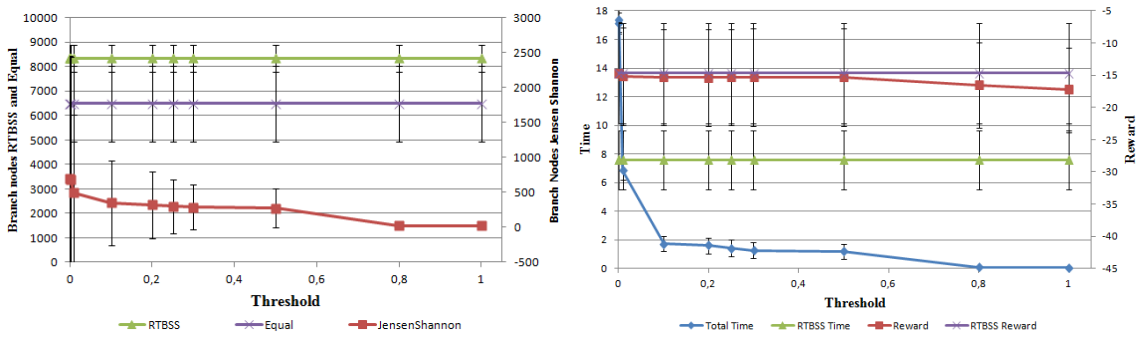
prune because it is not defined in case q_i is equal to 0. It is important, moreover, to notice that even in the case of the removal of equal nodes, a significative reduction of nodes is achieved with respect to the original RTBSS.

More importantly, we also compare the expected reward to check how the reduction of nodes affects the performance of the planner. In Fig. 11, right, we can see the comparison of 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. 12 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. 11).

Regarding RockSample we can finally see in Fig. 12 the 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



(a) (b)

Figure 13: (a) 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). (b) Expected Reward (right axis) and execution time (left axis) for RTBSS and for FSBS using the JS divergence (Tag).

Rényi), it can be seen that the Bhattacharyya gets the lowest execution time.

6.2.2 Tag Benchmark

In the case of the Tag benchmark [40] 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. 13 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. 13, right, we can observe the comparison of the expected cumulative reward for FSBS using the JS divergence and 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.

6.3 Approximation analysis

In this section we analyze the error committed by our approach. Basically we need to characterize the error on the computed value function for the root node in, given by:

$$\hat{V}(b_0 = \max_{a \in A}(R(b_0, a) + \gamma \sum_{z \in Z} P(z|b, a) FSBS(b^z_{0a}, D-1, \delta))$$

There are several theorems that can lead us to this analysis:

First, the following theorem bounds the error committed when performing a full look-ahead search and using an approximate value function $\hat{V}$ at the leaves [44, 23]:

Theorem 1. *Let $\hat{V}$ be an approximate value function and $\epsilon = \sup_b |V^*(b) - \hat{V}(b)|$. Then the approximate value $\hat{V}^D(b)$ returned by a D-step lookahead from belief b, using $\hat{V}$ to estimate fringe node values, has error bounded by $|V^*(b) - \hat{V}^D(b)| \leq \epsilon\gamma^D$.*

For $\gamma \in [0, 1)$ it means that the estimated value at the root is improved given any approximation to the value function (like the lower bound employed above). The deeper the search the lower the error committed.

This theorem can be applied to the original RTBSS case. In FSBS, during the lookahead full subtrees are approximated by the values of other subtrees whose root nodes are similar (considering the similarity measures, see Fig. 10) and which have been previously explored.

Therefore, it is important to characterize which error is committed when we use a similar belief point instead of the actual one in FSBS. The following lemma [26] determines the error committed when using the value function computed at b to compute the value at b':

Lemma 1. *For any two belief points b and b', $|V^*(b') - V^*(b)| \leq \frac{R_{max}}{1-\gamma}||b' - b||_1$*

Furthermore, the following lemma relates the L1-norm with the Jensen-Shannon distance [54]:

Lemma 2. *Given to belief points b and b', $\frac{1}{2}||b' - b||_1 \leq \sqrt{D_{JS}(b, b')} \leq \sqrt{\log 2||b' - b||_1}$*

Then, we are now in condition to determine the error we commit when we substitute the optimal value function at a belief point b', $V^*(b')$ by the FSBS estimation at a different belief point b, $\hat{V}(b)$, which is closer to b' (in the JS sense) than a threshold δ. This error is denoted by $|V^*(b') - \hat{V}(b)|$.

Theorem 2. *Let $\hat{V}$ be an approximate value function and $\epsilon = \sup_b |V^*(b) - \hat{V}(b)|$. The error committed by the FSBS algorithm for a lookahead D and considering a threshold δ to determine that two belief points are equal is bounded by:*

$$|V^*(b') - \hat{V}^D_{FSBS}(b)| \leq \frac{(1-\gamma^{D+1})R_{max}}{(1-\gamma)^2} 2\sqrt{\delta} + \gamma^D \epsilon \tag{17}$$

Proof. If at a given depth, we substitute the optimal value $V^*(b')$ at the root of a subtree b' by the estimated value by lookahead search at a different node of the same level b, $\hat{V}(b)$, we have:

$$\begin{aligned}
|V^*(b') - \hat{V}^D(b)| &= |V^*(b') - \hat{V}^D(b) + V^*(b) - V^*(b)| \leq \quad \text{add 0} \\
&\leq |V^*(b') - V^*(b)| + |V^*(b) - \hat{V}^D(b)| \leq \quad \text{Triangular inequality} \\
&\leq \frac{R_{max}}{1-\gamma}||b' - b||_1 + |V^*(b) - \hat{V}^D(b)| \leq \quad \text{Lemma 1} \\
&\leq \frac{R_{max}}{1-\gamma}||b' - b||_1 + \epsilon\gamma^D \leq \quad \text{Theorem 1} \\
&\leq \frac{R_{max}}{1-\gamma} 2\sqrt{D_{JS}(b', b)} + \epsilon\gamma^D \leq \quad \text{Lemma 2} \\
&\leq \frac{R_{max}}{1-\gamma} 2\sqrt{\delta} + \epsilon\gamma^D \quad \text{Maximum distance (threshold)}
\end{aligned}$$

where ϵ is the error on the approximation used in the lookahead from b and D the depth of the subtree rooted at b.

This should be iterated, because this error ϵ may depend on the approximations used in this subtree due to the JS distance-based pruning. Therefore, for the leave nodes ($D = 0$), we have that the error is bounded by:

$$\phi_0 = \frac{R_{max}}{1-\gamma} 2\sqrt{\delta} + \epsilon$$

At the next parent level, we have:

$$\phi_1 = \frac{R_{max}}{1-\gamma} 2\sqrt{\delta} + \gamma\phi_0 = \frac{R_{max}}{1-\gamma} 2\sqrt{\delta} + \gamma(\frac{R_{max}}{1-\gamma} 2\sqrt{\delta} + \epsilon)$$

At the following level:

$$\phi_2 = \frac{R_{max}}{1-\gamma} 2\sqrt{\delta} + \gamma\frac{R_{max}}{1-\gamma} 2\sqrt{\delta} + \gamma^2(\frac{R_{max}}{1-\gamma} 2\sqrt{\delta} + \epsilon)$$

At the root node:

$$\phi_D = \frac{R_{max}}{1-\gamma} 2\sqrt{\delta}(1+\gamma+\gamma^2+\ldots+\gamma^D) + \gamma^D\epsilon = \frac{(1-\gamma^{D+1})R_{max}}{(1-\gamma)^2} 2\sqrt{\delta} + \gamma^D\epsilon$$

But $\phi_D = |V^*(b') - \hat{V}^D_{FSBS}(b)|$ is the error at the root node, which is the one we are looking:

$$|V^*(b') - \hat{V}(b)| \leq \frac{(1-\gamma^{D+1})R_{max}}{(1-\gamma)^2} 2\sqrt{\delta} + \gamma^D\epsilon \tag{18}$$

□

7 Graph-based Online POMDP (GO-POMDP)

The goal of previous section was to show that introducing similarity measurements between belief points in online POMDP algorithms reduces the complexity and minimizes the number of nodes used. We have employed a branch and bound algorithm as baseline because 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, like AEMS2 [44], can obtain much better policies. In heuristic search algorithms, the belief tree is not completely explored for a fixed depth. Instead, the fringe nodes $b \in fringe(T)$ are expanded according to a heuristic function $H_T(b)$ that indicates how relevant is a particular node. At each iteration, the fringe node that maximizes the heuristic is expanded. Then, the values of its ancestors are updated, and the heuristics are recomputed.

The heuristic computation implies an additional cost and the heuristic value 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 in general improves the final result.

In this section we present the final POMDP technique developed. This technique extends heuristic search by introducing the similarity measures within the belief tree. By searching similar nodes all over the tree, and not only at each level of the tree, the final structure is a graph, in which actions can lead to revisit existing belief nodes. This structure reflects more accurately the real topology of the belief manifold (see Fig. 14); by reusing visited nodes we expect to combine the strength of online and offline POMDPs.

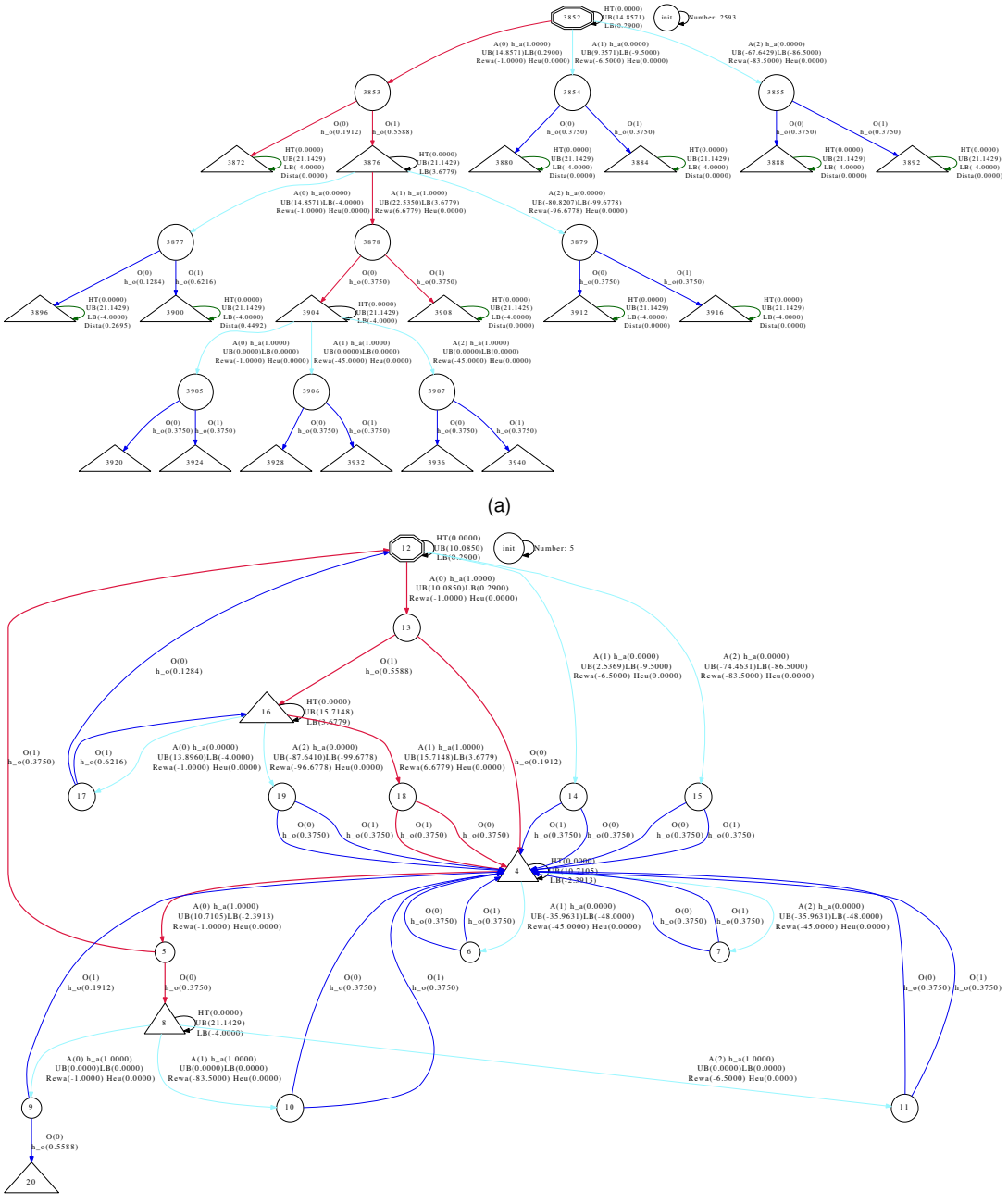
The main issues are the implementation of the two main steps on the POMDP solution using a graph:

- the online building of the graph: that is, how the graph should be expanded at every iteration;
- and the update of the value functions. Now, the backup equation (5) has to be iterated, as there may be loops in the graph

In order to cope with the graph structure, we will extend and modify Iterative-Boundling LAO* [56], a variant of LAO* [22], which is a technique for solving heuristic search problems, and in particular MDPs, in graphs. We apply those algorithms to the problem at hand, our graph-based POMDP. These techniques can be applied as a POMDP can be seen as a MDP over the belief states.

7.1 LAO* and Iterative Bounding LAO*

As commented above, the main difference of a graph version with respect to the tree lies on the solving procedure. While in a tree the approximate value on the leaves is propagated towards the root through a backwards induction (11), in the case of a graph, as there can be cycles, the propagation of the value involves a dynamic programming step by means of value iteration.



(a)

(b)

Figure 14: (a) Tree resulting of a heuristic search . The octogonal shape represents the initial belief. The triangles represent OR nodes (where an action should be taken) and the circles AND nodes (contingent nodes where observations occur). (b) The resulting graph. It can be see how many of the OR nodes are reused, loops occur, etc. In red it is marked the current best solution graph.

The POMDP AND/OR graph can be also represented as an hypergraph, in which the nodes of the hypergraph are the OR nodes of the AND/OR graph. To each action can be associated an hyperedge that links this node to all the OR nodes that can be reached by applying that action (they can be several depending on the observations at the AND node).

We need some definitions:

Definition 1. *The POMDP* implicit *hypergraph G can be defined as follows:*

- *the initial belief b_0 is part of the graph.*
- *For all the actions a that can be applied to a belief point b in the graph, the successors of b are $\tau(b, a, z), \forall z$. All these successors are part also of the graph G (for each action a hyperedge there will be several successors due to the several potential observations).*
- *The leaves of G consists of absorbing belief states, that is belief states in which a termination action is selected, or in which every action leads to the same belief node.*

Notice that absorbing belief states may not exist in general, and the POMDP graph could grow indefinitely or be closed (with no leaves).

For an implicit (hyper-)graph G we can define a solution graph:

Definition 2. *A solution graph G_π of the graph G is a subgraph $G_\pi \subseteq G$ in which*

- *The initial belief point b_0 is part of the solution graph, $b_0 \in G_\pi$*
- *For any non-leaf node b in G_π there is exactly just one hyperedge corresponding to one action a, and all the belief successors $b_a^z = \tau(b, a, z), \forall z$ of that action are part of G_π*
- *Only the leaves in G can be leaves in G_π*

A solution graph defines a policy $\pi(b)$ which indicates the action that should be taken in any belief point reachable from b_0. There are many possible solution graphs. They could be ranked according to the value function V^π, (19), associated to the policy defined by the graph. The best solution graph will correspond to the optimal value function V^* and optimal policy π^*.

The objective is to find a solution graph without having to explore the full *implicit* graph G. LAO* and Iterative-Bounding LAO proceed by building incrementally an *explicit* subgraph $G' \subseteq G$ that initially consists only of the initial belief point b_0.

Let G'_π be a solution graph of this subgraph G', built by selecting at each node b the hyperedge corresponding to action a that

$$a = \pi(b) = \arg\max_{a \in A} [R(b, a) + \gamma \sum_{z \in Z} P(z|b, a) V(b_a^z))] \tag{19}$$

If all leaves of G'_π are leaves of G (absorbing beliefs) or G'_π is closed (it has no leaves), then G'_π is also a solution of G. If the leaves of G'_π are non-terminal, then G'_π is called a *partial solution graph* (leading to a partial policy). LAO expands the current explicit graph G' by indicating actions for the fringe of the current partial solution G'_π, thus incrementally expanding it until a solution of G' is also a solution of G.

Definition 3. *The set $fringe(G'_\pi)$ consists of all the leaf nodes of the graph G'_π which are not absorbing nodes.*

The LAO* algorithm is summarized in Algorithm 2. As it can be seen, in line 4 the algorithm performs value iteration on the ancestors belief states of the expanded node in the partial solution graph G'_π until convergence. Then, in step 6, value iteration is performed in the full partial solution graph G'_π until convergence or the partial solution graph changes and a new leave appears.

Our algorithm is based on Iterative Bounding LAO* [56], which is a variant of LAO* more suitable for real-time performance, and that can be applied to MDPs (and POMDPs in this case).

In the following, the modification and specifies for our algorithm are described.

Algorithm 2: LAO* Algorithm

1: $G' \leftarrow \{b_0\}$
2: **while** $fringe(G'_\pi) \neq \emptyset$ **do**
3: Expand one belief node in $fringe(G'_\pi)$ and add the resulting node to G'
4: Perform value iteration on all the ancestor nodes of the newly expanded node according to the marked actions on the best solution graph G'_π.
5: **end while**
6: Perform value iteration on all the nodes of G'_π
7: **if** $fringe(G'_\pi) \neq \emptyset$ **then**
8: Go to step 2
9: **end if**
10: **return** G'_π

Algorithm 3: GO-POMDP Algorithm

1: $G' \leftarrow \{b_0\}$
2: **while** !timeout **do**
3: **while** $\epsilon(b_0) > \epsilon$ and !timeout **do**
4: **if** $fringe(G'_\pi) \neq \emptyset$ **then**
5: Select $b \in fringe(G'_\pi)$ to expand
6: EXPAND(b,δ)
7: UPDATE(b)
8: **else**
9: Perform a backup operation on all nodes of G'_π.
10: **end if**
11: **end while**
12: **end while**
13: **return** G'_π

7.2 GO-POMDP

The GO-POMDP algorithm builds the graph online, while at the same time solving for the optimal action. As in FSBS (see Section 6), at each belief node a lower $L_G(b)$ and upper $U_G(b)$ bounds on the optimal value function $L_G(b) \leq V^*(b) \leq U_G(b)$ are maintained. These bounds will be used to prune search branches when the error bound on the optimal solution value is below a certain threshold. Then, the algorithm proceeds until the error bound on the root is below a given threshold, or a timeout is reached.

The stop condition is met when the difference between bounds $\epsilon(b_0) = U_G(b_0) - L_G(b_0)$ is higher than a predefined value ϵ; or when $L_{b_0} \geq U_G(b_0, a) \forall a \neq \arg\max_{a \in A} L_G(b_0, a)$, that is, when an action dominates all the others at the root node.

Algorithm 3 summarizes the procedure. Two steps are interleaved, expansion and backup of the ancestors. During backup, and differently from LAO, no value iteration is performed, but just one backup operation per ancestor.

An important issue is how to determine the nodes to expand from the fringe. A good heuristic is key to perform an intelligent search for the nodes.

7.2.1 Update and partial solution selection

The update step in GOPOMDP is described in Algorithm 4. During this update step, the bounds on all the ancestors of the new belief are updated using the backup operation:

$$L_G(b) = \max \left[L_G(b), \max_{a \in A} \left[R(b,a) + \gamma \sum_{z \in Z} Pr(z|b,a) L_G(b_a^z) \right] \right]$$

Algorithm 4: GO-POMDP: Update Function

function UPDATE(b')
  while $b' \neq b_0$ and not visited b' **do**
    Select the ancestor $b \in G'_\pi$ of the node b' according to the current best policy (b' is reached by applying the action a_π at b)
    $L_G(b) = \max\left[L_G(b), \max_{a \in A}\left[R(b,a) + \gamma \sum_{z \in Z} Pr(z|b,a) L_G(\tau(b,a,z))\right]\right]$
    $U_G(b) = \min\left[U_G(b), \max_{a \in A}\left[R(b,a) + \gamma \sum_{z \in Z} Pr(z|b,a) U_G(\tau(b,a,z))\right]\right]$
    $a_\pi = \arg\max_{a \in A}\left[R(b,a) + \gamma \sum_{z \in Z} Pr(z|b,a) U_G(\tau(b,a,z))\right]$
    $H^*_G(b) = \max_{z \in Z} \gamma p(z|b,a_\pi) H_G(\tau(b,a_\pi,z))$
    $z^* = \arg\max_{z \in Z} \gamma p(z|b,a_\pi) H_G(\tau(b,a_\pi,z))$
    $b^*_G(b) = b^*_G(\tau(b,a_\pi,z^*))$
    $b' \leftarrow b$
  end while
  return G'_π
end function

Algorithm 5: GO-POMDP: Expand Function

function EXPAND(b,δ)
  for all $a \in A$ **do**
    for all $z \in Z$ **do**
      $b' \leftarrow \tau(b,a,z)$
      if !SEARCH(b',G',δ) **then**
        $L_T(b') = L(b')$
        $U_T(b') = U(b')$
        $H^*_T(b') = U(b') - L(b')$
        $b^*_T(b') = b'$
        Add b' to G'
      end if
    end for
  end for
  return G'
end function

$$U_G(b) = \min\left[U_G(b), \max_{a \in A}\left[R(b,a) + \gamma \sum_{z \in Z} Pr(z|b,a) U_G(b^z_a)\right]\right]$$

During this operation, the partial solution graph G'_π is also updated; that is, the actions marking the current solution. The partial solution graph is performed over the upper bound policy, so that

$$a_\pi = \arg\max_{a \in A}[R(b,a) + \gamma \sum_{z \in Z} Pr(z|b,a) U_G(b^z_a))] \tag{20}$$

This step can lead to a different best solution graph G'_π, which is indicated by the actions a_π at each node. The current best partial solution graph is indicated by the red edges in Figs. 14 and 15.

7.2.2 Expansion

The expansion of the selected node consists simply on generating all the successor nodes for all the potential actions and observations. For each successor node b', first the current graph G' is searched to see if a there is a similar node, closer than δ. In that case, the hyper edges of the graph are adjusted accordingly. If there is none, for the newly expanded belief b' the upper and lower bounds are initialized using precomputed bounds. Algorithm 5 describes the main steps.

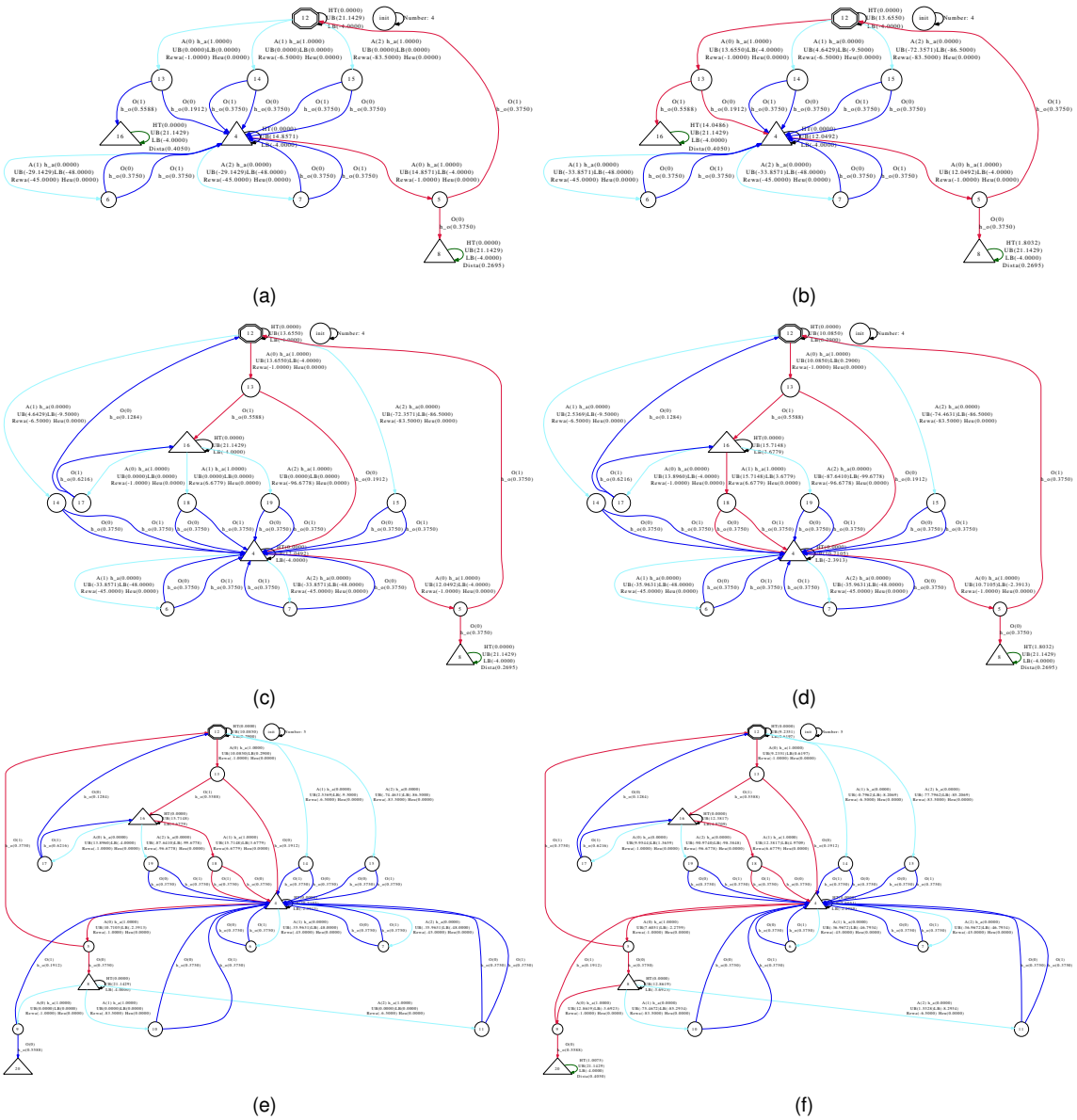


Figure 15: Left: expansion phase. Right: update phase. (a) The graph after an expansion. (b) The bounds, as well as the heuristic values H_T, are updated. Also, the new best action graph is obtained. (c) Node 16, with the highest heuristic value, is expanded. (d) Again, the bounds and heuristics are updated. (e) Node 8 is expanded. (f) The values are updated. The current best solution graph indicates a policy (which action should be taken at each node).

The key issue on the expansion is how to select the next node from the fringe to be expanded. The intuition behind typical approaches is to expand those nodes that can contribute most to the error of the currently estimated optimal value function at the root node b_0.

According to [43], for a tree-based heuristic search, the error contribution of a fringe node on the error committed at the root estimation is given by:

$$e_G(b, b_0) = \gamma^{d(H_{\pi^*}(b,b_0))} p(H_{\pi^*}(b, b_0)|b_0, \pi_*)(V^*(b) - L_G(b)) \tag{21}$$

where $H_{\pi^*}(b, b_0)$ is a path between b_0 and b following the optimal policy π^* and, by a slight abuse of notation, $p(H_{\pi^*}(b, b_0)|b_0, \pi_*)$ is the probability of that path given the policy (that is, the probability of getting the observations z that lead to that particular path between the root belief point, b_0, and b).

As the optimal value and optimal policy is not known, the idea proposed in [44] is to approximate $(V^*(b) - L_G(b))$ by $(U_G(b) - L_G(b))$, and also the optimal policy by the currently estimated policy using the upper-bound. Then, this can be computed by propagating up the heuristic value through the current policy. This heuristic is called AEMS2.

In the case of a graph, the computation of the error at the root is more complicated, as there can be several paths between the root and a particular fringe node. So the total error is the sum for all the potential paths between b_0 and b, $H_G(b, b_0)$.

$$e_G(b, b_0) = \sum_{h \in H_G(b,b_0)} \gamma^{d(h)} p(h(b, b_0)|b_0, \pi_G)(U_G(b) - L_G(b)) \tag{22}$$

In our approach, we propagate the heuristic value using the current partial solution graph (and policy) and not revisiting any node. Then, for each node in the graph, a heuristic value is maintained $H^*_G(b)$, which is the contribution of that particular node in the path to the fringe node:

$$H^*_G(b) = \begin{cases} U_G(b) - L_G(b) & \text{if } b \in fringe(G') \\ \max_{z \in Z} \gamma Pr(z|b, a_\pi) H_G(\tau(b, a_\pi, z)) & \text{otherwise} \end{cases}$$

Also, at each node of the graph a reference to the best node to be expanded $b^*_G(b)$ is maintained and updated if needed:

$$b^*_G(b) = \begin{cases} b & \text{if } b \in fringe(G') \\ b^*_G(\tau(b, a_\pi, z^*)) & \text{otherwise} \end{cases}$$

where

$$z^* = \arg\max_{z \in Z} \gamma Pr(z|b, a_\pi) H_G(\tau(b, a_\pi, z))$$

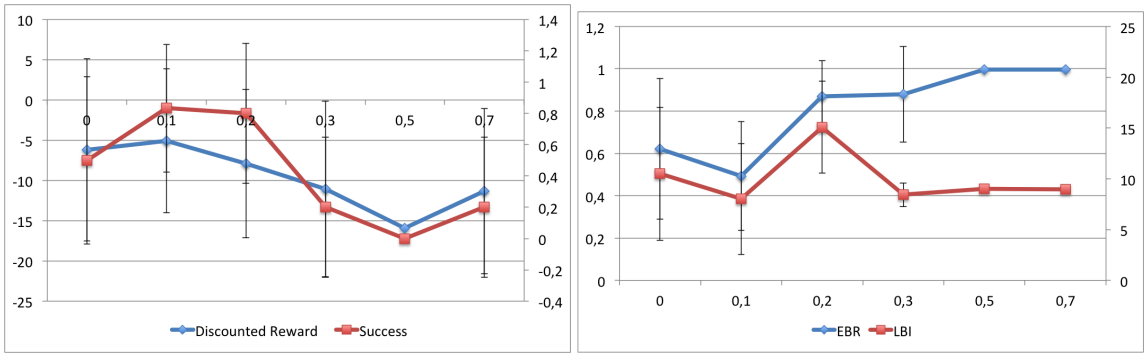
These computations are performed during the backup operations as before.

When a new expansion is to be performed, the reference to the best node to expand at the root, $b^*_G(b_0)$ is selected and expanded following Algorithm 5.

7.3 Approximation analysis

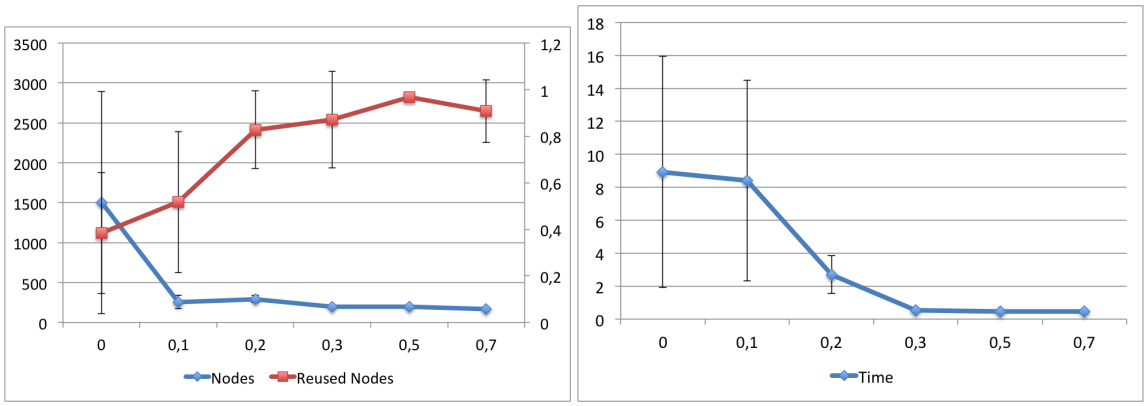
In GOPOMDP we should also estimate the error committed at the root node b_0 due to the approximation bounds employed and the threshold used in the distance to determine if two beliefs are similar (this distance will influence the structure of the graph).

Two analysis lead to the same result. If the graph is closed, then at every point we perform value iteration until convergence (or until a timeout). This situation is the same as in point-based offline solvers, which select only a discrete set of beliefs to determine the optimal value function.



(a) (b)

Figure 16: (a)Expected Reward and success rate (right axis) for the Tag benchmark as a function of the similarity threshold. (b) EBR and LBI (right axis)



(a) (b)

Figure 17: (a)Mean number of nodes expanded and percentage of reused nodes (right axis) between cycles. Mean total execution time for the Tag mission.

In particular, following the same argument as in [41] for PBVI, it can be seen that in this case, and for a maximum distance threshold of δ between any belief point and the beliefs in the graph, the error with respect to the optimal value function is given by:

Theorem 3. *For a given distance threshold δ, the error on the estimated optimal value function $\hat{V}$ for each node of the graph committed by the GO-POMDP algorithm is bounded by:*

$$|V^*(b) - \hat{V}(b)| \leq \frac{R_{max}}{(1-\gamma)^2} 2\sqrt{\delta} \tag{23}$$

The same conclusion can be reached if we considered a closed graph as a version of a lookahead tree with infinite depth. In that case, from Theorem 2, and setting $D \to \infty$, we arrive at the same result.

7.4 Benchmarking

We have also evaluated the approach by using the Tag benchmark [40] against the AEMS2 heuristic search [44] for a fixed time at each planning cycle of 0.5 seconds. We use the JS divergence in this case. We have executed the Tag problem 5 times at each configuration.

Figure 16 shows the expected reward obtained by different threshold levels. The threshold $\delta = 0$ corresponds to the AEMS2 algorithm. In this particular case, we do not perform the search (that is, this algorithm does not incur on the additional cost of searching in the graph).

It can be seen how the expected reward degrades with the distance threshold employed, as expected from the analysis above. However, this degradation is low for small thresholds.

Following [44], we also computed the following comparison metrics. The Error Bound Reduction (EBR), defined as:

$$EBR(b_0) = 1 - \frac{U_G(b_0) - L_G(b_0)}{U(b_0) - L(b_0)} \tag{24}$$

which indicates how much the algorithm is able to reduce the initial precomputed bounds. Furthermore, the Lower Bound Improvement (LBI) is computed:

$$LBI(b_0) = L_G(b_0) - L(b_0) \tag{25}$$

which indicates the true error reduction. In both cases, the higher the value the better. It can be seen that the larger the threshold the better the EBR. This indicates that the larger the threshold the algorithm is able to converge (EBR close to 1) by interating over a coarser graph. However, the LBI in these cases do not improve due to the larger error committed.

Figure 17 shows the mean number of nodes expanded, as well as the percentage of reused nodes. It can be seen how even for low thresholds, like 0.1, the number of expanded nodes is much lower than in the heuristic search. However, the cost of searching similar nodes compensates the time saved by searching through less nodes. For larger thresholds the time improvement can be nearly 10-fold, and keeping a similar expected reward outcome. By observing both figures, it can be seen how a tradeoff can be reached between quality on the solution and time. Furthermore, new algorithm enhancements would allow to reduce the computational times, like using kd-trees to ease the search between points in the graph.

8 Results

8.1 Simulation results

In order to analyze the modeling of the task by using POMDPs, in this section we present some results obtained by applying the mentioned algorithm in simulations.

8.1.1 Model

For the simulations, we consider the following particular implementations of the models of Section 4.

We consider the task of guiding a person between 2 points. The path between these two points is of 54 meters. The trajectory selected is discretized with a 2-meter resolution, giving 27 potential values (the discretization used in space and time is dependent on the velocity of the robot and the planning horizon).

As commented above, the state consists of the position of the robot r and the person p within the path, as well as the hidden commitment of the person g, which can have 2 potential states, giving thus an state space of dimension $|S| = 1458$ ($27 \times 27 \times 2$).

The observations are composed by the estimated position of the robot within the path, and the observations from the lasers (9). The latter is discretized into two possible values (if the persons are detected behind -up to 5 meters- or at the same position of the robot). Thus, the number of potential observations are $|O| = 54$.

The observation model is characterized by the localization and person detection functions. The localization system is able to locate the robot with a certain error. As a general model, we will consider that the localization system is able to provide with a position in the vicinity of the real discretized location, and thus:

$$p(z_{l,t}|r_t) = \begin{cases} pL & \text{if } z_{l,t} = r_t \\ 1 - \frac{pL}{2} & \text{if } z_{l,t} = r_t - 1 \\ 1 - \frac{pL}{2} & \text{if } z_{l,t} = r_t + 1 \\ 0.0 & \text{otherwise} \end{cases}$$

In the real FROG robot, the localization system has a mean error below half a meter, so it would imply that the position is directly observable, and $pL = 1.0$.

The detection module can be characterized by its probability of detection when the persons are behind the robot (pD), and the probability of classifying persons as followers when they are not (pF).

$$p(z_{p,t} = 1|r_t, p_t) = \begin{cases} pD & \text{if } 0 \leq r_t - p_t \leq 2 \\ pF & \text{otherwise} \end{cases}$$

The actions available are moving forward ($a = 0$), waiting ($a = 1$), the same two options but including a voice message asking to follow the robot ($a = \{2, 3\}$)and a cancel action to abort the tour ($a = 4$). Thus the action space size is $|A| = 5$. The motion of the robot directly depends on these actions:

$$p(r_t|r_{t-1}, a) = \begin{cases} pM & \text{if } r_t = r_{t-1} + 1 \text{ and } a_t = \{0, 2\} \\ 1 - pM & \text{if } r_t = r_{t-1} + 1 \text{ and } a_t = \{0, 2\} \\ 1.0 & \text{if } r_t = r_{t-1} \text{ and } a_t = \{1, 3, 4\} \\ 0.0 & \text{otherwise} \end{cases}$$

which basically states that the robot will move to the next point in the path if commanded so. The probability pM can be used to model temporal blockages of the robot due to crowds or due to avoidance maneuvers.

The motion of the person will be influenced by its goal and the motion of the robot. Thus, if the person is committed to follow the tour ($g = 1$) and the robot moves forward $a_{t-1} = \{0, 2\}$ we assume that the person will move also forward with a certain probability pT if it is behind the robot (this probability is used to model the possibility that the person can stop to take photos or similar), or will wait if it is ahead of the robot.

$$p(p_t|r_{t-1}, p_{t-1}, g_{t-1} = 1, a_{t-1} = \{0, 2\}) = \begin{cases} pT & \text{if } p_t = p_{t-1} + 1 \text{ and } p_{t-1} \leq r_{t-1} \\ 1 - pT & \text{if } p_t = p_{t-1} \text{ and } p_{t-1} \leq r_{t-1} \\ 1.0 & \text{if } p_t = p_{t-1} \text{ and } p_{t-1} > r_{t-1} \\ 0.0 & \text{otherwise} \end{cases}$$

If the robot stops, we assume that the person will continue forward until it is close to the robot:

$$p(p_t|r_{t-1}, p_{t-1}, g_{t-1} = 1, a_{t-1} = \{1, 3\}) = \begin{cases} pT & \text{if } p_t = p_{t-1} + 1 \text{ and } p_{t-1} \leq r_{t-1} - 2 \\ 1 - pT & \text{if } p_t = p_{t-1} \text{ and } p_{t-1} \leq r_{t-1} - 2 \\ 1.0 & \text{if } p_t = p_{t-1} \text{ and } p_{t-1} > r_{t-1} - 2 \\ 0.0 & \text{otherwise} \end{cases}$$

The probability pT will be higher if the robot uses its voice to ask to be followed, as we assume that this will push the person to continue.

If the person is not committed to follow the tour ($g = 0$), we will assume that the person will stay or move away from the robot with equal probability, regardless the action of the robot:

$$p(p_t|r_{t-1}, p_{t-1}, g_{t-1} = 0, a_{t-1}) = \begin{cases} \frac{1}{3} & \text{if } p_t = p_{t-1} \text{ or } p_t = p_{t-1} - 1 \text{ or } p_t = p_{t-1} + 1 \\ 0.0 & \text{otherwise} \end{cases}$$

Finally, the evolution of the person intention is modeled in the following way. We will assume that the the person may change its mind if it is far from the robot. Furthermore, if the person is close to the robot but the robot stays the person may change his mind. Finally, there is always a small chance that the person decides to quit.

Table 2: Parameters employed in the simulations

Parameter	Value	Parameter	Value
pL	1.0	pA	0.2
pD	0.9	pA2	0.1
pF	0.05	pAsmall	0.05
pM	1.0	pR	0.8 (0.5 if $a = 0$, 0.3 if $a = 1$)
pT	0.7 (0.9 if $a = \{2, 3\}$)		
ω_g	10	ω_d	10
ω_{ann}	1	C	100

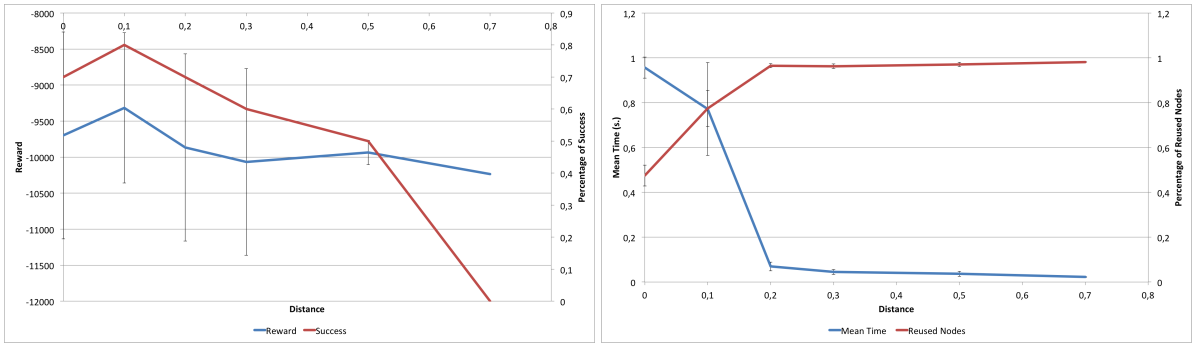


Figure 18: Left: Mean discounted reward and percentage of successful missions. Right: Mean time for planning iteration, and percentage of reused nodes.

$$p(g_t = 0|r_{t-1}, p_{t-1}, g_{t-1} = 1, a_{t-1}) = \begin{cases} pA & \text{if } |p_{t-1} - r_{t-1}| > 3 \\ pA2 & \text{if } |p_{t-1} - r_{t-1}| <= 3 \text{ and } a = \{1\} \\ pAsmall & \text{otherwise} \end{cases}$$

The probability $pA2$ will depend on the robot voice, being lower if the robot waits but asks to be followed ($a = 3$). Finally, we also model the possibility to bring back a person into the tour:

$$p(g_t = 1|r_{t-1}, p_{t-1}, g_{t-1} = 0, a_{t-1}) = \begin{cases} pR & \text{if } |p_{t-1} - r_{t-1}| <= 3 \\ 0 & \text{otherwise} \end{cases}$$

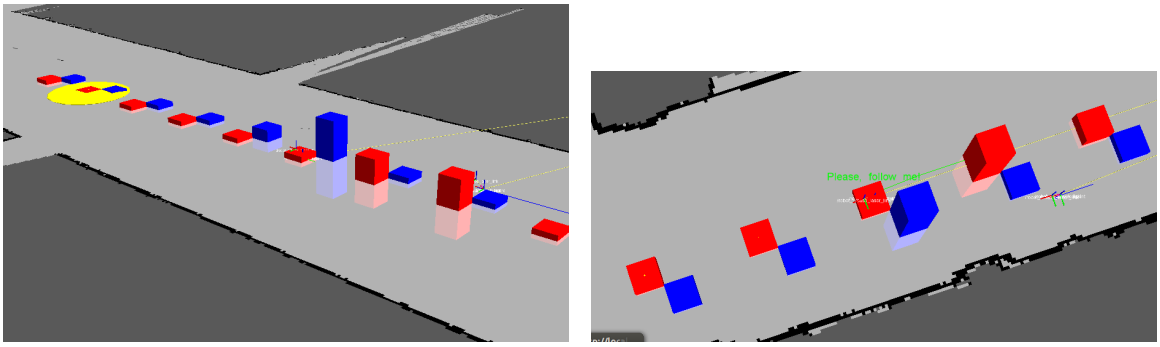
An AND-OR tree for this problem and depth 5 contains more than 10^{10} nodes.

The main design choice in a POMDP is the reward function, with which we encode the desired behavior of the robot. For this task, the objective is to follow the path, guiding the person and adapting to the person motion and intentions. Furthermore, it is important not to lose the person. The reward function described in Section 4.5 is considered here. The term c_{ann} penalizes that the robot ask to be followed ($a = \{2, 3\}$) when the persons are already committed to follow the robot ($g = 1$).

$$c_{ann}(a, g) = \begin{cases} C & \text{if } g = 1 \text{ and } a = \{2, 3\} \\ 0 & \text{otherwise} \end{cases}$$

8.1.2 Solution analysis

Firstly, we have performed simulations of the discrete system, sampling from the transition and observation models with the same parameters (and thus, eliminating the details and effects coming from the discretization). We have performed 10 simulations for several distances to determine similarities between belief points on the GOPOMDP algorithm. The parameters used for this simulations are described in Table 2. In the simulations, we consider as a success if the person arrives at the destination,



(a) (b)

Figure 19: a) The robot and the person (the belief on their positions in blue and red, respectively) are approaching the final destination goal. The yellow circle represents the current belief on the intention of the person to reach the final destination. (b) The robot moves forward while asking the person to follow it.

and a failure if the person does not arrive there after 70 iterations. A planning horizon of 1 second is given to the GOPOMDP algorithm.

These simulations are devoted to analyze the capabilities of GOPOMDP to solve the problem at hand. Figure 18 shows the results. As before, it can be see how increasing the threshold used to determine if two beliefs are similar increments the number of reused nodes during the planning steps, and, thus, reduces the time needed to compute the solution. There is a compromise in the election of the threshold: if the threshold is too large the error on the estimated value function with respect to the optimal one will do that the planner converge to a suboptimal policy, reducing the efficiency of that solution to accomplish the task; if the threshold is too low, the additional complexity of the graph algorithm will balance the savings from reusing parts of the policy graph.

8.1.3 FROG simulator results

Additional simulations are performed using the simulator described in D5.2 [14]. This simulator is based on Stage and ROS. Besides the FROG robot simulated using the same interface, persons can be also simulated. In this particular case, one person and the guide robot are simulated. The map of the Royal Alcázar is used, and a real path between two points of interest is considered.

In the simulation, the robot is controlled by the algorithm, while the person is simulated as an additional robot. The person is commanded giving waypoints to this robot on the path according to the prediction model and transitions described above, but adding additional obstacle avoidance facilities. Also, the simulated person maintains an internal intention variable which is evolved following the same prediction model.

Observations are generated from the poses of the robot and the person, considering the detection and failures of the observation models.

The model employed is the same presented above, modifying some of the parameters to reflect better the additional details introduced by this more realistic simulator (in particular, the probability $pM = 0.8$ as there are chances that the robot does not reach the next point in the path due to local navigation maneuvers; and $pL = 0.9$, to allow some slight misalignments between the poses from the localization and the discretized path).

Fig. 19 describe some of the situations encountered in the execution. Also, Fig. 20 shows the evolution of the distance between the robot and the person in one of the executions, as well as the actions performed by the robot. It can be seen how the estimation on the person intention (not directly observable by the robot) provokes the robot to wait and ask to be followed (action $a = 3$) trying to influence the person to continue the tour.

Besides the GOPOMDP algorithm, we have also tested a basic feedback algorithm, in which the robot

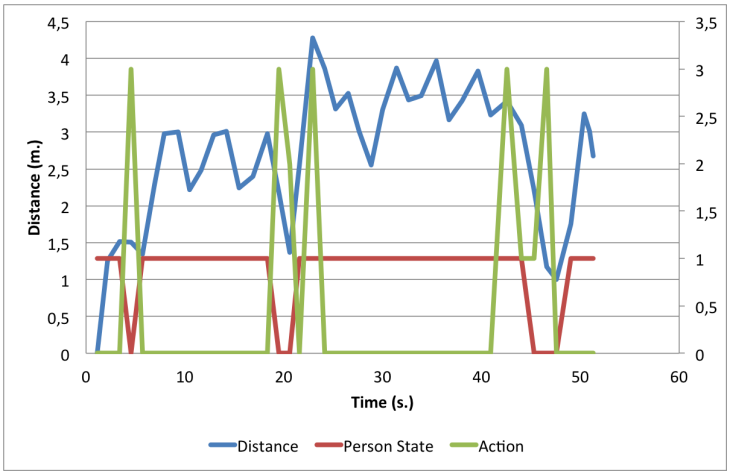


Figure 20: Distance to the person (blue, left axis)), person intention (red, right axis) and robot actions (green, right axis). The robot does not have direct access to these quantities, and it estimates them using its local sensors. The actions are: 0, forward, 1, wait; 2 forward and ask to be followed; 3 stay and ask to be followed. It can be seen how the robot estimates the possibility of the person leaving the tour and uses its robot utterances (in this case, action 3 and 2) to push the person.

Table 3: Comparison between a feedback supervisor and the POMDP planner.

	Distance	Utterances	Failures
Feedback	2.57 ± 1.07	–	30%
POMDP	2.67 ± 0.93	4.67 ± 0.5	10%

moves towards the destination if it detects, through its sensors, that the person is following the robot, and waits otherwise, adapting its pace to the one of the person. 10 simulations were carried out of the same mission by both algorithms. Table 3 summarizes the result. The distance in both cases is similar, slightly lower in the case of the simple controller as it will stop more often (for instance, in case of misdetections), while the POMDP controller plans and predicts into the future. Furthermore, as the POMDP controller is considering the effects of the robot behavior on the person intention, the number of failures (the person quitting the tour) is lower.

8.2 Experimental Results

During the experiments in the Royal Alcázar, some particular experiments have been performed to test the guiding module. In these experiments, a group of researchers followed the robot. A planning horizon of 1 second is considered, and the discretization takes into account the maximum velocity of the robot. The model employed is the same presented in the FROG simulator.

Table 4 shows the mean and minimum distances to the guided persons, as well as the mean number of persons detected per step. Also, the number of robot utterances between POIs is shown (the moments in which the robot ask to be followed). Figure 22 shows a visualization of the experiment, including the belief estimation on the positions of the robot and the persons.

Figure 21 shows the evolution of the distances of the persons with respect to the robot, as well as the actions commanded, for the POI3-POI4 part, which is the longest motions within the tour.

In most of the tours given during the final weeks, the persons were willingly following the robot, and the robot was autonomously navigating between waypoints with no further control. The robot asked to be followed every 10 seconds in any case. The data of those tours has been also used to test a smaller version of the planner in which only the voice actions are considered (asking to be followed). In this case, as the data is processed offline, the effects of the actions cannot be considered by the planner (it is executed "open loop"), but this allows to check the number of occasions in which the planner thinks

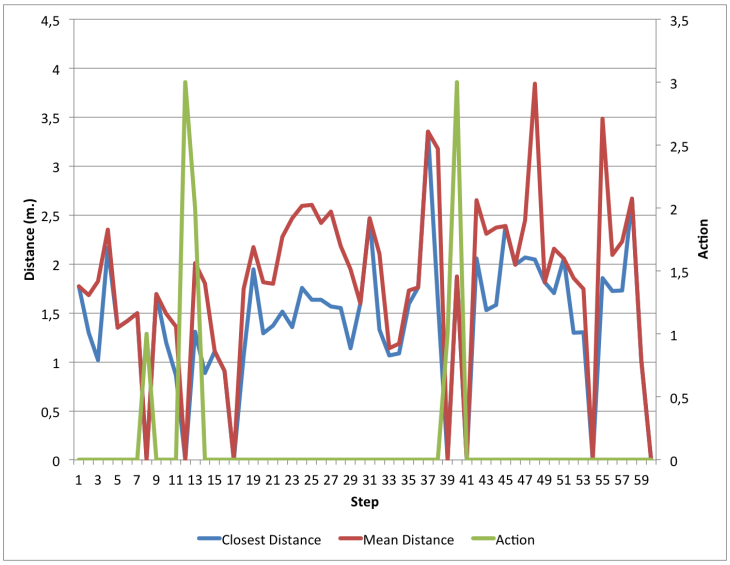


Figure 21: Mean and closest distance during the POI3-POI4 part (red and blue; 0 means no data) and action performed (in green: 0-forward; 1-wait; 2-forward asking to be followed; and 3 wait-asking to be followed).

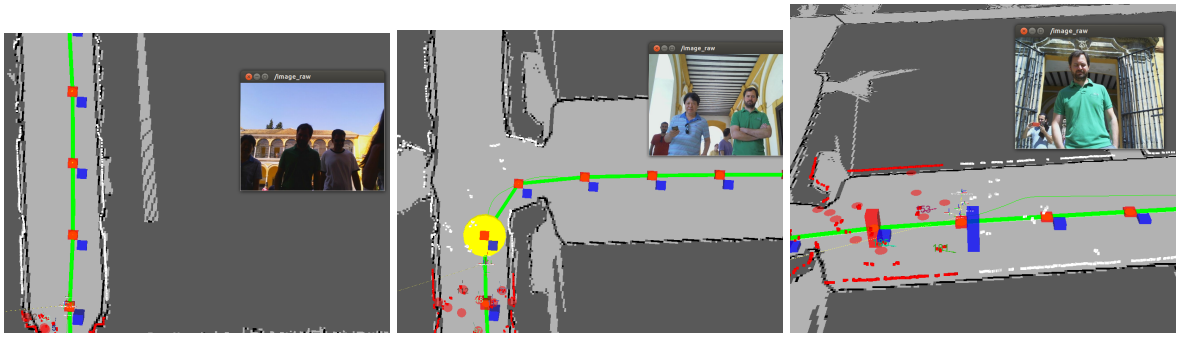


Figure 22: A moment of the guidance experiment. In this experiment, a backwards looking camera was located in the robot for debriefing purposes. In the visualization, the current belief on the position of the robot is represented in blue, the estimated position of the persons is in red. The tracks from the laser can be also seen in the figure.

that the robot should ask to be followed. Table 5 shows the results for one of the full tours, and the estimated voice commands that should have been given. It can be seen, as the persons are actually committed following the robot, very few commands are required according to the plan.

9 Conclusions

This deliverable describes the techniques considered for the guiding task. POMDPs are used as the framework to model the task. POMDPs are a way of reasoning about the uncertainties of the system in a principled way, which allows to enhance the robustness of the robot operation. Furthermore, it also permits to reason about hidden states, like the person goals and intentions, and on the effects of the robot actions on those states.

In order to apply these methods to robotics, new techniques are required, able to cope, in real-time, with larger state, action and observation spaces. The deliverable describes new methods developed in the project to alleviate the complexity of POMDPs. Using this method it is possible to apply a POMDP model to the guiding task in real-time.

The methods described are applied to the task at hand. The results are evaluated through simulations

Table 4: Results for real experiment 1.

	Closest Distance	Mean Distance	Number of Persons	Utterances
POI1-POI2	2.07 ± 0.85	2.56 ± 0.72	1.34 ± 1.95	2
POI2-POI3	1.55 ± 0.46	1.84 ± 0.34	2.24 ± 1.72	3
POI3-POI4	1.60 ± 0.47	2.04 ± 0.60	2.12 ± 1.46	3
POI4-POI5	1.51 ± 0.63	1.80 ± 0.52	2.06 ± 2.16	0
POI6-POI7	1.76 ± 0.90	2.16 ± 0.77	1.03 ± 1.33	2

Table 5: Results from real experiment 2.

	Closest Distance	Mean Distance	Number of Persons	Utterances
POI1-POI2	2.47 ± 0.70	2.75 ± 0.57	1.69 ± 1.21	0
POI2-POI3	2.51 ± 0.75	2.92 ± 0.76	1.41 ± 1.24	0
POI3-POI4	2.04 ± 0.96	2.50 ± 0.92	1.61 ± 1.61	2
POI4-POI5	2.71 ± 1.09	3.58 ± 0.79	1.67 ± 1.25	0
POI6-POI7	1.97 ± 0.79	2.15 ± 0.73	0.70 ± 1.14	0

and in experiments carried out with the robot in the real scenarios of the project.

Regarding the use of POMDPs for navigation tasks in pedestrian environments, one of the main issues is the definition of the models and the design of the reward functions to define the particular task. In Table 2, while some of the parameters can be measured (like the sensor failure rates), the transitions probabilities regarding the person states should be introduced by hand. As future work we consider the learning of the models and reward functions as part of the problem. Regarding the rewards, Deliverable D2.2 [12] has presented techniques that can be used to learn costs from examples given by experts. In the same way, the prediction functions could be learned from experience.

Bibliography

[1] S. Alili, M. Warnier, M. Ali, and R. Alami. Planning and plan-execution for human-robot cooperative task achievement. In *19th International Conference on Automated Planning and Scheduling*, 2009.

[2] Minoo 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*, ICMLA '05, pages 43–48, Washington, DC, USA, 2005. IEEE Computer Society.

[3] Blai Bonet and Héctor Geffner. Solving pomdps: Rtdp-bel vs. point-based algorithms. In *Proceedings of the 21st international jont conference on Artifical intelligence*, IJCAI'09, pages 1641–1646, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[4] Wolfram Burgard, Armin B Cremers, Dieter Fox, Dirk Hähnel, Gerhard Lakemeyer, Dirk Schulz, Walter Steiner, and Sebastian Thrun. Experiences with an interactive museum tour-guide robot. *Artificial intelligence*, 114(1):3–55, 1999.

[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] E. Choi and C. Lee. Feature extraction based on the Bhattacharyya distance. *Pattern Recognition*, 36(8):1703–1709, 2003.

[7] Aurélie Clodic, Hung Cao, Samir Alili, Vincent Montreuil, Rachid Alami, and Raja Chatila. SHARY: A Supervision System Adapted to Human-Robot Interaction. In Oussama Khatib, Vijay Kumar, and George J. Pappas, editors, *Experimental Robotics, The Eleventh International Symposium, ISER 2008, July 13-16, 2008, Athens, Greece*, volume 54 of *Springer Tracts in Advanced Robotics*, pages 229–238. Springer, 2008.

[8] Aurélie Clodic, Sara Fleury, Rachid Alami, Raja Chatila, Gérard Bailly, Ludovic Brethes, Maxime Cottret, Patrick Danes, Xavier Dollat, Frédéric Eliseï, et al. Rackham: An interactive robot-guide. In *Robot and Human Interactive Communication, 2006. ROMAN 2006. The 15th IEEE International Symposium on*, pages 502–509. IEEE, 2006.

[9] Philip R Cohen and Hector J Levesque. Intention is choice with commitment. *Artificial intelligence*, 42(2):213–261, 1990.

[10] FROG Consortium. Deliverable D1.1: Functional Requirements, Interaction and Constraints. `https://www.frogrobot.eu/wordpress/wp-content/uploads/2014/03/D1.1_Final.pdf`.

[11] FROG Consortium. Deliverable D2.1: Robot 6-DOF Precise Localization Component. `https://www.frogrobot.eu/wordpress/wp-content/uploads/2014/03/d2.1-v1.1.pdf`.

[12] FROG Consortium. Deliverable D2.2: Path planning and execution component for efficient and human-aware navigation. `https://www.frogrobot.eu/wordpress/wp-content/uploads/2014/01/d2-2v2.pdf`.

[13] FROG Consortium. Deliverable D3.1: Final feature extraction component. `https://www.frogrobot.eu/wordpress/wp-content/uploads/2014/03/D3.1-Finalv2.pdf`.

[14] FROG Consortium. Deliverable D5.2: Simulation environment for FROG AR. `https://www.frogrobot.eu/wordpress/wp-content/uploads/2014/01/d5-2.pdf`.

[15] D.M. Endres and J.E. Schindelin. A new metric for probability distributions. *Information Theory, IEEE Transactions on*, 49(7):1858 – 1860, july 2003.

[16] Felix Faber, Maren Bennewitz, Clemens Eppner, Attila Görög, Christoph Gonsior, Dominik Joho, Michael Schreiber, and Sven Behnke. The humanoid museum tour guide Robotinho. In *Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 891–896, Toyama, Japan, September 2009.

[17] D. Feil-Seifer and M. Mataric. People-aware navigation for goal-oriented behavior involving a human partner. In *Proceedings of the IEEE International Conference on Development and Learning (ICDL)*, 2011.

[18] Amalia F Foka and Panos E Trahanias. Predictive autonomous robot navigation. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 1, pages 490–495. IEEE, 2002.

[19] Anais Garrell and Alberto Sanfeliu. Local optimization of cooperative robot movements for guiding and regrouping people in a guiding mission. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 3294–3299. IEEE, 2010.

[20] Anais Garrell and Alberto Sanfeliu. Cooperative social robots to accompany groups of people. *The International Journal of Robotics Research*, 31(13):1675–1701, 2012.

[21] J.F. Gómez-Lopera, J. Martínez-Aroza, A.M. Robles-Pérez, and R. Román-Roldán. An analysis of edge detection by using the Jensen-Shannon divergence. *Journal of Mathematical Imaging and Vision*, 13(1):35–56, 2000.

[22] Eric A. Hansen and Shlomo Zilberstein. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1-2):35–62, 2001.

[23] M. Hauskrecht. Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, 13:33–94, 2000.

[24] 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.

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

[26] David Hsu, Wee Sun Lee, and Nan Rong. What makes some pomdp problems easy to approximate? In *Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems*, 2007.

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

[28] Thibault Kruse, Amit Kumar Pandey, Rachid Alami, and Alexandra Kirsch. Human-aware robot navigation: A survey. *Robotics and Autonomous Systems*, 61(12):1726–1743, 2013.

[29] H. Kurniawati, D. Hsu, and W.S. 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.

[30] Jianhua Lin. Divergence Measures Based on the Shannon Entropy. *IEEE Transactions on Information Theory*, 37:145–151, 1991.

[31] M.E. López, L.M. Bergasa, R. Barea, and M.S. Escudero. A navigation system for assistant robots using visually augmented POMDPs. *Autonomous Robots*, 19(1):67–87, 2005.

[32] 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*, pages 367–374. AUAI Press, 2009.

[33] L. Merino, A. Gilbert, J. Capitan, R. Bowden, J. Illingworth, and A. Ollero. Data Fusion in Ubiquitous Networked Robot Systems for Urban Services. *Annals of Telecommunications, Special Issue Ubiquitous Robots*, 67, 2012.

[34] Oscar Martinez Mozos, Ryo Kurazume, and Tsutomu Hasegawa. Multi-part people detection using 2D range data. *International Journal of Social Robotics*, 2(1):31–40, March 2010.

[35] Illah Nourbakhsh, Clayton Kunz, and Thomas Willeke. The mobot museum robot installations: A five year experiment. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 4, pages 3636–3641. IEEE, 2003.

[36] S. Ong, S. Wei Png, D. Hsu, and W. Sun Lee. POMDPs for Robotic Tasks with Mixed Observability. In *Proc. Robotics: Science and Systems, RSS*, 2009.

[37] Amit Kumar Pandey and Rachid Alami. Towards a sociable robot guide which respects and supports the human activity. In *Proceedings of the Fifth Annual IEEE International Conference on Automation Science and Engineering*, CASE'09, pages 262–267, Piscataway, NJ, USA, 2009. IEEE Press.

[38] 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)*, pages 133–147, 2006.

[39] Xavier Perrin, Francis Colas, Cedric Pradalier, and Roland Siegwart. Learning to identify users and predict their destination in a robotic guidance application. In Andrew Howard, Karl Iagnemma, and Alonzo Kelly, editors, *Field and Service Robotics*, volume 62 of *Springer Tracts in Advanced Robotics*, pages 377–387. Springer Berlin Heidelberg, 2010.

[40] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *International Joint Conference on Artificial Intelligence*, volume 18, pages 1025–1032. Citeseer, 2003.

[41] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, Acapulco, Mexico, 2003. IJCAI.

[42] 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.

[43] Stéphane Ross, Joelle Pineau, and Brahim Chaib-draa. Theoretical analysis of heuristic search methods for online pomdps. In *Advances in Neural Information Processing Systems, NIPS*, 2007.

[44] Stephane Ross, Joelle Pineau, Sebastien Paquet, and Brahim Chaib-draa. Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research*, 2008.

[45] Roland Siegwart, Kai O. Arras, Samir Bouabdallah, Daniel Burnier, Gilles Froidevaux, Xavier Greppin, Björn Jensen, Antoine Lorotte, Laetitia Mayor, Mathieu Meisser, Roland Philippsen, Ralph Piguet, Guy Ramel, Gregoire Terrien, and Nicola Tomatis. Robox at Expo.02: A large-scale installation of personal robots. *Robotics and Autonomous Systems*, 42(3-4):203–222, March 2003.

[46] Emrah Akin Sisbot, Luis Felipe Marin-Urias, Rachid Alami, and Thierry Siméon. A Human Aware Mobile Robot Motion Planner. *IEEE Transactions on Robotics*, 23(5):874–883, 2007.

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

[48] Trey Smith. *Probabilistic Planning for Robotic Exploration*. PhD thesis, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, July 2007.

[49] Trey Smith. ZMDP software for POMDP and MDP planning, 2012. `http://www.cs.cmu.edu/ trey/zmdp/`.

[50] Matthijs T. J. Spaan and Nikos Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24:195–220, 2005.

[51] T. Taha, J.V. Miro, and G. Dissanayake. Pomdp-based long-term user intention prediction for wheelchair navigation. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 3920–3925, 2008.

[52] S. Thrun, M. Beetz, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, and C. Hahnel. Probabilistic algorithms and the interactive museum tour-guide robot minerva. *The International Journal of Robotics Research*, 19:972–999, October 2000.

[53] Gian Diego Tipaldi and Kai O. Arras. Planning Problems for Social Robots. In *Proceedings fo the Twenty-First Internacional Conference on Automated Planning and Scheduling*, pages 339–342, 2011.

[54] F. Topsoe. Some inequalities for information divergence and related measures of discrimination. *Information Theory, IEEE Transactions on*, 46(4):1602 –1609, jul 2000.

[55] Peter Trautman and Andreas Krause. Unfreezing the robot: Navigation in dense, interacting crowds. In *IROS*, pages 797–803. IEEE, 2010.

[56] Håkan Warnquist, Jonas Kvarnström, and Patrick Doherty. Iterative Bounding LAO*. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI)*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 341–346. IOS Press, 2010.