Analysis of Methods for Playing Human Robot Hide-and-Seek in a Simple Real World Urban Environment

Alex Goldhoorn, Alberto Sanfeliu, and René Alquézar

Institut de Robòtica i Informàtica Industrial, CSIC-UPC,
C/ Llorens i Artigas 4-6, 08028 Barcelona, Spain
{agoldhoorn,sanfeliu,ralqueza}@iri.upc.edu
http://www.iri.upc.edu

Abstract. The hide-and-seek game has many interesting aspects for studying cognitive functions in robots and the interactions between mobile robots and humans. Some MOMDP (Mixed Observable Markovian Decision Processes) models and a heuristic-based method are proposed and evaluated as an automated seeker. MOMDPs are used because the hider's position is not always known (partially observable), and the seeker's position is fully observable. The MOMDP model is used in an off-line method for which two reward functions are tried. Because the time complexity of this model grows exponentially with the number of (partially observable) states, an on-line hierarchical MOMDP model was proposed to handle bigger maps. To reduce the states in the on-line method a robot centered segmentation is used. In addition to extensive simulations, games with a human hider and a real mobile robot as a seeker have been done in a simple urban environment.

Keywords: Robotics, Human Robot Interaction, Hide-and-Seek, MOMDP

1 Introduction

Small differences of the hide-and-seek game can be found in the interactions between humans, or robots and humans, in urban spaces, as for example looking for a person in a crowded urban environment. In 2005 Johansson and Balkenius [1] suggested that the game of hide-and-seek is an ideal domain for studying cognitive functions in robots and it is a basic mechanism for human robot interaction in mobile robotics, because hide-and-seek requires the robot to navigate, search, interact on and predict actions of the opponent. In this work we focus on the prediction of the opponent's actions. This could be used to find persons; furthermore hide-and-seek could be seen as a simplification of search-and-rescue.

The hide-and-seek game is an interactive game in which the two players interact indirectly, one trying to catch while the other is trying to flee (the seeker is a robot and the hider is a person in our case). Players of the game can follow several strategies to win the game, depending on their role. The robot's strategy could be simply pre-programmed, but a more intelligent approach would be to decide a strategy which can be applied in multiple situations. In this game, as in real life, there are uncertainties of the location of the other player. The later is a reason why we chose MOMDPs (Mixed Observability Markov Decision Processes) [2, 3], a variant of Partially Observable Markov Decision Processes (POMDPs) [4, 5]. POMDPs have been successfully applied to various robotic tasks [6, 7], but unfortunately, computing an optimal policy exactly is generally intractable [8] because the size of the belief space grows exponentially with the number of states. POMDPs have been also used in [2, 3, 9] to play the game *tag*, a variant of hide-and-seek.

In our MOMDP approach of the hide-and-seek game the robot and the person move at the same time from one position to another (given by the (x, y) coordinates of the grid cells where they are located). The robot updates a belief (a probabilistic state estimate) and chooses an action (a robot movement) to maximize the expected future reward, meanwhile the person chooses also an action following its own strategy to win the game.

MOMDPs are POMDPs in which the partly and fully observable state variables are separated. In our game we assume that the robot's position is fully observable, and the hider position is not always visible (partially observable). This results in a compact lower-dimensional representation of its belief space.

In this work, we analyze and apply an off-line and on-line MOMDP model as proposed in [10]. This off-line model works very well for maps with a small number of grid cells, but it becomes intractable (PSPACE-hard, [8]) for a large number of grid cells. For this reason we proposed an on-line MOMDP model [10] that computes a locally near-optimal policy at every step, which in principle can be applied to large maps. The on-line method is a hierarchical model of two levels, where the top level MOMDP has a reduced number of states. The state reduction is obtained through a segmentation process of the map. The bottom level contains a fine resolution of the map in which the beliefs are computed. Finding a policy however is done with the top level MOMDP. The on-line method can also be applied to navigation problems, or other like problems where a high resolution map is used. Finally an automated heuristic seeker is tested for comparison.

All seeker methods have been tested in simulation and in real life experiments using a real robot (Tibi and Dabo [11, 12]) against a human hider in a simple urban environment (see Figure 3).

2 Definition of the Hide-and-Seek Game

Our version of the hide-and-seek game is defined as follows. There are two players, a seeker and a hider, who play on a grid of $n \times m$ cells. The grid contains: a special free cell called the *base*, other *free cells* on which the players can move, and *obstacle cells* that are not accessible by the players and also limit their mutual visibility. In the initial state of the game, the seeker is placed on the base and the hider can be placed on any free cell not visible from the base.

The game is run for a maximum of H time steps. At each time step, both the seeker and the hider can stay in the same cell or move to a free neighbor cell in an 8-connectivity neighborhood (i.e. a maximum of 9 actions for each player). The seeker wins if it approaches the hider sufficiently (we use a distance of 1 cell) and "catches" it. The hider wins if it reaches the base before being caught by the seeker. And the result is a *tie* when no player has won within the maximum predefined time H or if both reach the base at the same time. Orientation angles are not considered for the players, they are both supposed to have 360° visibility at each time step, only limited by the obstacles. Hence, the visibility for each player is calculated with a ray-tracing algorithm in simulation or with a range laser in the real world.

The MOMDPs presented in the next two sections model the game from the point of view of the seeker, this is, we want to learn a policy for the seeker assuming that the hider follows a certain unknown strategy. It is also assumed that the seeker's state is fully observable for itself (no local uncertainty), whereas the hider's state is partially observable, in the sense that the hider's position is identified if the hider is visible from the seeker's position and otherwise it is *unknown* for the seeker.

In our simulations, two virtual robots are involved: an automated seeker applying the MOMDP learnt policy or using a heuristic is confronted with a random or a "smart" (heuristically driven) hider. In our real-world experiments, a physical robot (Dabo) has the role of the seeker and plays against a human opponent in the role of the hider.

3 Off-line MOMDP Model for Hide-and-Seek

The hide-and-seek game can be cast as an off-line MOMDP model [2, 3], where the state is composed by the grid cell positions of both players. This means that the number of states is the square of the number of grid cells of the 2D map where the game is going to be played. The number of grid cells depends on the resolution that we want to consider in the game (e.g., a grid cell of 1×1 m^2 in a 2D map of 10×10 m^2 implies 10 000 MOMPD states). Formally, the hide-and-seek game is modelled as:

$$\langle \mathcal{X}, \mathcal{Y}, \mathcal{A}, \mathcal{O}_\mathcal{X}, \mathcal{O}_\mathcal{Y}, T_\mathcal{X}, T_\mathcal{Y}, Z_\mathcal{X}, Z_\mathcal{Y}, R, \gamma \rangle \tag{1}$$

where:

- $\mathcal{X}$: the fully-observable state variable that contains the seeker's position $x = (x_{\text{seeker}}, y_{\text{seeker}})$;
- $\mathcal{Y}$: the partially-observable state variable containing the hider's position $y = (x_{\text{hider}}, y_{\text{hider}})$;
- $\mathcal{A}$: the 9 actions of the seeker: *north, northwest, west, ..., halt.* Each of the actions represents a movement of one grid cell at maximum per time step, except for the action *halt* which represents staying at the same state;

- $\mathcal{O}_\mathcal{X}$: $\mathcal{O}_\mathcal{X} = \mathcal{X}$, since $o_x = x$ for all states of the seeker, note that this is added for completeness and in reality x is used directly instead of o_x;
- $\mathcal{O}_\mathcal{Y}$: $\mathcal{O}_\mathcal{Y} = \mathcal{Y} \cup \{unknown\}$, which is the union of the set of hider positions and a special observation value *unknown*, which represents the cases when the hider is not visible to the seeker;
- $T_\mathcal{X}$, $T_\mathcal{X}(x, y, a, x') = p(x'|x, y, a)$: the transition probabilities of the seeker's state given an action. In our case the actions are deterministic for the seeker's position: given the current position x, action a, and the map it brings the seeker directly to a new state x', independently of the current position of the hider y. Therefore these probabilities will always be 1 or 0, taking into account that the result of an infeasible action is defined as staying on the same cell. For example, when the seeker has a wall in the north of it and it chooses the action *north*, the action cannot be done and therefore it will result in not modifying the seeker's state. Also reaching the final state will result in staying in the same state;
- $T_\mathcal{Y}$: the transition probabilities of the hider's state given a seeker's action and locations of the seeker and hider, $T_\mathcal{Y}(x, y, a, x', y') = p(y'|x, y, a, x')$. These probabilities are not as evident as the previous ones since the action of the hider is not known. There are two suggested solutions: the first is to spread the probabilities of the movement of the hider uniformly, the second option is to use historical data of human players. Both options are discussed in detail in [13], but since the results with historical data were not significantly better, we chose for the easier option of using uniform probabilities. Also here the probability will be 1 if a final state has been reached;
- $Z_\mathcal{X}$: the observation probabilities $Z_\mathcal{X}(x', y', a, o_x) = p(o_x|x', y', a)$ will be 1 if $o_x = x'$ and 0 otherwise, like $\mathcal{O}_\mathcal{X}$ this is put here for completeness;
- $Z_\mathcal{Y}$: the observation probabilities $Z_\mathcal{Y}(x', y', a, o_x, o_y) = p(o_y|x', y', a, o_x)$ depend on the locations of the seeker and hider and the map. The probability will be 1 if $o_y = y'$ and y' is visible from x', or if $o_y =$*unknown* and y' is not visible from x', otherwise the probability will be 0;
- R: two reward functions have been tested (see below);
- γ: the discount factor.

Two different reward functions R are described next; these give rise to two different off-line MOMDP models, from which a near-optimal policy can be learnt off-line [2, 3]. These functions are:

- *Simple* reward: non-zero values only for final states (positive for $x = y$ and negative for $y =$*base*, $x \neq y$).
- *Triangle* reward: this reward makes use of the three important distances in the game: the distance between the seeker and the hider (d_{sh}), the distance between the hider and the base (d_{hb}) and between the seeker and the base and (d_{sb}). The reward is shown in Equation (2), where D is a maximum distance constant depending on the map size. In order to compute the three distances one may use the simple Euclidean distance or the length of the shortest path that depends on the map [13]. We have used the last one for our experiments.

$$R(s,h,b) = \begin{cases} D - d_{sh}, & \text{if } d_{hb} > d_{sb} \\ -d_{sh}, & \text{otherwise} \end{cases} \tag{2}$$

While the triangle reward is much more informative than the simple reward, its computational cost is also slightly higher. Note that the simple reward can be computed extremely fast at each step without the need of memorising its values for each state. On the other hand, for the triangle reward, either its values are precalculated for each state (higher memory cost) or the computation time is increased considerably if calculated at each step.

Finally we have to define the initial belief $b_{Y,0}$: if the hider is visible then the belief of that state is 1.0, otherwise the belief is uniformly distributed over the not visible states.

4 On-line MOMDP Model for the Hide-and-Seek Game

The issue with the off-line method is that it takes a relatively long time to generate a policy (from 2 hours for maps of 12×12 up to more than 40 hours for maps of 20×20). Furthermore the time and memory complexity grows with the number of states due to the curse of history and dimensionality [14].

We present a hierarchical model, as shown in Figure 1(a), in which the lower level is an MOMDP as defined in the previous section. The big difference is that this MOMDP is not used to calculate the policy, but instead the top level MOMDP with less states is used. The state reduction of the top level MOMDP is obtained by grouping a spatially adjacent group of positions in the bottom level map. In the top MOMDP, the transition and observation probabilities, and the initial belief will be calculated from those in the bottom MOMDP. The top MOMDP will be solved on-line and directly thereafter used to choose the best action to do. Furthermore the bottom level is used to keep track of the belief. The actions are common to both levels.

4.1 Bottom-level MOMDP

The bottom level is a full MOMDP (as (1)) defined in the same way as described in Section 3, however no policy is computed at this level. Only the beliefs of all the states at this level, are computed before generating the top-level MOMDP. The belief is initialized as in the off-line version, and when an action a has been executed (whereby the seeker's position changed from x to x') and an observation (o_x and o_y) has been obtained, the bottom-level belief $b_{\mathcal{Y}}$ is updated:

$$\begin{aligned} b'_{\mathcal{Y}}(y') = \eta p(o_x|x',y',a)p(o_y|x',y',a,o_x) \\ \times \sum_{y\in\mathcal{Y}} p(x'|x,y,a)p(y',x,y,a,x')b_{\mathcal{Y}}(y) \end{aligned} \tag{3}$$

where $\eta = 1/p(o|b,a)$ is a normalization factor.

4.2 Top-level MOMDP

To reduce the number of states a segmentation function ψ is used that groups adjacent map cells. This segmentation is used to generate the new top state variables $\mathcal{Y}_T$. Where each value of y_T will be associated with a spatially adjacent set of values of y. Formally, the function $\psi(y_T)$ gives the set of bottom-level adjacent states which are covered by each of the top level state y_T. When reducing the number of partially observable states $\mathcal{Y}$, the belief space is reduced. Also the fully observable state variable $\mathcal{X}$ could be reduced in the same way, but this did not give significant better results nor did it increase the speed in finding the policy.

The problem of finding a proper function ψ can be posed as a segmentation based on the map itself, the location of the players, the reward obtained in each state and/or the belief of each state.

The segmentation can be done by applying some known image segmentation algorithm such as k-means [15] where a fixed set of k clusters is defined in the "intensity value" domain. In any case, the number of segmentation regions obtained should be limited to assure a small number of states.

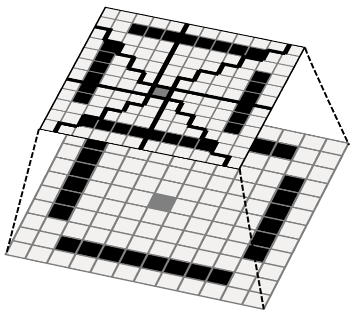
Robot Centered Segmentation We propose a method that centers on the robot and divides the space based on the eight directions seen from the robot and the distance. Figure 1 shows the robot centered segmentation in which the robot is at location 0, and the segmentation is done from that point in the eight directions and based on a fixed distance to the center. The focus of this segmentation is the direction and not the exact location; which is sufficient because a new robot centered top MOMDP model is generated for each step. Since the hider and base positions are of vital importance for the game, they are added as a separate superstate if known; these superstates will represent only one cell in the bottom level.

Top MOMDP The top-level MOMDP can be defined as follows:

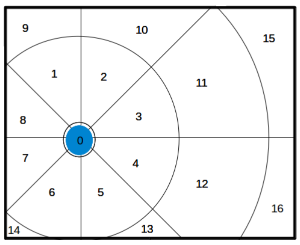
$$\langle \mathcal{X}_T, \mathcal{Y}_T, \mathcal{A}_T, \mathcal{O}_{\mathcal{X},T}, \mathcal{O}_{\mathcal{Y},T}, T_{\mathcal{X},T}, T_{\mathcal{Y},T}, Z_{\mathcal{X},T}, Z_{\mathcal{Y},T}, R_T, \gamma \rangle \tag{4}$$

where some of the top MOMDP components will be equal to those of the bottom level MOMDP:

- $\mathcal{X}_T = \mathcal{X}$;
- $A_T = A$, the actions keep referring to the lower level actions, in the top level however the transition probability is adapted to abstract top level states;
- $\mathcal{O}_{\mathcal{X},T} = \mathcal{O}_{\mathcal{X}}$;
- $\mathcal{O}_{\mathcal{Y},T} = \mathcal{O}_{\mathcal{Y}}$, these observations do not change, but their probabilities $Z_{\mathcal{Y},T}$ do change;
- $Z_{\mathcal{X},T} = Z_{\mathcal{X}}$.



(a) Layered segmentation



(b) Robot centered segmentation

Fig. 1. The hierarchical method with two layers is shown in (a), where the top layer has less states due to segmentation of the lower level map. The *robot centered* segmentation (b) centers on the robot's location (0 in the figure) and from there on creates segments based on the direction and distance.

Therefore, the Top MOMDP reduces to the following tuple:

$$\langle \mathcal{X}, \mathcal{Y}_T, \mathcal{A}, \mathcal{O}_\mathcal{X}, \mathcal{O}_\mathcal{Y}, T_{\mathcal{X},T}, T_{\mathcal{Y},T}, Z_\mathcal{X}, Z_{\mathcal{Y},T}, R_T, \gamma \rangle \tag{5}$$

where the transition and observation probabilities and rewards are averaged from the bottom level:

$$p(x'|x, y_T, a) = \frac{1}{|\psi(y_T)|} \sum_{y \in \psi(y_T)} p(x'|x, y, a) \tag{6}$$

$$p(y'_T \mid x, y_T, a, x') = \frac{1}{|\psi(y_T)|} \sum_{y' \in \psi(y'_T)} \sum_{y \in \psi(y_T)} p(y'|x, y, a, x') \tag{7}$$

Note that in our implementation we have not made the seeker's action (x) dependent on the hider's position (y), therefore equation (6) will not change in our case, but has been put here for completeness.

To speed up the process of finding a good policy the final state is defined to stay in the same state independent of the action a: $p(x_f|x_f, y_{T,f}, a) = 1.0$ and $p(y_{T,f}|x_f, y_{T,f}, a, x_f) = 1.0$ where $(x_f, y_{T,f})$ is a final state. The final state is defined as either $y_{T,f}$ being on the base, or if $x_f \in \psi(y_{T,f})$, i.e. the seeker is in the same superstate as the hider.

The observation probability is simply averaged:

$$p(o_y|x', y'_T, a) = \frac{1}{|\psi(y'_T)|} \sum_{y' \in \psi(y'_T)} p(o_y|x', y', a) \tag{8}$$

The top reward function $R_T(x, y_T, a)$ could have been defined averaging the rewards of the bottom level, however it was found that using only rewards in the final states converged much quicker in a good policy. A reward of 1 is given when the seeker is in the hider's superstate ($x \in \psi(y_T)$), and -1 if the hider is at the base.

Before a policy is learned, the bottom level belief is compressed to the top level:

$$b_{\mathcal{Y},0,T}(y_T) = \sum_{y \in \psi(y_T)} b_{\mathcal{Y}}(y) \tag{9}$$

Note that besides the $\mathcal{Y}$ component, also the $\mathcal{X}$ and O components can be segmented; however experiments [10] showed that this did not give better results nor faster convergence.

4.3 The On-line Algorithm

The algorithm for solving the on-line two-level MOMDP is based on [16] and uses SARSOP to generate a policy [9, 2]. SARSOP is a state-of-the-art off-line solver for POMDPs, but can be used on-line by simply alternating a planning and an execution phase [17]. Algorithm 1 shows how the on-line method is implemented. First the bottom belief is initialized based on the seeker position (x) and the belief of the hider position (b_Y), which will make the belief 1.0 on its visible position, otherwise it will be uniformly distributed over all non-visible cells. From now on the algorithm is run until a final condition is reached: some player wins or the time has passed. The segmented hider states are calculated in line 3; here we apply the robot centered segmentation (Figure 1). In line 4 the belief is compressed up using formula (9), where after the top level MOMDP M_T is generated from the bottom level MOMDP M and the segmented states S_T using the formulas presented in the previous subsection. In line 6 the policy P_T is learned and applied to get the best action. When the action is done an observation of the seeker's own position and the hider's position is done in line 9, which is used to update the bottom level belief.

Algorithm 1 On-line two-level MOMDP planner.

```
1: $(b_Y, x) \leftarrow$ **initBelief(M)**
2: **while** not finished game **do**
3:     $S_T \leftarrow$ **segmentStates**$(M, x)$
4:     $b_{Y,T} \leftarrow$ **compressBelief**$(M, S_T, b_Y, x)$
5:     $M_T \leftarrow$ **generateTopLevel**$(M, S_T)$
6:     $P_T \leftarrow$ **solveTopLevel**$(M_T, b_{Y,T}, x)$
7:     $a \leftarrow$ **getBestAction**$(M_T, P_T, b_{Y,T}, x)$
8:     **doAction**$(a)$
9:     $(b_Y, x) \leftarrow$ **doObservation()**
10:    $(b_Y, x) \leftarrow$ **updateBelief**$(b_Y, x, o_y)$
11: **end while**
```

5 Smart Seeker

Using the previously defined reward (2) an automated heuristic seeker has been made, called the *Smart Seeker*. This seeker calculates a score for each action it can take and then chooses the action with the maximum score. At maximum 9 actions are possible (one step or staying at the same position) for both the seeker and hider. One action gets the seeker to a position, which can be used to calculate R; but at the same time the hider can make a move, which we take into account by averaging the score over these moves:

$$w(s', h, b) = \sum_{h' \in \text{moves}(h)} R(s', h', b)/|\text{moves}(h)| \tag{10}$$

When the hider is not visible to the seeker the only thing we know is that the hider is at a not visible position; therefore the score w is calculated for every possible hider's position and then averaged.

6 Simulations and Experiments

In this section the set-up of the simulations and the real world experiments are explained.

6.1 Maps

In order to do the simulations and real-world experiments, we created a discretized 2D grid map of the environment, where the players can move in one of the 8 directions.

For both the simulations and the real-world experiments we built a small square in between two buildings with a size of 7 m $\times$ 9 m. The size of the grid cells is 1 m which implies a grid size of 7 $\times$ 9 cells. Two obstacles with a length of three cells (3 m) have been placed on different positions as well as the base. We used the two maps which can be seen in Figure 2, and for simulations we used the same maps but with a higher resolution: 9 $\times$ 12.

6.2 Simulations

We have first run a series of simulations playing against a *random hider*, which moves completely randomly, and a *smart hider*. The latter uses the triangle rule and uses the following equation to score each possible action: $w = D - d_{hb} + 0.4d_{sh} + noise$ where $D = \text{rows} \times \text{cols}$, d_{hb} the distance between the hider and the base, and d_{sh} the distance between the seeker and the hider. The noise has a maximum value of 2 and is reduced as soon as the distance is less than 3, because when a hider is either close to the seeker or to the base, it should respectively always flee or go to the base directly.

We have used the hierarchical and the off-line MOMDP models. To generate the policies we used the *Approximate POMDP Planning Software* [1] [2]. The

[1] `http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/`

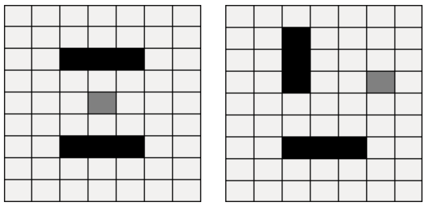

(a) Map 1 (b) Map 2

Fig. 2. The maps used in the simulated and real experiments. Black cells are obstacles, the dark gray cell is the base.

simulations where done on a stand alone PC with 8 GB of RAM and an Intel Core$^{\text{TM}}$i5 CPU 760 @ 2.80 GHz with 4 cores and Ubuntu 12.04 as OS.

We set up a maximum time (measured in time steps) to play the game which is computed based on the size of the map: $2(rows + cols)$, that means 32 time steps for our simulations. The win condition for the robot has been adapted to its size. The robot wins if it is within one cell distance of the hider.

6.3 Real-World Experiments

The robot Dabo of the URUS project [11, 12] has been used to play the hide-and-seek game. Dabo has two on-board computers (Intel Core 2 Quad CPU @ 2.66 and 3.00 GHz with 4 GB RAM) that manage all the running processes and sensor signals, and a laptop is used for external monitoring. The systems run Ubuntu 12.04 and use a middle ware called ROS, a software developmental environment for robot system integration that provides a useful and large set of libraries and tools.

A map was created by the robot by scanning the environment using the robot range-lasers. For the game, the Dabo robot used the front and back range-lasers to know its position and to detect persons and obstacles [11, 12]. The robot movements were given by the seeker algorithm that uses the MOMDP models. To detect persons we used a classifier developed in the IRI based on the range-laser information.

Since the seeker, the Dabo robot, has been designed to work in a limited controlled environment, and because these experiments were a first step in the real-world, we had to impose some constraints in the hide-and-seek game. First of all, the robot and the person were only allowed to do one action at the same time step in one of the eight directions (or no motion) and they could move at most one grid cell. The grid cells were marked with tape on the floor such that it was clear to the person. The human hider started at any grid location. After this, the robot scanned the environment to detect any person in the neighborhood and to detect its own position using the localization algorithm. These two measurements were used as observations in the MOMDP model. Since the model only allowed discrete movements of one cell distance, the observations were checked before feeding them into the model. The people detector sometimes detected persons outside the limits of the field or on the obstacles, therefore these detections were filtered out. The observations were entered into the model which gave the action. Next the person is told to do its next movement and at the same time the robot was commanded to go to its next position. Then another scan was done, and this was continued until one player won or the time passed.

6.4 Results

Simulations More than 5000 simulated games were done with the different models against the two automated hiders (*random* and *smart*). The results of the games are shown in Table 1 and Table 2. It shows that the off-line model with the *triangle* reward works best ($p < 0.001$; Fishers exact test, two-sided, this has been used to check all the win statistics). Only on the small map (7×9) the on-line method was found to work better than the off-line method ($p < 0.05$), and both the on-line method and off-line method were found to work better than the heuristic method ($p < 0.05$).

Comparing the automated hiders we see that more games were won against the *random* hider (97%) than against the *smart* hider (95.3%; $p < 0.001$). No significant difference was found in winning for the two map configurations (Figure 2), nor for their sizes.

Table 1. The win percentages for the four seeker methods against the two automated hiders. The last column shows the total number of simulated games done.

Map	Hider	Seeker	Win	Lose	Tie	Total
1	random	off-line – simple	99.8%	0.2%	0.0%	483
		off-line – triangle	100.0%	0.0%	0.0%	481
		on-line	99.6%	0.0%	0.4%	245
		smart	92.8%	0.0%	7.2%	360
	smart	off-line – simple	90.9%	9.1%	0.0%	243
		off-line – triangle	100.0%	0.0%	0.0%	243
		on-line	93.5%	6.3%	0.3%	400
		smart	97.3%	0.0%	2.7%	366
2	random	off-line – simple	99.7%	0.3%	0.0%	380
		off-line – triangle	99.7%	0.0%	0.3%	380
		on-line	99.0%	0.5%	0.5%	194
		smart	89.2%	0.0%	10.8%	360
	smart	off-line – simple	91.1%	8.9%	0.0%	192
		off-line – triangle	99.5%	0.0%	0.5%	187
		on-line	95.5%	4.3%	0.5%	400
		smart	95.0%	0.0%	5.0%	361
Total			96.9%	0.9%	2.2%	4475

In Table 2 win statistics are shown per map size and seeker type. It also shows the average number of actions and average duration per step for the won games.

Note that passing 32 actions resulted in a *tie*. The off-line MOMDP model used least steps when winning ($p < 0.001$; Wilcoxon ranksum). When the off-line method used the *triangle* reward it required more steps to win than using the *simple* reward. It was found that the seeker needed more steps on map 1 than on map 2 ($p < 0.001$; Wilcoxon ranksum), which might be because map 1 is symmetric and map 2 is not. For the on-line method the average time per step is highest (see last column of Table 2), because the MOMDP model is calculated and a policy is learned at each time step; and lowest for the heuristic method ($p < 0.001$; Wilcoxon ranksum test, 2-sided).

Table 2. The win percentages per map size and seeker type. The one before last column shows the average ± standard deviation number of actions for won games, and the last column shows the average ± standard deviation duration of one action for won games.

Map Size	Seeker	Win	Lose	Tie	Total	Win Actions	Win Dur.(s)/Act.
7 × 9	off-line – simple	96.6%	3.4%	0.0%	760	5.25 ± 2.49	0.19 ± 0.09
	off-line – triangle	99.7%	0.0%	0.3%	756	6.83 ± 4.3	0.17 ± 0.1
	on-line	97.6%	2.0%	0.5%	656	9.07 ± 6.09	2.39 ± 0.24
	smart	92.9%	0.0%	7.1%	1012	10.67 ± 7.31	0.13 ± 0.09
9 × 12	off-line – simple	97.2%	2.8%	0.0%	538	7.26 ± 3.61	0.17 ± 0.09
	off-line – triangle	100.0%	0.0%	0.0%	535	9.22 ± 5.57	0.15 ± 0.09
	on-line	94.5%	5.1%	0.3%	583	11.71 ± 7.42	6.70 ± 0.37
	smart	95.2%	0.0%	4.8%	435	12.77 ± 8.94	0.13 ± 0.09

The duration of the calculation of the off-line policies are shown in Table 3. Although the off-line method with triangle reward works better than the on-line method, we can also see from Table 3 that the calculation of an off-line policy with triangle reward takes relatively much more time. The on-line method requires us to calculate a policy at every time step, which for the 9 × 12 on average were 6.7 s, and on average it took 13 steps to win (see Table 2), which results in approximately 87 s to complete a game. This is quite less than the calculation of the off-line policy for the triangle reward.

Real-World Experiments With the real robot a total of 44 games were played against 15 adults from which 12 games were won by the hider and 32 by the seeker, see Table 4 for the detailed results. From the 32 games won by the seeker, 9 games ended in a situation where the hider reached the base, but at the same time was caught by the seeker. No game ended due to reaching the maximum number of time steps, 32. The average number of actions for won and lost games is shown in the last columns of Table 4,

There is no significant difference in the game results for map 1 and map 2, but the games are won in significantly less steps on map 1 ($p < 0.001$; Mann-Whitney test).

Table 3. The time it took to calculate the policies off-line for the different maps and with the triangle and simple reward.

Map Size	Map	Reward	Time (s)
7×9	1	simple	4.5
		triangle	23.2
	2	simple	5.2
		triangle	71.5
9×12	1	simple	29.9
		triangle	480.0
	2	simple	36.8
		triangle	260.0

Like in the simulations (Table 2) using the off-line MOMDP with the triangle reward resulted in significantly more won games than the on-line hierarchical method. The low win percentages shown for the on-line method could be explained by a special strategies used by the human players.

Table 4. The results of the real world experiments against de different seekers. The win column shows the percentage of games in which the seeker won even when the hider reached the base; **tie* shows the games in which the hider reached the base, but the seeker caught it. The last two columns show the average number of actions it took the seeker to win or lose the game respectively.

Map	Seeker	Win	(Tie*)	Lose	Total	Win Act.	Lose Act.
1	off-line – simple rew.	64.3%	(7.1%)	35.7%	14	4.4	7.2
	off-line – triangle rew.	100.0%	(25.0%)	0%	8	8.3	-
	on-line	40.0%	(0%)	60.0%	5	6.5	12.7
	smart	100.0%	(25.0%)	0%	4	14.5	-
2	off-line – simple rew.	100.0%	(50.0%)	0%	2	26	-
	off-line – triangle rew.	100.0%	(75.0%)	0%	4	10	-
	on-line	25.0%	(0%)	75.0%	4	19	17
	smart	66.7%	(33.3%)	33.3%	3	10	15

Three games are shown in Figure 3 which show the map and the laser collision detections which represent obstacles and walls. The light yellow area represents the game field. On top of the map the obstacles are drawn as black rectangles and the base as a dashed square. For both the seeker and hider, steps are shown with arrows and lines; S_0 is the seeker start place, which is always the base, and H_0 the hider's start place.

The first image (Figure 3(a)) shows a game played on map 1 in which the model used the simple reward. It can be seen that the robot followed the hider around the obstacle and with this the hider won. The second map (Figure 3(b)) shows map 1, but this time the robot used the triangle reward. It can be seen that the robot tried to find the hider at some hidden place, and when it could not see it (but it did have knowledge about the hider's position through the belief) it returned to protect the base. This game ended in a tie because both the hider and the seeker arrived at the base at the same time. The last map (Figure 3(c)) shows a game where the hider tries to get the seeker to follow him but when the distance to the base is too far, the seeker returns and finally catches the hider before reaching the base.

7 Conclusions

In this paper we analyzed four methods to play the hide-and-seek game, from which three were based on an MOMDP model and one on a heuristic method. These have been extensively tested in simulation, and initial experiments have been performed with a mobile robot playing against a human hider in a simple real world urban environment. The simulated experiments showed that all methods performed really well, and the best method was the off-line MOMDP model with the triangle reward.

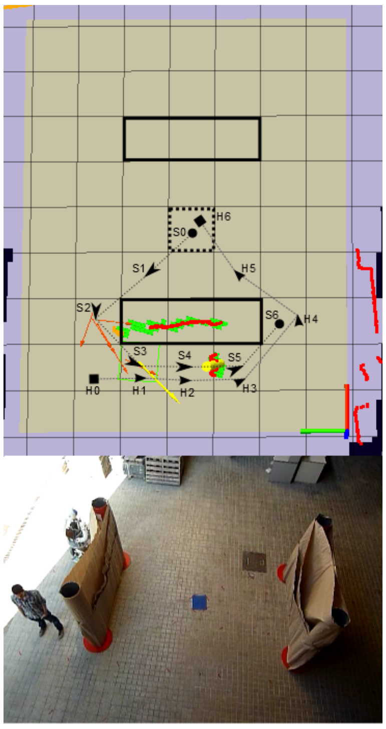
The on-line hierarchical method was previously proposed to reduce the number of partially observable states, and thereby tackling the *curse of dimensionality*. Even though the very good results of the on-line method in simulation, this was not reflected in experiments done in the real world. This can be explained by the strategy used by the hiders. In the simulations the *random* hider moved randomly and therefore was relatively easy to catch since it did not consider the game objectives. The *smart* hider on the other hand did take into account the rules of the game and therefore was more predictable. For the human players it was found that some of them did not take the optimal path, but used a strategy in which they "mislead" the robot by leading it around an obstacle and thereby won.

Although relatively few experiments were done, they gave us important insights in the functionality of the automated seeker methods used by a real mobile robot in the real world playing (*interacting* and *predicting*) against humans. In this first step towards the real world, limitations were set to have similar conditions as in the simulations, but our next steps are to overcome these limitations by incorporating sensing uncertainties and working in bigger real world environments.

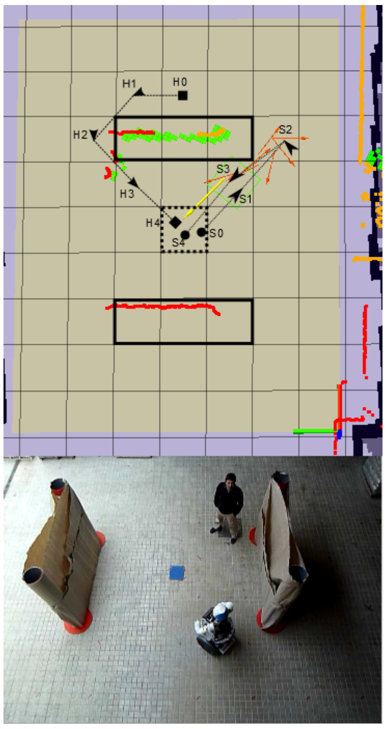
Acknowledgments. This work has been partially funded by the Spanish Ministry of Economy and Competitiveness under projects TaskCoop DPI2010-17112.

References

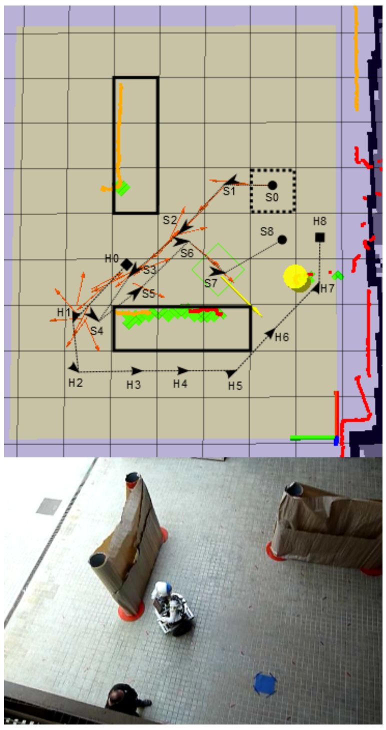
1. Johansson, E., Balkenius, C.: It's a child's game: Investigating cognitive development with playing robots. In: International Conference on Development and Learning. (2005) 0:164



(a) Seeker at S_3 and hider at H_3.



(b) Players are at S_3 and H_3.



(c) Players are at S_7 and H_7.

Fig. 3. Fragments of three played games against a human hider. In (a) and (b) map 1 was used, in (c) map 2. The seeker used the simple reward in (a) and in (b) and (c) the triangle reward was used. The light yellow brown area shows the field on which the game was played. The dashed square is the base, the black rectangles are the obstacles, which were also detected by the robot's laser (red, orange and green). The yellow arrow shows the goal position and the red arrows the previous odometry.

2. Ong, S.C.W., Png, S.W., Hsu, D., Lee, W.S.: Planning under Uncertainty for Robotic Tasks with Mixed Observability. The International Journal of Robotics Research **29**(8) (May 2010) 1053–1068
3. Araya-López, M., Thomas, V., Buffet, O., Charpillet, F.: A closer look at MOMDPs. In: 22nd International Conference on Tools with Artificial Intelligence - ICTAI. (2010)
4. Braziunas, D.: Pomdp solution methods. Technical report, University of Toronto (2003)
5. Hauskrecht, M.: Value-function approximations for partially observable markov decision processes. Journal of Artificial Intelligence Research **13** (2000) 33–94
6. Cassandra, A., Kaelbling, L., Kurien, J.: Acting under uncertainty: discrete bayesian models for mobile-robot navigation. In: Intelligent Robots and Systems '96, IROS 96, Proceedings of the 1996 IEEE/RSJ International Conference on. Volume 2. (nov 1996) 963–972 vol.2
7. Spaan, M., Vlassis, N.: A point-based pomdp algorithm for robot planning. In: Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on. Volume 3. (2004) 2399 – 2404 Vol.3
8. Papadimitriou, C., Tsisiklis, J.: The complexity of markov decision processes. Mathematics of Operations Research **12**(3) (1987) 441–450
9. Kurniawati, H., Hsu, D., Lee, W.: Sarsop: efficient point-based pomdp planning by approximating optimally reachable belief spaces. In: Robotics: Science and Systems, 2008. (2008)
10. Goldhoorn, A., Sanfeliu, A., Alquézar, R.: Comparison of momdp and heuristic methods to play hide-and-seek. Accepted for the Sixteenth International Conference of the Catalan Association of Artificial Intelligence (2013)
11. Trulls, E., Corominas Murtra, A., Pérez-Ibarz, J., Ferrer, G., Vasquez, D., Mirats-Tur, J., Sanfeliu, A.: Autonomous navigation for mobile service robots in urban pedestrian environments. Journal of Field Robotics (May 2010)
12. Sanfeliu, A., Andrade-Cetto, J., Barbosa, M., Bowden, R., Capitán, J., Corominas, A., Gilbert, A., Illingworth, J., Merino, L., Mirats, J.M., Moreno, P., Ollero, A., Sequeira, J.a., Spaan, M.T.J.: Decentralized Sensor Fusion for Ubiquitous Networking Robotics in Urban Areas. Sensors **10**(3) (March 2010) 2274–2314
13. Georgaraki, C.: A POMDP approach to the hide and seek game. Master's thesis, Universitat Politècnica de Catalunya, Barcelona, Spain (2012)
14. Pineau, J., Gordon, G., Thrun, S.: Point-based value iteration: An anytime algorithm for pomdps. In: International Joint Conference on Artificial Intelligence, 2003. (2003) 477–484
15. Stockman, G., Shapiro, L.G.: Computer Vision. 1st edn. Prentice Hall, Upper Saddle River, NJ, USA (2001)
16. Foka, A., Trahanias, P.: Real-time hierarchical POMDPs for autonomous robot navigation. Robotics and Autonomous Systems **55**(7) (July 2007) 561–571
17. Ross, S., Pineau, J., Paquet, S., Chaib-Draa, B.: Online Planning Algorithms for POMDPs. The journal of artificial intelligence research **32**(2) (July 2008) 663–704