Universitat Politècnica De Catalunya

Doctoral Programme:

Automatic Control, Robotics And Computer Vision

Ph.D. Thesis

Searching and Tracking of Humans in Urban Environments with Humanoid Robots

Alex Goldhoorn

Advisor:

Prof. Alberto Sanfeliu

Co-advisor:

Prof. René Alquézar

June, 2017

Searching and Tracking of Humans in Urban Environments with Humanoid Robots

by Alex Goldhoorn

Doctoral Programme:
Automatic Control, Robotics and Computer Vision

This thesis has been completed at:

Institut de Robòtica i Informàtica Industrial, CSIC-UPC

Advisor:
Alberto Sanfeliu
Co-advisor:
René Alquézar

The last version of this document and extra material will be available at
http://alex.goldhoorn.net/thesis.

To my wife Juliette,

my parents Anneke and Klaas

and my sister Brenda.

Acknowledgements

First of all, I would like to thank my supervisors Alberto Sanfeliu and René Alquézar for their support, ideas, feedback and funding. Furthermore, I would like to thank: Anaís Garrell for her support, contributions to the work and help during the experiments; Fernando Herrero and Sergi Hernández for their technical support and their patience during the experiments; all the people that helped and participated in the experiments; and all people that contributed to the research in general. Also, I want to thank my office colleagues and all the other colleagues for making the stay at IRI pleasant and fruitful.

And last but not least, I want to thank my family and friends, and especially my wife Juliette for their mental support, patience and love.

This work has been supported by the Institut de Robòtica i Informàtica Industrial (IRI), Universitat Politècnica de Catalunya (UPC) and the following research projects:

- The Collective Experience of Empathic Systems (CEEDs) [FP7-ICT-2009-5] (funded by the European Union).
- RobTaskCoop: Cooperación robots humanos en áreas urbanas [DPI2010-17112] (funded by MINECO).
- Rob-In-Coop: Interacción, aprendizaje y cooperación robot-humano en áreas urbanas [DPI2013-42458-P] (funded by MINECO).

Abstract

Searching and tracking are important behaviours for a mobile service robot to assist people, to search-and-rescue and, in general, to locate mobile objects, animals or humans. Even though searching might be evident for humans, for robots it is not, since it requires exploring, handling noisy sensors, coping with dynamic obstacles, and coordination in the case of multiple agents.

In this thesis, we present several methods to search and track a person in an urban environment. All methods were first tested extensively in simulation and then in real-life, using one or two mobile service robots, called Tibi and Dabo. The robots have laser rangefinders, which are used to navigate, to detect obstacles and to detect people's legs. Since we focus on search-and-track methods, we use existing methods for robot navigation, for people detection and person recognition.

First tests are done with the hide-and-seek problem, in which the robot learns to catch the hider. Concretely, a Mixed Observable Markov Decision Process (MOMDP) model is used, in which the seeker's location is fully observable and the hider's location partially observable. Since the computational complexity depends on the number of states, we propose a hierarchical on-line method that reduces the state space by grouping them together. Although the method worked properly in simulation, in the real-life experiments the results were not satisfying and the on-line policy calculation was not fast enough to work in real-time.

To handle larger environments, work in continuous state space and run in real-time, we propose to use an approach, the Continuous Real-time POMCP (CR-POMCP), that does Monte-Carlo simulations to learn a policy. The method performed correctly in simulation, but on the real robot it resulted in slow zigzag movements. Therefore, a new method is proposed, which uses the highest probable locations, according to its

probability map (belief). Since the belief propagation of the POMCP resembles how a Particle Filter (PF) works, we also propose a method that uses a PF to maintain the belief. The PF method has to handle lack of observations, therefore, we introduce a special weight function. Both belief update methods take into account sensor and actuator noise, false negative detections, false positive detections (for a short time) and dynamic obstacles.

Finally, a cooperative distributed multi-agent method is presented, it makes use of the previous belief update functions, but it uses all the agents' observations. Next, the search locations are assigned to explore the whole working environment, taking into account: the belief, the distance to the search location and if another agent already will search close to it.

Summarizing, the main contributions of this thesis are several methods to search and track a person in an urban environment with one or more mobile service robots. All these methods have been shown to work through a set of simulations and real-life experiments.

Resum

La cerca i el seguiment de persones són comportaments importants per un robot mòbil de servei per poder assistir, trobar i ajudar als humans, i en general, per localitzar objectes, animals o vianants. Tot i que la cerca és fàcil per als humans, no ho és per a un robot, ja que requereix exploració, maneig de soroll de sensors, fer front als obstacles dinàmics, i la coordinació en el cas de múltiples agents.

En aquesta tesi presentem diferents mètodes per a buscar i seguir a una persona en un entorn urbà. Tots els mètodes han estat provats extensivament en simulació i després en el món real, utilitzant dos robots mòbils de servei, la Tibi i en Dabo. Els robots utilitzen sensors làser per a navegar, detectar obstacles i detectar les cames de les persones. Atès que aquest treball es centra en mètodes de cerca i seguiment, s'han usat els mètodes existents per a la navegació del robot, la detecció i el reconeixement de persones.

Primerament, s'han fet proves amb el conegut joc del fet i amagar, on el robot aprèn a trobar l'amagador. S'ha fet servir el model Mixed Observable Markov Decision Process (MOMDP), on la posició del trobador és completament visible i la posició de l'amagador és parcialment visible. Degut a que la complexitat computacional depèn del nombre d'estats, es proposa un mètode jeràrquic en línia que redueix l'espai d'estats, tot agrupant-los. Tot i que el mètode va funcionar correctament en simulació, en els experiments reals els resultats no van ser satisfactoris, i el càlcul de la política no va ser prou ràpid com per treballar en temps real.

Per tal de fer front a entorns de més superfície, treballar en l'espai continu i executar en temps real, proposem un nou enfocament, el Continuous Real-time POMCP (CR-POMCP), que fa simulacions de Monte-Carlo per aprendre una política. El mètode va funcionar correctament en l'entorn simulat, però a l'entorn real el robot realitzava

lents moviments en zig-zag. Per tant, es proposa un mètode nou, que utilitza els llocs amb més alta probabilitat, d'acord amb el seu mapa de probabilitats (belief). Atès que la propagació de les probabilitats en el POMCP és similar al funcionament d'un filtre de partícules (PF), proposem, a més, un mètode que utilitza un PF per mantenir el belief. El mètode de PF ha de manejar la manca d'observacions. Per tant, introduïm una funció del pes especial. Tots dos mètodes de creences tenen en compte el soroll dels sensors i actuadors, la detecció de falsos negatius i positius (per a un curt període de temps) i els obstacles dinàmics.

Finalment, es presenta un mètode multi-agent distribuït cooperatiu, que fa ús de les anteriors funcions d'actualització de la creença (belief), i a més utilitza totes les observacions dels agents. En el proper pas, les ubicacions de cerca s'assignen mitjançant l'exploració de l'entorn de treball, tenint en compte la creença, la distància a la ubicació de cerca i si un altre agent ja buscarà a prop d'ella.

En resum, les principals contribucions d'aquesta tesi són diversos mètodes per a la cerca i seguiment d'una persona en un entorn urbà amb un o més robots de serveis mòbils. Tots aquests mètodes han demostrat que funcionen a través d'un conjunt de simulacions i experiments en la entorn real dinàmics.

Resumen

La búsqueda y el seguimiento de personas son comportamientos importantes para un robot móvil de servicio para poder asistir, buscar y ayudar a la gente, y en general, para localizar un objeto, animal o humano. Aunque la búsqueda puede parecer muy fácil para los humanos, para los robots no lo es, ya que requiere explorar, manejar ruido de sensores, enfrentarse con obstáculos dinámicos y la coordinación en el caso de haber más agentes.

En esta tesis, presentamos diferentes métodos para buscar y seguir a una persona en un entorno urbano. Todos los métodos han sido probados excesivamente en simulación y en experimentos reales, usando uno o dos robots móviles de servicio, Tibi y Dabo. Los robots tienen localizadores láser, los cuales se usan para navegar, detectar obstáculos y detectar piernas. Ya que el principal enfoque en este trabajo son los métodes de buscar-y-seguir, utilizamos métodos existentes para la navigación del robot, la detección de personas y el reconecimiento del humano.

Las primeras pruebas se hicieron con el juego de escondite, en el cual el robot aprende a buscar el ocultador. Concretamente, usamos un modelo MOMDP, donde la posición del buscador es completamente observable y la posición del ocultador lo es parcialmente. Como la complejidad computacional depende del número de estados, proponemos un método en línea jerárquica, que reduce el espacio de los estados agrupándolos. Aunque el método funcionó correctamente en simulación, en los experimentos reales los resultados no fueron satisfechos y el cálculo de la política en línea no fue suficientemente rápido.

Para poder trabajar en áreas largas, espacio continuo y en línea, proponemos un enfoque, el Continuous Real-time POMCP (CR-POMCP), que hace simulaciones de Monte-Carlo para aprender la política. El método funcionó correctamente en simu-

lación, pero con el robot real resultaba en movimientos lentos en forma zigzag. Por eso, otro método fue propuesto, el cual usa las posiciones con la probabilidad más alta según el mapa de probabilidades (belief). Como la propagación del belief se parece mucho a como funciona un PF, proponemos un método que usa un PF para mantener el belief. El método PF tiene que manejar la falta de observaciones y por eso introducimos una función del peso especial. Los dos métodos para actualizar el belief tienen en cuenta el ruido de los sensores y actuadores, falsos negativos y positivos (durante un periodo corto de tiempo) y obstáculos dinámicos.

Finalmente, se presenta un método cooperativo y distribuido para multi agentes, que usa el mapa de probabilidades (belief), y éste usa todas las observaciones. Después, se asigna las posiciones de búsqueda a los agentes, explorando el entorno, y teniendo en cuenta: la probabilidad de la posición, la distancia a la posición y si otro agente ya buscará cerca del lugar.

En resumen, la contribución más importante de esta tesis son diferentes métodos de búsqueda y seguimiento de una persona en un entorno urbano con uno o más robots de servicio móviles. Todos estos métodos fueron comprobados en simulación y en experimentación real.

Samenvatting

voor een mobiele servicerobot om mensen te kunnen assisteren, zoek- en reddingsoperaties uit te kunnen voeren en, in het algemeen, om mobiele objecten, dieren of mensen te lokaliseren. En hoewel zoeken evident is voor de mens, is dit niet het geval voor robots. Het vereist namelijk verschillende vaardigeheden zoals verkennen, omgaan met ruis in de sensoren en dynamische obstakels en coördinatie in het geval van meerdere agenten.

In deze thesis presenteren we verschillende methoden om een persoon te zoeken en te volgen (search-and-track) in een stedelijke omgeving. De methoden zijn eerst getest met simulaties en daarna met twee echte mobiele servicerobots, genaamd Tibi en Dabo. De robots hebben laser rangefinders om te navigeren en om obstakels en benen van mensen te detecteren. Omdat we ons richten op het zoeken en volgen van mensen gebruiken we bestaande methoden voor de robotnavigatie, detectie van personen en het herkennen van de gezochte persoon.

De eerste testen gedaan met het spel hide-and-seek, waar de robot leert om de persoon te vinden. Een Mixed Observable Markov Decision Process (MOMDP) is gebruikt, waar de positie van de robot (zoeker) volledig waarneembaar is en die van de persoon (verstopper) gedeeltelijk. Omdat de berekeningscomplexiteit van het aantal staten afhangt, stellen we een hiërarchische online methode voor die de staatruimte verkleind door het groeperen van staten. Hoewel de methode goed werkt in simulatie, waren de resultaten van de experimenten met de robots minder goed. Verder was het online genereren van de politiek niet snel genoeg.

Om in grote omgevingen te kunnen werken en continue staatruimte te gebruiken in real-time, hebben we de Continuous Real-time POMCP (CR-POMCP) voorgesteld. Dit algoritme gebruikt Monte-Carlo simulaties om de politiek (beste actie voor een

bepaalde staat) te leren. De methode werkte goed in de simulaties, maar de robot maakte in de experimenten zigzaggende bewegingen. Om deze reden hebben we een nieuwe methode voorgesteld die gebruik maakt van een waarschijnlijkheidskaart (belief) over de lokatie van de persoon. Omdat de propagatie van de POMCP lijkt op hoe een Particle Filter (PF) werkt, stellen we ook een methode voor die een PF gebruikt om de belief bij te houden. De PF methode moet om kunnen gaan met missende observaties en we introduceren daarom een speciale gewichtsfunctie. Beide belief update methoden houden rekening met sensor en actuator ruis, incorrecte negatieve detecties, incorrecte positieve detecties (gedurende een korte tijd) en dynamische obstakels.

Als laatst presenteren we een coöperatieve multi-agent methode die gebruik maakt van de genoemde belief update methoden, maar tevens gebruik maakt van de observaties van alle agenten. Vervolgens worden de zoeklocaties toegekend om de hele omgeving te verkennen waarbij rekening gehouden wordt met: de belief, de afstand tot de zoeklocatie en of een andere agent al dichtbij de lokatie gaat zoeken.

Samenvattend zijn de belangrijkste bijdragen van deze thesis de verschillende methoden om personen te zoeken en te volgen in een stedelijke omgeving met één of meer mobiele servicerobots. Alle methoden zijn getest met simulaties en met experimenten met echte robots.

Contents

List of Figures

List of Tables

Abbreviations

2D	Two Dimensional
3D	Three Dimensiona
ACT-R	Adaptive Control of Thought-Rational
APPL	Approximate POMDP Planning
AR	Augmented Reality
BRL	Barcelona Robot Lab
CPF	Centralized Particle Filter
CR-POMCP	Continuous Real-time POMCP
DP	Dynamic Programming
DPF	Distributed Particle Filter
EKF	Extended Kalman Filter
EM	Expectation Maximization
FME	Facultat de Matemàtiques i Estadística
fov	field of view
GMM	Gaussian Mixture Model
GPG	Generalized Policy Graph
GRASP	Greedy Randomized Adaptive Search Procedure
GSP	Graph Search Problem
HB	Highest Belief
HB-CR-POMCP	Highest Belief Continuous Real-time POMCP
HB-PF	Highest Belief Particle Filter
HMM	Hidden Markov Model
HOG	Histogram of Oriented Gradients
HRI	Human Robot Interaction

IF	Information Filter
IRI	Institut de Robòtica i Informàtica Industrial
KF	Kalman Filter
MCTS	Monte-Carlo Tree Search
MCVI	Monte-Carlo Value Iteration
MDP	Markov Decision Process
MHT	Multiple Hypothesis Tracking
MOMDP	Mixed Observable Markov Decision Process
OS	Operating System
PBVI	Point-Based Value Iteration
PF	Particle Filter
POMCP	Partially Observable Monte-Carlo Planning
POMDP	Partially Observable Markov Decision Process
RGB-D	Red Green Blue Depth
RL	Reinforcement Learning
ROS	Robot Operating System
SARSOP	Successive Approximations of the Reachable Space under Optimal Policies
SVM	Support Vector Machine
TD Learning	Temporal-Difference Learning
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
UPC	Universitat Politècnica de Catalunya
URUS	Ubiquitous Networking Robotics in Urban Areas
USAR	Urban Search and Rescue
WSN	Wireless Sensor Network

Nomenclature

A	The actions which can be done, this is a discrete set.
$D_{\max}$	The maximum distance in a map (used in hide-and-seek).
H	The maximum number of time steps for a (hide-and-seek) game.
O	Observation space, which, in this work, is always discrete. It can contain different observation variables.
$O_\mathcal{X}$	The set of fully visible observation variables in an MOMDP.
$O_\mathcal{Y}$	The set of partially visible state variables in an MOMDP.
P_{vis}	The visibility probability $P_{\text{vis}}(s_1, s_2)$ gives the probability of seeing s_1 from s_2 and vice versa.
R	The reward function $R(s, a, s')$ returns the reward in function of the state s, action a and next state s'. Note that in many models the state only depends on the current state s: $R(s)$.
S	The state space can contain one or more continuous or discrete state variables.
T	The transaction function $T(s, a, s') = p(s'\|s, a)$ is the probability of going from state s to state s' using action a.
$T_\mathcal{X}$	The transition probabilities for the fully visible state variables in an MOMDP: $T_\mathcal{X}(x, y, a, x') = p(x'\|x, y, a)$.
$T_\mathcal{Y}$	The transition probabilities for the partially visible state variables in an MOMDP: $T_\mathcal{Y}(x, y, a, x', y') = p(y'\|x, y, a, x')$
Z	The observation probability function $Z(o, s', a) = p(o\|s', a)$ gives the probability of observing o from state s' after having done action a.
$Z_\mathcal{X}$	The observation probabilities for the fully visible state variables in an MOMDP: $Z_\mathcal{X}(x', y', a, o_x) = p(o_x\|x', y', a)$.
$Z_\mathcal{Y}$	The observation probabilities for the partially visible state variables in an MOMDP: $Z_\mathcal{Y}(x', y', a, o_x, o_y) = p(o_y\|x', y', a, o_x)$
Γ	The set of α-vectors: $\Gamma = \{\alpha_i\}$.
Σ	The covariance.
α	The learning factor.
α_{vis}	The reduction for the visibility probability (Eq. 5.7).
$\boldsymbol{H}$	List of Highest Belief points.
γ	The discount factor; which is used in the Value Function to reduce importance of future expected rewards.

$\mathcal{A}$	The list of all agents.
$\mathcal{B}$	The belief space.
$\mathcal{G}$	The POMDP simulator, which is used by the POMCP solver, instead of the transition function T. $(s', o, r) = \mathcal{G}(s, a)$ returns the new state s', observation o and reward r, based on the current state s and action a.
$\mathcal{I}$	The initialization function for the POMCP belief: $b_0 = \mathcal{I}(o_0)$, where o_0 is the initial observation.
$\mathcal{X}$	The set of fully visible state variables in an MOMDP.
$\mathcal{Y}$	The set of partially visible state variables in an MOMDP.
$\mathcal{Y}_T$	The set of top level states of the partially visible state variables in a Hierarchical MOMDP, which is defined by the segmentation function ψ.
μ	The mean.
π	The policy of a Reinforcement Learning model, such as an MDP or a POMDP.
ψ	The segmentation function that segments the top level of the Hierarchical MOMDP: $\psi(Y_T) = \{y_0, y_1, \ldots, y_k\}$; with Y_T being the top level state that covers the bottom level states y_i.
σ_{agent}	The standard deviation of the Gaussion noise for the agent's movement.
$\sigma_{\text{obs,agent}}$	The standard deviation of the Gaussion noise for the observation of the agent's location.
$\sigma_{\text{obs,person}}$	The standard deviation of the Gaussion noise for the observation of the person's location.
σ_{person}	The standard deviation of the Gaussion noise for the person's movement.
σ_w	A parameter to tune the PF weight in Eq. 5.10. A higher value of σ_w results in a higher weight for the same distance.
ε_b	The belief error ε_b (see Eq. 5.11) indicates the weighted distance from the real location x of the person, with respect to the belief.
b_0	The initial belief.
c	The exploration constant, which is used in the POMCP policy calculation, to explore other actions than the ones with the highest expected reward.
d_{cons}	The distance to which an observation is categorized to be consistent, which is used in the `ConsistencyCheck` function (Algorithm 6.13).
$d_{\text{max\_range}}$	The maximum range in which the exploration score (EXPL_SCORE, Eq. 6.3) is affected.
$d_{\text{max\_search}}$	The maximum range in which a Highest Belief point is searched.
d_{max}	The maximum tree search depth for the POMCP policy calculation.
$d_{\text{v,max}}$	The maximum distance for the visibility probability (Eq. 5.7).
d_{sh}	The shortest path distance between s and h.

e_{count}	The expand count of a POMCP indicates how many node visits are required before a subtree is expanded.
n_{belief}	The number of belief points.
n_{hb}	The number of HB points.
$n_{\text{particles}}$	The number of particles.
n_{sim}	The number of simulations to generate the policy for a POMCP model.
$p_{\text{false\_neg}}$	The probability of a false negative detection.
$p_{\text{false\_pos}}$	The probability of a false positive detection.
$p_{\text{see\_occ}}$	The probability of seeing the person even though the person is occluded.
$p_{\text{v,max}}$	The maximum of the visibility probability (Eq. 5.7).
p_o	A probability of the correctness of observation o; this is used to indicate how trustworthy a certain observation is.
t_{update}	The time between updates of the goal location for the robot.
w_{cons}	The particle weight (Eq. 5.10), which is given when the person is not visible, but the observation is consistent.
w_{inc}	Used in the particle weight (Eq. 5.10), which is given when the person is not visible and the observation is not consistent.
RAND	A function that returns a uniformly random value in $[0, 1]$.

Glossary

α-vector The α-vector represents the Value Function for a specific action a over the $|S|$-dimensional hyper-plane, where S is the set of discrete states.

Multi-agent HB-CR-POMCP Explorer A multi-agent method with distributed coordination that uses the HB-CR-POMCP and an explorer to search and track a person.

ACT-R Adaptive Control of Thought-Rational; a cognitive architecture by [Anderson et al., 2004].

adversarial Adversarial games include an opponent that wants to flee and another that wants to attack the target, e.g. pursuit-evasion.

agent An (intelligent) agent (IA) is an autonomous entity that observes the environment through its sensors, and acts upon it using actuators. Moreover, it has a goal (i.e. is rational). Note that this can be a robot or a person.

anytime An anytime algorithm can return a result without having finished completely, although leaving it more time normally allows it to reach a better solution. An anytime POMDP solver, for example, can return a not yet converged policy.

backup In the POMDP policy calculation, backup refers to updating the values (expected reward) for the states, using the values of the future states. The Bellman Equation can be used as backup function in MDPs and POMDPs to update the state values.

base In the hide-and-seek game the base is the starting location of the seeker, while the hider wins the game when reaching the base without being caught.

belief Since in POMDPs the state is not known, a probability distribution, the belief, is maintained. In the case of a discrete state space, a probability $b(s)$ over each state s can be maintained. In the POMCP model, the belief is stored as a list of possible states, which makes it easy to have continuous states (like in the CR-POMCP). The belief is used in this work to refer to the probability of the location of the tracked person.

belief space The space of all belief is an infinite space with $|S|$ dimensions, S being the set of discrete states.

Bellman Equation The Bellman Equation [Bellman, 1957, Eq. 4.2] is used, for example, to calculated the expected reward for the MDP or POMDP models.

cell A (grid) cell refers to one element of a discrete grid. In this thesis we use equally sized square cells, but a cell can have any shape.

CR-POMCP Searcher & Tracker A method to search and track a person that uses the CR-POMCP. In [Goldhoorn et al., 2014] this method is called the *Adaptive CR-POMCP Follower*.

Dabo This is a humanoid robot, like his "sister" Tibi. See for more information Section 3.3.

detect To detect a person means that you can localise a person without necessarily knowing which person it is.

dynamic obstacle An obstacle that can move and which is not on the map, for example an agent, a car, a truck, etc.

Dynamic Programming Dynamic Programming (DP) is used to solve complex problems by breaking them into smaller problems and by storing the (partial) solutions. The Value Function of an POMDP, for example, is calculated with Dynamic Programming, using the Bellman Equation.

explore Exploring can be done to find something or someone, or to discover unknown environments. In the search-and-track task, we do exploration in order to find the person, which can be done with one agent or more.

follow To follow a person means staying close to him/her, at the back-side of the person, trying to staying in his/her path. Here we also use tracking to indicate that we follow the person.

goal A goal, in this thesis, is a destination point on the map, where the agent is assigned to go to. This goal can, for example, be the location of the person or a location to explore the environment.

HB-CR-POMCP Searcher & Tracker A method to search and track a person that uses the Highest Belief over the belief of the CR-POMCP. In [Goldhoorn et al., 2014], this method was called *Adaptive HB-CR-POMCP Follower*.

HB-PF Searcher & Tracker A method to search and track a person that uses the Highest Belief over the belief maintained with a PF.

hide-and-seek A game where a seeker wins when it catches the hider, while the hider wins when it reaches the base without being caught. The game ends in a tie if the game has not finished within a maximum time (H).

hider The person that is looked for by the seeker; this name is mainly used in the hide-and-seek game.

Hierarchical MOMDP A hierarchical MOMDP that groups neighbouring states, to reduce the number of state at the top level. The policy is learned for the top MOMDP model.

highest belief point A point in the belief grid, which has the highest probability. Note that we use a discrete state, or—in the case of continuous states—a point in a discretized belief space.

marker A marker, also called tag, is a visual pattern that is used to recognise the person.

mobile robot A robot that can move itself in its environment, using wheels or legs, for example.

Monte-Carlo Monte-Carlo methods normally use random sampling to approach a desired function.

multi-agent A multi-agent system (MAS) is a system that has several interacting (intelligent) agents.

Multi-agent HB-PF Explorer A multi-agent method with distributed coordination that uses the HB-PF and an explorer to search and track a person.

non-adversarial Non-adversarial games do not include an opponent that wants to flee for any of the agents, e.g. search-and-track.

obstacle An obstacle blocks an agent's path and its vision, so the agent cannot pass nor see through.

off-line Off-line learning methods are method for which the solution is learned before it is being used, for example the policy in RL.

on-line On-line learning methods refer to methods that learn while they are being used, in contrast to off-line methods.

person In this thesis, in most cases, a person refers to the tracked person.

policy In Reinforcement Learning models, such as MDPs, the policy π indicates the best action to do in a certain state (for MDPs) or belief state (for POMDPs).

pursuit-evasion An adversarial game, like hide-and-seek, where the pursuer is trying to catch the evader, while the latter is trying to flee.

Random Hider A hider that takes random actions.

recognise To recognise a person means that she/he is detected and also identified.

robot A robot is a machine that can carry out a complex series of actions automatically. In this work, robots are mobile robots that act like a seeker agent.

search Estimation of the position of the tracked person, also known as *one-sided search*.

search-and-rescue An application of the search problem, where a person is rescued by one or more agents.

search-and-track A scenario where one or more agents look for a tracked persons, and when found, they track him/her.

See All Follower A follower that sees all, also through obstacles.

seeker The agent that looks for the hider; this name is mainly used in the hide-and-seek game.

Simple Follower A follower that goes to the last position were the person was seen.

Smart Hider A heuristic hider that greedily chooses the best action (subsection 4.8.2).

Smart Seeker A heuristic seeker that greedily chooses the best action (subsection 4.8.1).

solver A solver calculates the policy for a Reinforcement Learning model.

state space The set of possible states; each state can have one or more variables, and each state variable can be continuous or discrete. The state space S then contains the Cartesian product of the state variables (S_A, S_B, S_C, ...): $S = S_A \times S_B \times S_C \times \ldots$

static obstacle An obstacle that is not dynamic, i.e. does not move. In this work, the locations of the static obstacles on the map are known beforehand.

target The person, object or animal that is being searched for.

Tibi A humanoid robot (Dabo's "sister") with a Segway as base, cameras, laser range scanners, a movieble head, moveable arms, etc. See for more information Section 3.3.

track To keep track of the person means to know where the person is continuously, but here we also use it to indicate that we follow the person.

tracked person The person for whom the seekers are looking and whom is being tracked.

Value Function In an MDP or POMDP model the Value Function $V(s)$ gives the expected reward for the state s.

Chapter 1

Introduction

Service robots have gotten more interest in recent years, especially since technology is now able to create humanoid robots which are able to navigate with people. Robots will be able to assist people in their daily life, either assisting with household tasks, guiding people, or even rescuing humans. In all of these tasks it is important to be able to find the person, and in many assistive tasks, the following (tracking) functionality is important.

In this work, we develop a *searching* and *tracking* behaviour for mobile service robots that can work in urban environments. However, the presented search-and-track methods can also be applied to search and track objects, or to search-and-rescue in a destroyed building, for example. But, depending on the task and environment, a different robot should be used.

For us humans, both searching and tracking seem relatively easy tasks, however, they require several cognitive tasks, such as planning, navigation and reasoning. When we want to teach these tasks to a young child or a robot, we realize that other issues have to be taken into account. We need to know about occlusions, the environment, navigation and the movement behaviour of the person we are looking for. Furthermore, working in the real world with robots means having to handle noise of the sensors and actuators, false negative and false positive sensor detections and other unexpected changes in the environment, such as other moving people or obstacles.

In this work, we present several methods to search and track a person with a mobile robot. First, we start with a simplification of the problem, the hide-and-seek game,

after which, we study the problem to more realistic urban environments. Finally, we introduce multi-agent cooperation, by having several robots execute the search-and-track task.

The rest of the introduction discusses the motivation, the problem definition and constraints, the objectives and main contribution, and we finish with an overview of the thesis.

1.1 Motivation

In order to make humanoid service robots be able to serve a person, the robot should be able to find him/her. At the same time, the robot should be able to follow (track) a person to serve him/her. There are several applications of searching persons, such as to:

- bring the person something (e.g. a package, or food in a restaurant [Cheong et al., 2016]);
- inform the person of something [Satake et al., 2013];
- find the person to guide him/her [Garrell and Sanfeliu, 2012];
- search-and-rescue [Micire, 2008].

A simplification of the real-world problem of finding people is the hide-and-seek game or pursuit-evasion [Chung et al., 2011], where there are one or more agents searching and one or more hiding. These are well known games, which have been used in a large amount of, mostly theoretical, works to test and compare planning algorithms.

Following or tracking of a person might seem to be an easy task, but it requires the robot to see the person, know where the person is going to, or both. Continuous vision and recognition of the person by the robot is difficult because of: obstacles—either static or dynamic—blocking the robot's vision; and sensory noise. Tracking and following is also an important social robot behaviour when helping the person (carrying something for him/her, for example), or having the person guide it to another place.

1.2 Problem Definition and Constraints

Before explaining the problem in depth, we will give some definitions and problem constraints.

1.2.1 Definitions

The main goal of this thesis is to develop and try several methods to search and track a person with one or more social mobile robots. While finding these methods, we focus on minimizing search time for searching, and the objective for tracking is to get close to the tracked person and keep him/her visible as much time as possible.

In this work, we use the terms *person*, *tracked person* or *target* for the person that is being searched for, and *agent*, *seeker* or *robot* for the searcher and follower. We also refer to agent as robot, since we use a mobile robot to do the searching and following. Moreover, we focus on finding strategies for the seeker and tracker agent, but we do not focus on the tracked person or hider. In the hide-and-seek game, the robot is the seeker and the person the hider. Finally, the terms track and follow are also used for the same action: trying to keep a minimum distance to the person and trying to have him/her in the field of view as long as possible.

1.2.2 Problem Constraints

This subsection describes the assumptions made for the model in simulation, and the limitations we came across while testing our model in real-life scenarios. There are at least two types of problem constraints: the first derives from the robot's perception and actuators; the second is the result of human behavioural reactions to the robot's instructions. The effects of these limitations on our study are summarized below:

- **Safety:** Although the robots have sensors to detect close obstacles or people, they are not guaranteed to see the person, since the sensors have a limited field of view, therefore, restrictions were put on the maximum speed and the distance to (dynamic) obstacles:
  - For safety reasons the robots were not allowed to go faster than around 1 m/s.

  - The robots were kept at a minimum distance of people, other robots, and any detected obstacles.
  - The person being followed was asked not to walk too fast (i.e. less than 1 m/s).

- **Map:** we assumed the map of the environment to be known beforehand in order to plan and predict.
  - The location of static obstacles were known (walls, doors, objects, etc.).
  - The size of the obstacles were measured in discrete cells, in order to do not have too many states for planning (since the map is a discrete grid).
  - The maps were assumed to be flat, i.e., we did not assume there to be any height difference in the area where the robot navigated.

- **Vision:** the robot's vision is limited, and since our main-focus is not computer vision we made some simplifications:
  - To recognise the person we made use of an artificial tag since our research was focused on the combined searching and tracking.
  - The orientation of the robots was not taken into account to reduce the state space and therefore, simplified the planning; for that reason a 360° view was assumed.
  - The static obstacles were assumed to occlude everything and did not allow to let anyone pass.
  - Dynamic obstacles were simulated to only occlude like an obstacle, but they did not block the robot's path, i.e. they could pass through, in order not to make the simulator too complex.

During the different experiments other specific problem constraints were considered, which will be discussed in each experimental section of this document.

1.3 Objectives and Main Contributions

As has been mentioned before, the main goal of this thesis is to search and track a specific person in a known urban environment using one or more mobile robots, taking into account the following issues:

1. **Handling large maps**: the robot should be able to search-and-track on large maps, such as the Barcelona Robot Lab on the North UPC campus (Section 3.1).
2. **Dynamic environments**: the algorithm should be able to handle dynamic environments in which obstacles or persons can move, i.e. appear and disappear.
3. **Continuous space**: the robot should work in the continuous space, instead of discrete grid space, to handle real-life environments.
4. **Multiple agents**: searching for several persons or use several agents to search-and-track (cooperation).

Now, we will comment the contributions shortly.

1.3.1 Hide-and-seek model

First, we focus on a simplification of the search-and-track problem: the hide-and-seek game, in which the seeker has to catch the hider, while the hider can win by reaching the base; the game ends in a *tie* when the maximum time is reached. We propose several models to play the game as a seeker, see Chapter 4. The methods, simulations and real-life experiments are published in:

- Goldhoorn, A., Sanfeliu, A. and Alquézar, R. (2013a). Comparison of MOMDP and heuristic methods to play hide-and-seek. In Gibert, K., Botti, V. J., and Bolaño, R. R., editors, *CCIA*, volume 256 of *Frontiers in Artificial Intelligence and Applications*, pages 31–40. IOS Press. https://doi.org/10.3233/978-1-61499-320-9-31
- Goldhoorn, A., Sanfeliu, A. and Alquézar, R. (2013b). Analysis of methods for playing human robot hide-and-seek in a simple real-life urban environment. In *ROBOT (2)*, volume 253 of *Advances in Intelligent Systems and Computing*, pages 505–520. Springer. https://doi.org/10.1007/978-3-319-03653-3_37

1.3.2 Hierarchical MOMDP

While creating hide-and-seek methods that make use of the off-line Reinforcement Learning model Mixed Observable Markov Decision Process (MOMDP), no large maps could be used due to the computational complexity. Therefore, we propose an on-line hierarchical model that reduces the number of states. This is explained in detail in Section 4.7 and it is published in [Goldhoorn et al., 2013a,b].

1.3.3 Search-and-track models

After having gained knowledge of the limitations of RL methods and the hide-and-seek game, we focus on larger environments in continuous state space, which is explained in Chapter 5 and published in:

- Goldhoorn, A., Garrell, A., Alquézar, R. and Sanfeliu A. (2014). Continuous real time pomcp to find-and-follow people by a humanoid service robot. In *Proceedings of the IEEE-RAS International Conference on Humanoid Robots*, pages 741–747. https://doi.org/10.1109/HUMANOIDS.2014.7041445
- Goldhoorn, A., Garrell, A., Alquézar, R., and Sanfeliu, A. (2017b). Searching and tracking people in urban environments with static and dynamic obstacles. *Robotics and Autonomous Systems*, 98(Supplement C):147–157. https://doi.org/10.1016/j.robot.2017.06.005

1.3.4 CR-POMCP

Since the previous model, MOMDP, could not be used for large maps, we decided to use a Partially Observable Monte-Carlo Planning (POMCP) model, which makes use of Monte-Carlo simulations. The proposed method is the Continuous Real-time POMCP (CR-POMCP) model, which works on-line; it is explained in Section 5.5 and published in [Goldhoorn et al., 2014, 2017b]. Also some other variants of the CR-POMCP method were presented, since the RL method alone did not work well enough on the real robot.

1.3.5 Particle Filter

Instead of using the position estimation of the CR-POMCP, we developed a method that is based on a Particle Filter (PF). We propose a PF version that takes into account the lack of observations, i.e. not seeing the person. This method is detailed in Section 5.7, and published in [Goldhoorn et al., 2017b].

1.3.6 Multi-agent Search-and-track

Finally, to comply with the last objective, we propose a cooperative distributed method to search-and-track with several agents, using either the PF or CR-POMCP method. This method is explained in Chapter 6 and it is published in:

- Goldhoorn, A., Garrell, A., Alquézar, R. and Sanfeliu A. (2016c). Un nuevo método cooperativo para encontrar personas en un entorno urbano con robots móviles. In *XXXVII Jornadas de Automática*, pages 206–213, Madrid, Spain.
- Goldhoorn, A., Garrell, A., Alquézar, R. and Sanfeliu A. (2017a). Searching and tracking people with cooperative mobile robots. *Autonomous Robots*. https://doi.org/10.1007/s10514-017-9681-6

1.4 Thesis Overview

In this thesis, the search for a search-and-track method for real mobile service robots is discussed, including the encountered problems. This thesis exists out of several chapters; here, an overview of the contents of each chapter is given:

- **Chapter 1** gives an introduction of the problem, indicating the motivation, objectives, main contributions, problem constraints and derived publications.
- **Chapter 2** reviews the state of the art of search-and-track, search, track, hide-and-seek, and related games and problems.
- **Chapter 3** details the experimental settings, such as the used robots, the environments where the experiments were done, and the used hardware and algorithms.
- **Chapter 4** defines our first hide-and-seek methods using an MOMDP for a small discrete map. Also a Hierarchical MOMDP method is presented, which reduces the amount of states by projecting a group of states to a *top state*. The method was tested in simulation and in a small outdoors environment with real people.
- **Chapter 5** presents search-and-track methods that are able to work in large environments and in continuous state space. To handle these larger environments, a RL method—that does Monte-Carlo simulations—was used: Partially Observable Monte-Carlo Planning (POMCP). We adapted the method for continuous space and we run it on-line, resulting in the Continuous Real-time POMCP (CR-POMCP). When testing the RL method with the real robot, several problems

were discovered, which resulted in adaptations of the method where the probability map (belief) of the person location was used to find the person. The location with the Highest Belief was used as goal for the robot to go to. A second method to maintain a probability map is based on the Particle Filter (PF). All methods were tested extensively in simulations, and the best were tested in real-life experiments in large urban environments.

- **Chapter 6** presents a multi-agent method that is able to search and track cooperatively, and at the same time is able to work on an individual level, thereby, making it more robust. The method explores the most probable locations of the person, according to the belief. Simulations with up to five agents were done, and real-life experiments with two mobile robots showed real cooperative search-and-track behaviour.
- **Chapter 7** gives the thesis' conclusions and future work.

Chapter 2

State of the Art

Search-and-track is an important robot behaviour, for example, for Human Robot Interaction (HRI), search-and-rescue, RoboCup soccer and theoretical games like pursuit-evasion. For example, [Satake et al., 2013] proposed a method for a mobile social robot to approach people that were not busy. Kanda et al. [2007] focused on estimating the relationship between children, using the time they spent together, and they tried to generate a long-term interaction with the children by calling them by their names, adapting its interaction to the child, and confiding personal matters. Mitsunaga et al. [2008] tried to learn correct interaction behaviour per person, such as interaction distance, gaze meeting and motion speed. They made use of Reinforcement Learning (RL) with the human's unconscious signals as feedback, such as gazing and movement. Search-and-rescue focuses on tasks to rescue people from disaster areas, as explained in subsubsection 2.2.1.1.

In this thesis, we present methods to do search-and-track with mobile robots in a real-life urban environment. Searching and tracking has been researched for many years, but much less research has been done in the combination, search-and-track. Therefore, we discuss the work done in searching and tracking separately, and thereafter, the combination. First, we comment the different characteristics of search-and-track related tasks in Section 2.1, then, we explain shortly the different search game variants. Finally, single agent and multi-agent approaches to search and track are discussed in Sections 2.3 and 2.4 respectively.

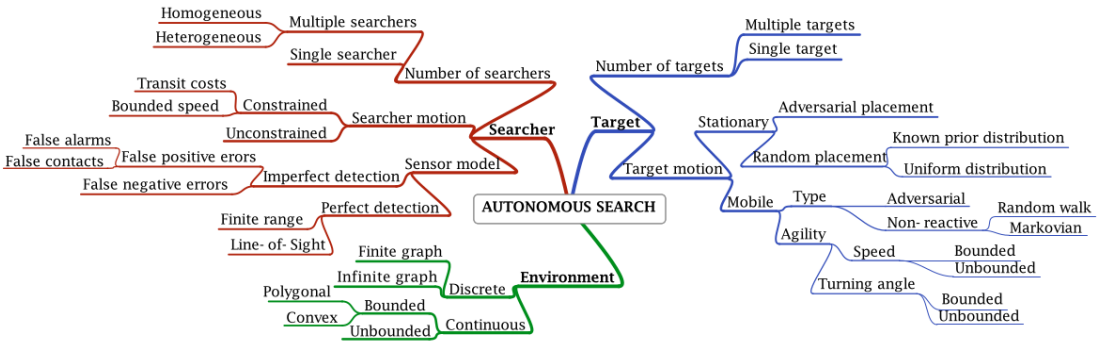


Figure 2.1: The different parameters for autonomous search models, as indicated by Chung et al.. [Taken from Chung et al., 2011]

2.1 Taxonomy and Characteristics

There are a large number of tasks and games related to search, track and search-and-track. Many approaches to solve these tasks are commented in the surveys of [Chung et al., 2011] and [Robin and Lacroix, 2016]. In the survey of [Robin and Lacroix, 2016], they give a taxonomy of the different search-and-track tasks. Following this taxonomy, the main focus of this thesis is *Probabilistic Search* for the searching part, and *Following* for the tracking part.

In the survey of [Chung et al., 2011], they set out the different parameters of the different autonomous search models, as shown in Figure 2.1. The parameters can be divided in *searcher*, *target* and *environment*. The *searcher* is defined by the number of agents to search, their motion model and their sensor model. The environment model can be discrete or continuous, and there can be one or more targets. Finally, the target motion model is very important for the search; a target can be a *stationary* or *mobile* object or a person. Also the target's behaviour is important, a target which wants to flee (i.e. adversarial) is more difficult to catch than a target which moves independently from the seeker (non-adversarial). The characteristics for the search-and-track methods presented in this thesis are listed in Table 2.1.

Table 2.1: The values of the autonomous search parameters indicated by [Chung et al., 2011] for the methods presented in this thesis.

Parameter	Value
Number of searchers	multiple & single searchers
Searcher motion	constrained bounded speed
Sensor model	imperfect detection with false positive and false negative errors
Environment	discrete finite graph
Number of targets	single target
Target motion	mobile, random walk with bounded speed and unbounded turning angle

2.2 Search Variants

In the previous section, we have made a distinction between the different search types and characteristics, here we comment different types of problems related to search-and-track. First, we comment search problems, then, track, and finally, search-and-track.

2.2.1 Search

There are different searching problems, which mainly depend on the behaviour of the person or object to be found, but also on the environment and the seeker's behaviour, as explained in Section 2.1. For example, adversarial search is done in pursuit-evasion problems, while non-adversarial search is done in search-and-rescue and in our problem, search-and-track.

2.2.1.1 Search-and-rescue

Robotic research has the potential to help humans in the domain of Urban Search and Rescue (USAR). In this task, robots support humans to make a situational appreciation of the disaster site and to rescue people. In cases of disasters—such as hurricanes, earthquakes, fires or other dangerous situations—human rescuers should not be put into extra danger, therefore, robots could be sent. For instance, in [Micire, 2008], several dangerous situations are discussed and they show how a remotely controlled robot is used to investigate the situation and to find survivors.

Search-and-rescue researches robots that participate in: 1) finding and rescuing victims in the rubble or debris as efficiently and safely as possible [Murphy, 2003], and

2) ensuring that human rescue workers' lives are not put at great risk [Doroodgar et al., 2010]. Generally, USAR environments are highly cluttered and all robots that operate in these environments do not have a priori information about landmarks in the scene. These conditions make it extremely difficult for robots to autonomously navigate in the scenes and identify victims. Therefore, current applications of mobile robots in USAR operations require a human operator in the loop to guide a robot remotely.

The SHERPA project [Marconi et al., 2012] focuses on search-and-rescue activities using a mixed group of ground and aerial vehicles, and humans. There is also a search-and-rescue track in the RoboCup league [Sheh et al., 2016], which focuses on searching in simulated and real-world environments.

2.2.1.2 Hide-and-seek

Johansson and Balkenius [2005] presented a simplification of the real-world problem of finding people, the hide-and-seek game, where there is an agent seeking, and there are one or more players hiding. The hide-and-seek game requires a high number of cognitive functions, including search and navigation to find suitable hiding places; context sensitive control of attention, depending on the phase of the game; sequentiation of behaviours in relation to the structure of the game; coordination with the other players; emotional control and control of emotional reaction; anticipation of the behaviours of others; verbal and non-verbal communication

Moreover, many works on RL use the game *tag* as a benchmark to compare POMDP solvers [Araya-Lopez et al., 2010, Kurniawati et al., 2008, Ong et al., 2010]. This game exists out of a robot that has to tag one or more fleeing agents. In this thesis, we have used the hide-and-seek game to create a controlled experimental environment for searching and tracking.

2.2.1.3 Pursuit-evasion

In the pursuit-evasion game, also called adversarial search, a *pursuer* tries to catch an *evader*, while the latter tries to escape. The evader is often treated as having a worst-case behaviour—i.e. it has infinite speed and complete knowledge about the pursuer and the environment—but, for practical applications (e.g. mobile service robots) an

average-case behaviour—i.e. having a model of the pursuer and trying to minimize the catch time—is better [Chung et al., 2011, Hollinger et al., 2010]. The game can be extended to a multi-agent problem, having several *pursuers* that try to catch one or more *evaders* [Robin and Lacroix, 2016], where the game is used to study cooperative multi-agent systems. Pursuit-evasion is related to the hide-and-seek game, but in the latter the evader does not necessarily start hidden, nor is there a goal to go to.

There are several variants of pursuit-evasion [Chung et al., 2011, Robin and Lacroix, 2016]:

- In the *edge searching* problem [Borie et al., 2011], several pursuers have to clear a graph from several fast evaders.
- In the *hunter-prey* problem [Abed-alguni et al., 2015], the hunters and preys start randomly on a grid map, and they can only move in four directions.
- In the *cop-and-robber* game [Chung et al., 2011], the agents know each other's positions, and the problem's focus is to calculate the minimum number of cops necessary to catch the robber.
- In the *hunter-and-rabbit* game, the agents can not observe each other.
- In the *lion-and-man* game, the lion has to catch the man, and the agents have a maximum speed. It is originally played in a circular arena without obstacles. A strategy is to stay at the same radius as the man while trying to move closer. Also different environment shapes have been tried, and according to [Bhadauria et al., 2012], three lions are enough to capture a man in any polygon-shaped environment.
- In the *competitive search* problem, several teams of searchers compete on locating a static object [Otte et al., 2016].

2.2.2 Track

Robin and Lacroix [2016] define following as using a single agent to search a single target. In other works and in this thesis, however, we use tracking and following for

the same: seeing the target and being close to the target using either one or more agents.

Tracking has been extensively discussed in different areas, for example the SPENCER project has as goal perception and tracking of people [Linder et al., 2016]. Furthermore, following can also be done to navigate through populated environments [Stein et al., 2013]. Many works that treat guiding and following do not handle situations in which the person to be followed is hidden, at most partly occluded [Martinson, 2014].

2.2.3 Search-and-track

In contrast to pursuit-evasion, in search-and-track the target is non-adversarial, i.e. we assume the person to not escape from the seeker. Still, the target may not know that he/she is being searched for.

Searching and tracking of a ball is done in the RoboCup soccer challenge [Ferrein and Steinbauer, 2016]. They have several leagues of different sizes, in which a team of robots plays soccer against another team. Their final goal is to win against the FIFA world championship with a team of humanoid robots by 2050.

2.3 Single Agent Approaches

In this section, several single agent approaches to tackle the search, track and search-and-track problems are discussed.

2.3.1 Search

First, we discuss several approaches related to searching and *locating* a target—which, in the most cases, is a person, but also can be an object.

2.3.1.1 Reinforcement Learning

In our first method (Chapter 4) we use RL to learn a strategy for the robot to play hide-and-seek. First, a model of the game is defined, which is then used by the RL solver to calculate a policy, i.e. the best action to do in each state. More about RL models and solvers will be explained in Section 4.2.

Partially Observable Markov Decision Process: A Partially Observable Markov Decision Process (POMDP) is used to define a problem, which contains states (e.g. the location of a target), actions (to change the state) and rewards (indicate how good a state is). The rewards guide the learning process, for example, having found the target would be a high reward in the search problem. Since the full state is not observable (because the target is not always visible), observations are used to calculate a probability map over the state space (the belief). Many works, like ours, use POMDPs to model the search problem.

POMDP models have been successfully applied to various robotic tasks [Cassandra et al., 1996, Spaan and Vlassis, 2004], but unfortunately, computing an optimal policy exactly is generally intractable [Papadimitriou and Tsisiklis, 1987], because the size of the belief space grows exponentially with the number of states. And at the same time, looking one time step further into the future, requires considering not only each possible action, but also each possible observation. Hence, the cost of computing the solution for a POMDP grows exponentially with the desired planning horizon.

In many works that present POMDP models and solvers, the game *tag* is used as a benchmark. In a game with two players, the state is the cross product of the possible locations of both players and an extra state value *tagged*; for a field of 29 free cells this gives 870 states. To arrive to a good policy, the Point-Based Value Iteration (PBVI) solver took 50 hours [Pineau et al., 2003]; the solver of [Spaan and Vlassis, 2004] took slightly less than half an hour; the Successive Approximations of the Reachable Space under Optimal Policies (SARSOP) solver took 6 s [Kurniawati et al., 2008]; and the SARSOP solver with the Mixed Observable Markov Decision Process (MOMDP) model took less than 5 s [Ong et al., 2010]. For a tag game of two seekers, on a field with 24 free cells, the number of states is 14 400. Finding a good policy for SARSOP took on average 1.5 hour, yet when using an MOMDP it took about 0.5 hour. Also [Araya-Lopez et al., 2010] used MOMDPs to simulate the hide-and-seek problem, however they only used small maps of up to 7×7 cells.

Other RL Approaches: Uyttendaele [2009] tested a simple reactive seeker and one that uses a Markov Chain. Adding memory to an otherwise reactive system was shown to improve performance by [Wasson et al., 1997].

2.3.1.2 Optimal Search Theory

Search theory started in the years 1942-1945 with the work of [Koopman, 1946] in the Anti-Submarine Warfare Operations Research Group of the U.S. Navy. The *Optimal Search Theory* was introduced by [Stone, 1975], and focuses on: 1) an efficient search plan to find the target within a certain time; 2) thereby, estimating the time necessary to find the object.

Stone [1977] focused on rescue on sea, as example case they used the search of a ship in trouble, which does not know its exact location. The objective was to search as fast and efficient as possible with the available resources. In their method, the search space is divided in cells, of which Stone comments: "Devising a probability distribution is an art rather than a science". An initial probability of the cells (i.e. a belief) is given and then propagated using a Monte-Carlo method.

Kratzke et al. [2010] explained the technology used by the US Coast Guard for search-and-rescue operations: the *Search and Rescue Optimal Planning System* (SAROPS). The planner iteratively tries to improve the localization, using a Particle Filter (PF) and environmental information (wind, drift, temperature, etc.).

2.3.1.3 Cognitive Approaches

There are different works that use hide-and-seek to study cognition [Kennedy et al., 2007, Trafton et al., 2006]. A method that uses a cognitive architecture, Adaptive Control of Thought-Rational (ACT-R) [Anderson et al., 2004], was presented in [Trafton et al., 2006] to play hide-and-seek. Their architecture is layered, in which the lower layer contains a navigation module and the top layer the cognitive architecture. The created ACT-R module learns how to play hide-and-seek, generating some new rules. Their research is based on hide-and-seek experiments with a 3.5 year old child. In [Kennedy et al., 2007], they presented a method where a human and robot cooperatively follow an other person. Both methods focus on cognitive methods, which require a high amount of symbolic knowledge of the world.

2.3.1.4 Other Approaches

Bhattacharya and Hutchinson [2010] calculate strategies, for the pursuer and evader, that are in Nash equilibrium, starting from their end positions. Both agents have bounded speed and know the map.

Also deep learning is used to find locations with a robot, for example in [Caley et al., 2016], they presented a method that learns the exit locations of a building, based on a data set of hand labelled floor plans. As comparison, they used frontier exploration and a method that uses Histogram of Oriented Gradients (HOG) features with a Support Vector Machine (SVM) as classifier. Using a set of new floor plans, they found that the error, distance to the real exit, was least (a few meters) for the deep learning method.

Lau et al. [2006] used a branch-and-bound approach to efficiently and quickly find a moving non-adversarial target in a connected region environment. Their goal is to plan a path that maximizes the probability of finding the target in minimum time.

The search of a static object in an unknown environment is discussed in [Kulich et al., 2016]. The problem is treated as an instance of the Graph Search Problem (GSP), where the agents can not see the neighbour vertices. Their approach uses the Greedy Randomized Adaptive Search Procedure (GRASP), which iteratively gets a feasible solution and then applies a local search step (for example, exchanges two vertices in the solution path). They propose two meta-heuristics that are used in the first GRASP step to calculate the path and three local search operators, used in the second step.

2.3.2 Track

There are many works on tracking and following, in some of them tracking only refers to knowing where the target is, whereas following refers to going the same path as the target and staying close to him/her. In this thesis, we use tracking to refer to both of them.

2.3.2.1 Filters

Bayesian filters, such as the Kalman Filter (KF) and the Particle Filter (PF), have been used to keep track of states, such as target or robot tracking [Thrun et al., 2005]. None

of these methods explicitly search for a person, nor do they mention keeping track of people when they have been hidden for a long time.

Kalman Filter and Extended Kalman Filter: In [Linder et al., 2016], they compare different algorithms to track people with a mobile robot in a busy airport terminal. They found a method that uses nearest neighbour and an Extended Kalman Filter (EKF) to work best.

Lian et al. [2015] tracked a person in a dynamic environment by trying to maximize the visibility of the target. Their method first uses a laser rangefinder and an EKF, then a look-ahead algorithm to follow the target and avoid obstacles at the same time.

Luber et al. [2011] combined a multi-hypothesis tracking that uses a KF with an on-line detector that uses Red Green Blue Depth (RGB-D) data. Their experiments were done in a crowded indoor environment using three Microsoft Kinect sensors.

Particle Filter: Many works make use of a PF [Thrun et al., 2005] for tracking, since it is a fast algorithm, its complexity mainly depends on the number of particles and it allows for any distribution, unlike a KF, which normally has a Gaussian distribution. Glas et al. [2009] used several fixed laser rangefinders to keep track of a person using a PF. Oyama et al. [2013] presented a robot that tracks visitors' positions in a museum guide tour. In [Montemerlo et al., 2002], a PF was used by the robot to track a large distribution of person locations, conditioned upon a smaller distribution of robot poses over time. Their method compares the measured distance, using the laser range scanner, to an object with the expected distance on the map. Also for the person's location, the measured distance to the person is compared to the expected distance, using a ray tracing algorithm. Choi et al. [2011] used Kinect sensors and a variant of a PF to keep track of several targets. Arulampalam et al. [2002] discussed different PFs for online nonlinear/non-Gaussian Bayesian tracking.

Authors of [Brscic et al., 2013] treated tracking of people in large public environments, where they used multiple 3D range sensors mounted at above the human height, to reduce occlusions. Each tracked person is assigned an identifier if he/she has been visible as a new cluster of particles during several steps; if the dispersion of the cluster

gets too high, the person gets deleted, and is recovered when the person gets detected close to the identifier. Experiments in a shopping centre showed good results.

Khan et al. [2005] used a PF to track several agents and a Markov Random Field (MRF) to model target interactions and maintain the identity. To handle the exponential complexity of the MRF, they used Markov Chain Monte-Carlo to do the sampling, which acts as different PFs when the agents are far and interacts when they are close. A reversible-jump method is used to handle changes in number of agents to track.

2.3.2.2 Other Approaches

Bandyopadhyay et al. [2009] handled tracking of a person in a crowded environment. They compared a greedy algorithm to an algorithm that uses a POMDP to model the target's behaviour. They showed that the latter resulted in more efficient tracking, and it could lose the target for some time while still continuing to track it.

In [Tipaldi and Arras, 2011], a model that represents human activity events was introduced. A Hidden Markov Model (HMM) is used to predict people's motion, to track them and to adapt the robot's navigation behaviour. Their experiments were done in a simulated office-like environment.

A blind guiding robot, created by [Martinson, 2014], used a Kinect to detect if the person was following. Gaussian Mixture Models (GMMs) were used for the person location and for obstacle locations, which were trained by hand-classified objects. Their system was able to detect the person, even when some occlusions occurred. Also [Shu et al., 2012] were able to cope with partial occlusions while tracking multiple persons using fixed cameras. They used SVM classifiers that detect humans.

2.3.3 Search-and-track

Volkhardt and Gross [2013] proposed a method to search and locate a person, thereby, using an occurrence map, which indicates the most probable positions of the person on the map. Their search method is guided by a list of predefined navigation points, which are chosen based on the closeness and probability of occurrence of the person; the points are marked as visited if the person is not found. Their method assumes there to be one person, and if there are more, the robot goes to the firstly detected

person. The verification—in case of a false positive detection—is done by waiting for the person to interact with the robot. They did experiments with a real robot in a scenario with three rooms where the robot was able to locate the person up to 91% of the time.

Granata and Bidaud [2012] used fuzzy rules to decide which strategy to use to search-and-track the person. When the person is not visible, exploration is done to find the person. When found, an EKF is used to track the person's location and speed. And when the detection of the person is lost, the robot uses the EKF, which estimated the person's trajectory. However, this can not be done for a long time, since the EKF might use an incorrect prediction. In the later case, or if the robot does not see the person, it uses fuzzy rules to go to the location which has been visited the longest time ago.

Foderaro et al. [2016] modelled the Ms. Pacman problem, which is fully visible, and is used as a benchmark problem for pursuit-evasion with multiple active adversaries. They used cell decomposition to generate a convex subset of cells, in which a path can be calculated easily. The connectivity tree was converted and was used to generate the decision tree using a profit function

2.4 Multi-agent Approaches

The task of search-and-track—or any of the related tasks discussed before, in Section 2.2—were explained for one seeker, however, the task can also be done by several seekers. And, in order to execute the task quickly, coordination between the agents is important.

The coordination can be *centralized*, i.e., there is one agent doing the coordination; it can be *distributed*, that means, each agent makes its own decisions; or it can be a hybrid of both, for example having smaller groups of agents that coordinate centralized. Furthermore, the coordination of multi-agent systems requires some sort of communication, which is mostly *explicit*, using radio signals, laser, or vision; or *implicit*, e.g. through leaving marks in the environment [Chu et al., 2007].

Recently, specialized coordination techniques have been published for certain domains. In the context of RoboCup, different coordination behaviours are used in com-

bination with role assignment techniques [Iocchi et al., 2003, Weigel et al., 2002].

Multi-agent teams can be homogeneous, i.e. exists of equal agent types; or they can be heterogeneous [Vidal et al., 2002], which can be an advantage, in search-and-rescue for example.

Like in the previous section, we will discuss several approaches for search, track and search-and-track separately.

2.4.1 Search

Especially in multi-agent search cooperation is important, therefore, games like pursuit-evasion are often used to to test multi-agent cooperation, such as in [Abed-alguni et al., 2015, LaValle et al., 1997].

2.4.1.1 Reinforcement Learning

Monte-Carlo Tree Search (MCTS) [Kocsis and Szepesvári, 2006] is used in [Nijssen and Winands, 2012] to play a hide-and-seek variant called *Scotland Yard*, this is a board game with one hider and several seekers.

Bilgin and Urtis [2015] used Q-learning—a temporal difference learning without predefined state transition probabilities—to learn the policy of the pursuer. Their simulations were done in a 9×6 cells environment, and for static evaders, the pursuers found an optimal path quickly, but they were not able to catch a random evader. When both the evader and pursuer used Q-learning, they were able to avoid each other.

Abed-alguni et al. [2015] used aggregated Q-learning to learn the policy for several pursuers. The agents were separated in consultants, who assigned sub-problems to the agents, tutors and workers, resulting finally in a Q-table. The method was shown to be faster than normal Q-learning.

Liu and Zeng [2006] presented a multi-agent cooperative RL method, which tries to improve the long-time reward. They only focus on learning the agent's own action, while also observing the others' actions. The computational cost is reduced, because the agents do not have to learn only its own actions. Simulations of pursuit-evasion were done with four agents.

Awheda and Schwartz [2016] developed a formation control mechanism to find a superior evader (higher or equal speed than the pursuer), using several pursuers. They proposed a fuzzy RL method to learn the control strategy of the pursuer, thereby guided by a formation control approach, which guarantees the pursuers to be distributed around the evader. They verified their method with simulations.

2.4.1.2 Other Approaches

Hollinger et al. [2010] presented an on-line decentralized multi-agent search algorithm. The maps were converted to a graph, with full vision assumed in each node. Their scheduler creates a path to find a person on a graph for multiple agents, and the path is optimized based on an adversarial and non-adversarial person model. The method of Hollinger et al. keeps track of a list of *contaminated* nodes (areas that not yet have been checked, or where the person could have returned to), thereby, assuring the person to be found. However, to assure a person to be found, a minimum number of search agents are necessary, which depend on the map configuration. In [Hollinger et al., 2015], the focus is on data fusion between the agents, and keeping track of the probability of the person being in each of the vertices. When there is communication, they take into account the other agents' paths, otherwise, after reconnection, the beliefs are fused. They showed two simulated experiments, one in a map, like in the previous mentioned paper, the second in a underwater sea environment, where communications disturbance is a real problem.

Many works on pursuit-evasion use several multi-agents to search for an evader and they calculate the minimum number of agents required to guarantee the target to be found. For example, [Borie et al., 2011, Fomin and Thilikos, 2008] try to clear a graph with a minimum number of pursuers. In [LaValle et al., 1997], the objective is to have the target always in sight, using edge labels to indicate whether an area is *contaminated* (i.e. the target could be there). They extend pursuit-evasion to grantee detection of adversarial agents in polygonal environments. The task is to find a path that guarantees to detect the evader, regardless the evader's path, but more than one pursuer might be required.

Goodrich et al. [2008] discussed the practical problem of search-and-rescue in the wilderness, using UAVs. They proposed a vision method that detects the person using

the camera of a UAV, and they proposed a task and role division, like is done by people who do search-and-rescue.

Vidal et al. [2002] used UAVs and Unmanned Ground Vehicles (UGVs) to pursuit evaders and to build a map at the same time. They tested a *local-max* policy, which only takes into account neighbouring cells, whereas the *global-max* takes into account the whole map. The latter was shown to have an upper bound on the expected capture time, depending on the pursuer and environment, whereas the local method did not.

With a team of search vehicles [Wong et al., 2005] presented a method to search multiple targets using multiple agents. Their method maintains a Bayesian filter for each target, and the sum of the probabilities of detecting the targets is used as utility function.

Chu et al. [2007] used pheromones to indicate where searchers have been, and since the pheromones evaporate, lower concentrations indicate that it has been a longer time ago a searcher was there. They showed that, initially, their method was not as efficient as others, but then converged to a better performance.

Vieira et al. [2009] presented a method that assigns teams to pursue an evader, based on the cost, with complete knowledge of the players. They guaranteed termination, and optimized for minimum search time. An optimal strategy was computed off-line through a state-based game-graph. Their experiments were done with small robots that were only able to follow a wall, which made them slower than the simulated games, but it showed feasibility.

In [Kolling and Carpin, 2010], they proposed a method for a pursuit-evasion variant, Graph-Clear, without using a map. A decentralized group of pursuers swept the environment, only guided by walls and neighbouring robots. Kolling et al. [2011] presented a method that does searching of moving targets in a real environment of 700 000 m^2, of which a height map is known. A graph of the environment is constructed to decide on the tasks and assign them to the agents. They used eight humans, which received the tasks through a tablet, to search for the targets.

Individual task execution scales well, but lacks coordination of the group, therefore, [Gerkey et al., 2005] proposed to use smaller groups. They introduced the *parallel stochastic hill-climbing with small teams* (parish), in which a value heuristic (based on

a benefit and cost) for each task and agent is used to assign tasks and generate small teams. They tried the method in simulation and with small real robots.

Katsilieris et al. [2013] created a multi-robot system to search-and-secure a potentially adversarial target. They used several mobile robots, and static robots (blockers), to block a path, preventing the target to pass through it. They did experiments with two large ground vehicles in an outdoor field with obstacles, clearing the field op targets.

2.4.2 Track

In this section, we focus on tracking of targets using a multi-agent system.

2.4.2.1 Filters

Xu et al. [2013] used a decentralized robot team to track a visible target, thereby learning the utility models of the robots and negotiating with the other robots. They used an Information Filter (IF), which is a variant of a KF, with as goal minimizing the uncertainty and optimizing the information obtained by the robots. Experiments were done with two Segway RPM robots, one with 360° and another with 180° vision.

Particle Filter: Glas et al. [2015] introduced a tracking algorithm using individual PFs to track multiple entities with multiple robots.

Santos and Lima [2010] explained a cooperative object localization method, which was applied to the RoboCup robots. They used a PF to keep track of the object, thereby exchanging the prior probability distributions using a GMM. To handle measurement uncertainty and disagreements, they used the Expectation Maximization (EM) algorithm to approximate the posterior distribution of the ball's position.

Ahmad and Lima [2013] tracked a spherical object (ball for the RoboCup soccer challenge) with a team of robots. They used a PF and they shared the observation, observation confidence and the localization confidence. The confidences were used as weights to update the PF. Their experiments were done on a RoboCup soccer field with four mobile robots.

Distributed Particle Filter: To do tracking with multiple agents, the combination of decentralized techniques and PFs has led to Distributed Particle Filters (DPFs) [Hlinka et al., 2013]. DPFs were used by [Gu, 2007, Jiang and Ravindran, 2011, Read et al., 2014, Sheng et al., 2005, Vázquez and Míguez, 2017], who all focused on tracking of one or more people with a Wireless Sensor Network (WSN), which is a large number of connected sensors. Each of these sensors can run the localization algorithm, or a few nodes, which are called Processing Elements (PEs) [Vázquez and Míguez, 2017].

Sheng et al. [2005] used DPFs to localize and track several targets within a WSN, and in order to reduce the information sent between nodes, they used a low dimension GMM. They compared methods that work separately, in sensor groups, and hierarchically. They worked with a previously proposed Centralized Particle Filter (CPF) tracking algorithm, in which the posterior distribution is updated based on all measurements, however, Sheng et al. did it with groups of sensors. Tests were done in simulations on an area of 100 m $\times$ 100 m with 25 fixed sensors and two targets to track.

Gu [2007] used an average consensus filter, which makes the DPF an approximation to the CPF, in which the information is diffused globally by having information exchange with neighbouring nodes only, and the EM is used to estimate the parameters sent from the other nodes. They treated one moving target that was tracked using the sensor grid, which measured the distance to the moving object.

Jiang and Ravindran [2011] designed a faster *Completely Distributed PF* (CDPF) to track one randomly moving target with a sensor network using the neighbourhood. Their method had a higher error rate than the Semi-DPF, but was much faster. In their simulation they had thousands of randomly deployed sensors in an area of 200 m $\times$ 200 m, of which each node could sense until around 10 m.

Read et al. [2014] guaranteed the particle weights to be constructed properly using a *Distributed Resampling with Non-proportional Allocation* (DRNA). Light sensors were used to detect a person, and they showed that having four processing units was four times faster than having one central processing element. Real-world tracking of a person in a 3.2 m $\times$ 6.0 m environment was done with an accuracy of about half a meter.

Vázquez and Míguez [2017] presented a DPF that uses the median posterior probability in order to efficiently combine local Bayesian estimators; in simulations they

showed that their method was more robust.

2.4.2.2 Other Approaches

In [Fleuret et al., 2008], a multi-people tracker using multiple cameras was presented. They used a Probabilistic Occupancy Map, in which they track several persons with high precision, allowing them to be partly occluded.

Charrow et al. [2013] used a team of robots with a range sensor, to localize a fixed radio source. For each robot, a measurement of the distance to the radio source was taken, which was used as input to a PF. The robots had a reading, with noise depending on how many obstacles were between it and the target. They used the entropy to optimize the control strategy for all the robots, reducing the uncertainty of the estimation of the target location. They did experiments with real robots in two environments of up to 40 m $\times$ 35 m.

Surveillance also requires tracking, which was done by [Capitan et al., 2016], with Unmanned Aerial Vehicles (UAVs). They did tracking of multiple targets, thereby assuming them to move independently. They used an MOMDP, combining the target and UAVs locations in the discrete states, and as actions, they had four movement directions and one staying on the same position. For each behaviour (target to track), a different POMDP policy was learned, and each behavior was selected using an auction method. The policies were learned for a reduced state space (a single target for a single UAV), since it is intractable for the combined state space. They did simulations and experiments with small UAVs in a small artificial environment.

2.4.3 Search-and-track

Mottaghi and Vaughan [2007] presented a method to search-and-track with one or more agents. They used a PF to represent the target's probability distribution, guiding the agents by a potential field, generated from the particles. The particles generate an attractive field, while obstacles generate a repulsive field. When using multiple agents, the agents are assigned a subset of all the particles. They did real-life experiments with two Pioneer 3 robots in an office like environment.

Chapter 3

Experimental Settings

In this chapter, the experimental settings are explained, which include the environment where experiments were done, the used material: robots and algorithms, and the safety guidelines and restrictions.

3.1 Environments

This thesis focuses on search-and-track in urban environments with mobile robots, therefore, we have done experiments on several areas, see Table 3.1, of our campuses in the North and South Campus of the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain. The campus is a large urban area in which there are static obstacles, such as building and pillars, but also dynamic obstacles, such as people walking around.

Table 3.1: The list of the maps used during the experiments with some characteristics of the areas, where the *Access.* is the surface without obstacles. Three areas of the UPC campus were used: the FME, the BRL and the Telecos Square (three variants were used).

Name	**Size** [m]	**Size** [cells]	**Surface** [m^2]	**Access.** [m^2]	**Figures**
FME	17.0×12.0	17×12	204	164	4.10,5.8,5.9,5.12(a),5.13
BRL	80.0×15.0	80×15	1200	703	5.10
Tel. Sq. 1	60.0×55.2	75×69	3312	1400	5.11a,5.11b
Tel. Sq. 1a	35.2×21.2	88×53	746	307	5.11c
Tel. Sq. 2	60.0×55.2	75×69	3312	2188	5.12(b),5.12(c),5.14
Tel. Sq. 3	38.4×44.8	48×56	1720	1154	6.9

The smallest environment, the Facultat de Matemàtiques i Estadística (FME) lab

(17 m × 12 m, 170 m^2 accessible; Figure 3.1(a)), is located next to our research institute IRI on the South Campus of the UPC, and was used mostly for the first trials. The FME environment is outdoors, but partly covered by a roof, and since no obstacles were in the field, we placed several artificial ones, as can be seen in Figure 4.10.

The Barcelona Robot Lab (BRL) is on the North Campus (80 m × 15 m, and an area accessible to the robot of about 710 m^2, Figure 3.1(b)) is a long environment with a large amount of pillars, which cause occlusions.

The *Telecos Square* (60 m × 55 m of which 1400 m^2 is accessible; Figure 3.1(c)) is the largest environment, and it contains a square, and two covered areas with several columns. Through all areas people pass by frequently, especially the last, since it is in the centre of the campus. Note that Table 3.1 shows several versions of the Telecos Square map, because in some experiments we used a slightly larger area, which includes a small room.

(a) FME (b) BRL (c) Telecos Square

Figure 3.1: An air photo of the different experiment areas. Taken from: Google Maps, ©2017.

3.1.1 Map Types

In the search-and-track simulations and real-life experiments, environment maps are used to plan the path and predict the person's location. In this thesis, the maps are assumed to be two dimensional, and locations are either *obstacles* or *free*.

Maps are created using the robots sensors and a mapping algorithm, as explained in Section 3.4, which results in the *scanned map* shown in Figure 3.2. Here, the obstacles are black, *free* space is light grey and unknown space is dark grey. This map has a resolution of 10 cm, and was used for navigation of the robot.

For the search-and-track algorithms, we used a discrete *grid map* (Figure 3.2) made by grouping points of the *scanned map*. Here black cellsobstacles are black, light grey are free cells and the dark grey cell indicates the base (for the hide-and-seek game). The cells are either free or obstacle, of around 1 m × 1 m. Obstacles indicate that neither robot nor person can pass through it, nor can see through it.

Finally, a *belief map* was created (Figure 3.2) to indicate the probability of the person being at a certain location. Black cells indicate the obstacles; the rest is free area for which white to red indicates a low to high probability of the person being there, and light blue indicates a probability of 0.

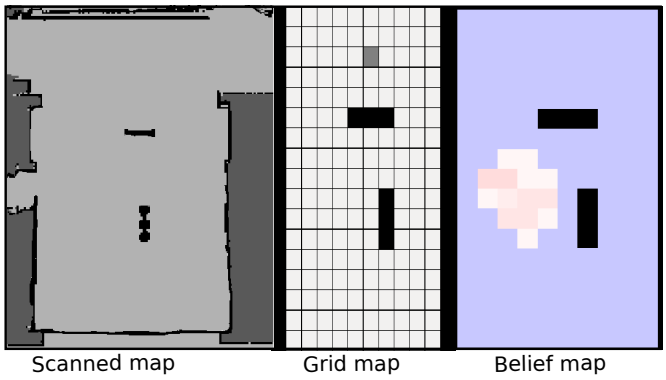


Figure 3.2: The different map types used in the experiments.

3.2 Architecture

The complete architecture of the *search-and-track* system, which is presented in this thesis, is shown in Figure 3.3. The architecture shows the components that are used to search and track, where our contributed components are shown in bold.

In order to search and track, we first need to know the positions of the robot and the visible people, which is done in the *Robot Perception* modules. The *Robot Localization* module (Section 3.4) uses the *Odometry* and *Laser* sensors to localize itself and returns the agent's position $o_{\mathrm{L,agent}}$. To detect all the people in the visible field, the *People Detection* module uses the horizontal range *lasers*, which detects leg shaped objects, see Section 3.5. Next, a tracking algorithm is used to keep track of the detections while they are visible. The list of detected people is later on (in Chapter 5) used as *dynamic*

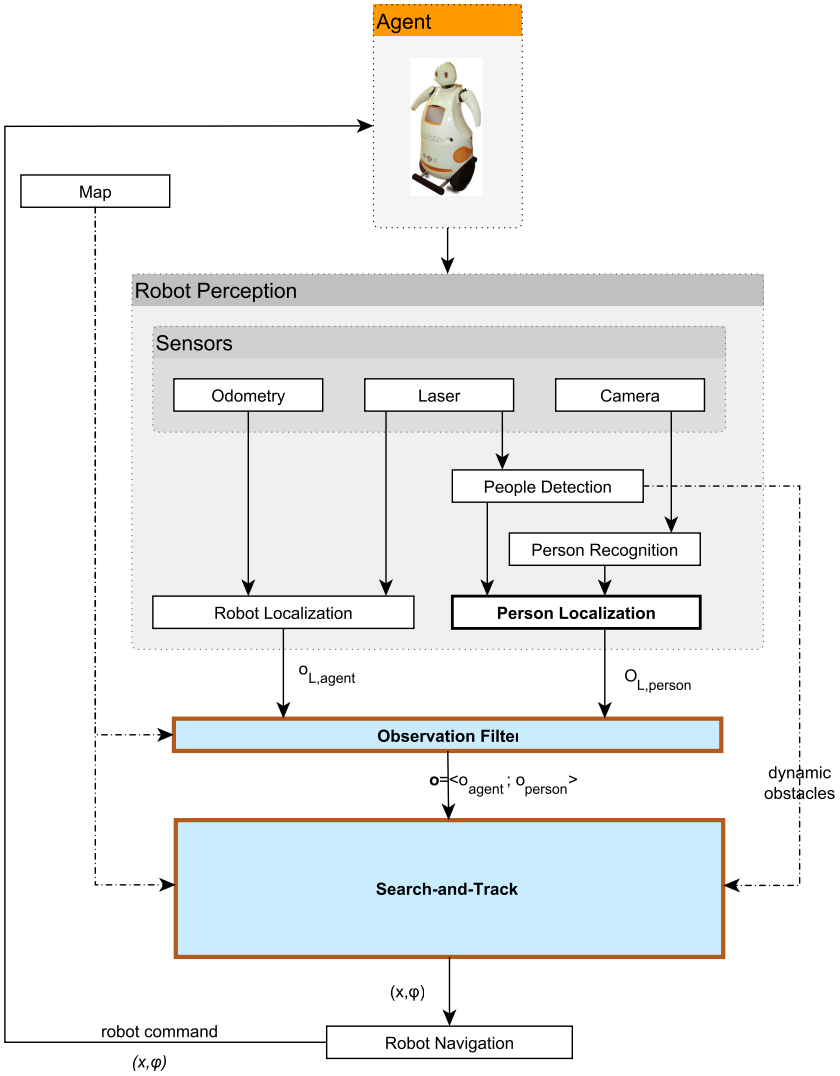


Figure 3.3: The architecture of the search-and-track method presented in this thesis, our contributions are shown in bold.

obstacles in our search-and-track algorithm, and in the *Person Localization* block. To recognise the person we are looking for, we put an AR marker [Amor-Martinez et al., 2014] on the person, see Section 3.5. The *Person Recognition* module detects the AR marker and outputs its position, which is then combined with the location of all the detected people in the *Person Localization* module to generate the observed location of the person $O_{\mathrm{L,person}}$. Note that this observation can be empty if the person is not

detected. The *Observation Filter* module makes sure that the observed robot location and person locations are inside the map, and that they are not inside an obstacle.

Next, the *search-and-track* method is executed using the observations. In the following chapters, Chapters 4-6, different hide-and-seek and search-and-track methods will be presented. Finally, the *Robot Navigation* module (see Section 3.4) calculates the path to the selected goal, which is sent as motion commands to the robot.

3.3 Robots

For the experiments, we used our mobile service robots Tibi and Dabo, which were created during the URUS project [Sanfeliu et al., 2010, Trulls et al., 2011] to work in urban pedestrian areas, and to interact with people. Tibi and Dabo have a two-wheeled Segway RMP200 platform as base, which can work as an inverted pendulum in constant balancing, can rotate on the spot (non-holonomic), and it has wheel encoders providing odometry and inclinometers providing pitch and roll data. Although that, for the experiments we incorporated two additional wheels in front and rear the platform to maintain the platform always vertical.

To perceive the environment, they are equipped with two Hokuyo UTM-30LX 2D laser range sensors used to detect obstacles and people, giving scans over a local horizontal plane at 40 cm above the ground, facing forward and backward. The lasers have a long detection range of 30 m, and a field of view (fov) of 270°, which is limited to 180° for each of the lasers because of the carcass of the robot. Additionally, a distance of about 45 cm between the front and rear laser causes a blind zone. As vision sensor Dabo uses a PointGrey Ladybug 2 360° camera, located on the top of its head, whereas Tibi uses a Bumblebee 2 stereo camera at the front and two Flea 2 cameras at the back, which in total cover much less than 360°, and therefore Tibi has less vision.

As social robots, Tibi and Dabo (Figure 3.4) are meant to interact with people using the following elements: a touchscreen, speaker, movable arms and head, and LED illuminated face expressions. Power is supplied by two sets of batteries, one for the Segway platform and one for the computers and sensors, giving about five hours of full working autonomy. Two onboard computers (Intel Core 2 Quad CPU @ 2.66 and 3.00 GHz with 4 GB RAM) manage all the running processes and sensor signals.

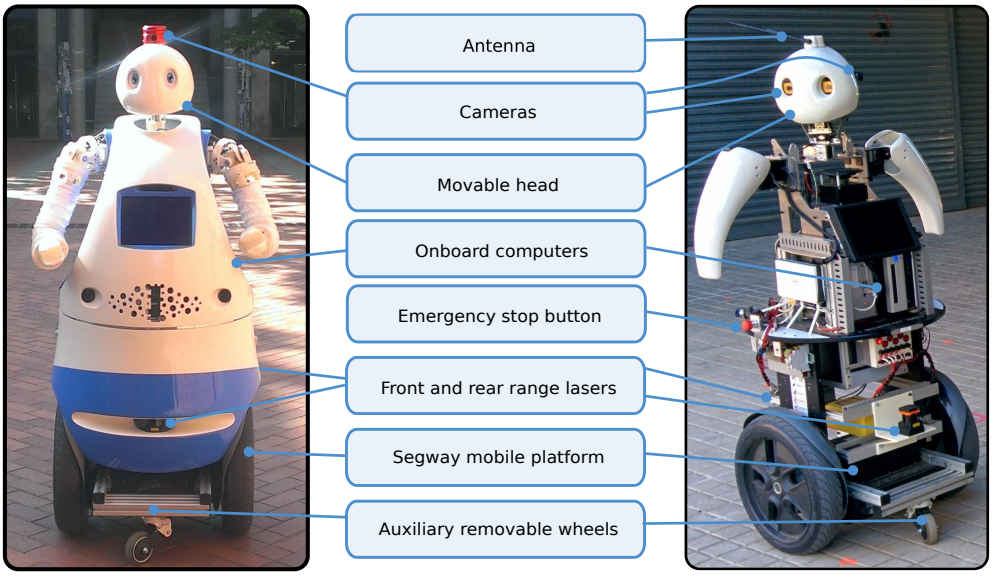


Figure 3.4: The mobile robots, Dabo (left) and Tibi (right), used in the experiments. Some of the important components of the robots are shown. Both robots normally wear a carcass, but in this photo Tibi does not.

An external laptop (Intel Core i5-2430M @ 2.40 and 3.00 GHz with 4 GB RAM) is used for external monitoring, and in some experiments to run the search-and-track algorithm. For the communication between the robots, a mobile 3G connection is used. As Operating System Ubuntu (first version 12.04, and 14.04 in Chapters 4-5) is used; and as middle ware the Robot Operating System (ROS) [Quigley et al., 2009] is used, a software environment for robot system integration that provides a useful and large set of libraries and tools.

3.4 Robot Mapping and Navigation

As can be seen in Figure 3.3, our search-and-track algorithm requires the position of the robot on the map and an algorithm that navigates the robot to a goal position. Although the robot localization and tracking of people can be done simultaneously [Montemerlo et al., 2002], we focus on the searching and tracking of a person, and, therefore, we make use of existing algorithms that do localization and navigation of the robot.

Prior to the experiments, a map was generated by the robot using the range lasers (back and front), with the ROS package GMapping, which implements OpenSlam's GMapping. This is a highly efficient Rao-Blackwellized Particle Filter (PF) that learns grid maps from laser range data [Grisetti et al., 2007]. Although this method can be used for localization and mapping, we did not want to use it during the experiments, because it also can mark persons as being obstacles if they stand still for too long. Instead, we use the Adaptive Monte Carlo Localization (AMCL) approach, also available as ROS package, for localization. This method uses a PF to track the pose of a robot against a known map [Arulampalam et al., 2002].

The robot moved through the environment using a set of navigation algorithms provided by ROS. It consists of a Dijkstra global planner that uses the previously generated map to calculate the shortest path. To avoid colliding with dynamic obstacles, a local Trajectory Roll Out [Gerkey and Konolige, 2008] planner is used, which generates and scores trajectories over a costmap that is updated with laser range data. The inputs of the navigation algorithm are the desired goal coordinates and orientation, see Figure 3.3, which are given by our search-and-track algorithm.

3.5 Person Detection and Recognition

Since our algorithms use as input the location of the person, if visible, we use already existing algorithms that give us this information using several sensors, as shown in Figure 3.3.

First we will discuss shortly how some other works solved the recognition and detection of persons, and then we will explain the method we implemented.

3.5.1 Person Recognition Methods

Many works handle tracking by detecting the persons, without recognising them:

- Luber et al. [2011] used three Kinect sensors to detect people and the Combo-HOD (Histograms of Oriented Depths and Gradients) algorithm (which is based on Histogram of Oriented Gradients (HOG), using depth and colour), and as output has the 3D box of each target.

- Glas et al. [2009, 2015] tracked several people using a PF and up to six laser rangefinders.
- Oyama et al. [2013] tracked people, while guiding them, using a laser range scanner on shoulder height, detecting the oval shape of the person.
- Montemerlo et al. [2002] used a 2D laser rangefinder and a PF to track people.
- Brscic et al. [2013] used several fixed 3D range sensors to detect the head and shoulders of people, using the depth images, and they found it to work better than a laser rangefinder-based detector.

Other works include the recognition of the persons, to improve the tracking system:

- Linder et al. [2016] used a 2D laser rangefinder with a random forest classifier to detect people, and monocular vision with a RGB-D sensor using a Histogram of Oriented Gradients (HOG) detector. The latter is sensitive to clutter and reflections.
- Choi et al. [2011] combined five observations models: HOG, shape from the depth, frontal face detection, skin detection and motion detection.
- Martinson [2014] used a robot with a Kinect sensor at chest level, and a GMM was used to follow the person. They tried several classifiers that could handle partial occlusions.
- Shu et al. [2012] used fixed cameras and then various steps (among which a HOG detector and a Support Vector Machine (SVM) classifier) to detect and track people, and thereby, also handling partial occlusions.
- Volkhardt and Gross [2013] used a HOG, leg, face, motion and body-shape detector to recognise the person.
- Granata and Bidaud [2012] used a laser-based leg detector and two vision based detectors for the whole body and upper body. They combined the outputs using a grid based approach and a GMM.

Figure 3.5: Combined laser (left) and marker detection (right: video image; centre: binarized image).

3.5.2 Our Person Detection and Recognition Method

In the previous subsection we have shown that there are many tracking methods that make use of a laser rangefinder, and a leg detector. We also do this, we use a boosting leg detector [Arras et al., 2007], which provides the position of potential people in the scene, using the horizontal front and rear range laser sensors. A Multiple Hypothesis Tracking (MHT) for Multiple Targets [Blackman, 2004] keeps the trail of the detected people. False positive detections are reduced by filtering out detections that are close to, or inside known obstacles.

A people detection algorithm alone is not enough, because we also have to recognise the person we are looking for. Although some works have used people detection, as shown in the previous subsection, they require either a large amount of computing time [Volkhardt and Gross, 2013] or a 3D camera (such as the Kinect [Martinson, 2014]), which does not work well outdoors. Since this thesis focuses on search-and-track of a person with mobile robots, we decided to use a fast and robust method. First we tried a vision method [Villamizar et al., 2012] to detect the person with the robot's camera, but this only worked within a few meters distance of the robot in controlled light conditions.

Therefore, we decided to use a robust method, Augmented Reality (AR) Markers [Amor-Martinez et al., 2014], which were worn by the person (Figure 3.5). The AR algorithm gives an estimation of the pose of the AR marker with respect to the camera. We used an improved version of this Pose Estimation algorithm [Amor-Martinez et al.,

2014] that, in combination with previous local window binarization, makes the method more robust to outdoors lighting issues. We used for Dabo a Ladybug 360° camera—which internally has five cameras—to detect a marker from any direction, and four separate cameras for Tibi. The AR detection algorithm was run on one computer for all cameras, and they ran on average at 4 Hz.

Since the AR marker detector sometimes gives false positive detections, we decided to also use the list of detected people returned by the MHT algorithm, and only if the AR Marker detection was close to the location of a detected person, it was accepted. As side-effect, some false negatives occurred, but these had a lower effect on the search-and-track method, because in most cases they just delayed the detection of the person, whereas a false positive detection misguides the robot to go to an incorrect place.

3.6 Other Functions

For the search-and-track algorithms we used several other standard algorithms, which we will comment here shortly.

3.6.1 Distance

In searching, and navigation in general, it is important to take obstacles into account when planning a path and calculating its distance. Therefore, we have used the A* path planner [Thrun et al., 2005] to calculate the shortest path. The complexity to calculate a path is $O(b^d)$, where b is the branching factor and d the path depth. When the heuristic distance estimator is good enough, then $b = 1$. Note that the distance values were cached, since these values were used continuously, for example in calculating the strategy (e.g. a POMDP policy).

3.6.2 Visibility Check

To detect the visibility, we have used a simple 2D ray tracing algorithm on a discrete map, in which the cells are either free or an obstacle, and where no height differences are taken into account. In later simulations, subsection 5.4.1, a visibility probability is added that besides using a ray tracing algorithm, also calculates the probability of

seeing a person based on the distance. Note that the ray tracing and distance calculations are assumed to be of constant complexity since we cache the results, therefore the visibility check is also of constant complexity.

3.7 Safety Measures

A number of methods have been developed to allow robots to navigate safely around people in specific, typically non-generalizable, tasks. Furthermore, several groups have begun to address issues on how to plan complete paths around people, rather than relying on solely reactive behaviours. Two different notions of human safety are treated in [Zinn et al., 2004]: *physical safety* and *mental safety*. According to this work, the notion of safety includes both physical aspects and psychological effects of the robot's motions on humans. Physical safety is necessary for the Human Robot Interaction (HRI), and is usually assured by avoiding collisions with humans and by minimizing the intensity of the impact in case of a collision.

The robots, Tibi and Dabo, are equipped with red-colored mushroom-headed emergency stop push buttons, one onboard and one on a yellow safety remote control, which cut the communication to the platform stopping its movement. Furthermore, the yellow remote control must be close enough to the robot, otherwise the robot is stopped automatically. Finally, the robot continuously checked for obstacles to prevent colliding with obstacles or people.

In order to reduce any potential incidents or accidents with either a person or robot, several limitations were put during the experiments. First, the speed of the robot was limited below 1 m/s to be sure that no damage could be done. Second, for each robot an assistant was assigned who wore the safety remote, which allowed the robot to be stopped, and in the case of the remote being too far, the robot automatically stopped.

Chapter 4

Hide-and-Seek in Reduced Discrete Environments

In this chapter, we start with the introduction of the search-and-track problem as a hide-and-seek problem, in which the seeker is trying to catch the hider, while the hider tries to evade from the seeker and tries reach the base. This—at first sight—simple game, has several interesting aspects, such as requiring several cognitive skills, being a predefined game and being easy to evaluate.

We use Reinforcement Learning to solve this problem, and in specific, the Mixed Observable Markov Decision Process (MOMDP) model, which mixes partially and fully observable states, and therefore, has advantages in calculating the policy. Two variants are proposed, firstly an off-line method that uses the full resolution of the map; however, it resulted in long policy learning times for large maps. Therefore, we created an on-line hierarchical method that reduces the number of states at a higher level with lower resolution, and hence, resulted in lower computing times for the policy.

The seeker's movement was modelled as deterministic, completely depending on the action; whereas the hider's movement was modelled as either random or depending on historically measured movements. However, it was found that using the historical movements did not improve the method, since not enough data was available.

The methods were tested in simulation, using different environments and conditions, and in real-life experiments using Dabo, and a person that played the role of hider.

The work in this chapter has brought the following contributions:

- Real-world experiments of the hide-and-seek game using a MOMDP model [Goldhoorn et al., 2013a].
- Extended simulations of using an MOMDP and Hierarchical MOMDP were done [Goldhoorn et al., 2013b].
- A hierarchical model was proposed that reduces the number of states: the Hierarchical MOMDP [Goldhoorn et al., 2013a,b].
- Experiments were done through a game interface were humans played against a computer.
- A model of the hider's movement was made that uses historical data.

4.1 Introduction

Johansson and Balkenius [2005] suggested that the game of hide-and-seek is an ideal domain for studying cognitive functions in robots, moreover, it is a basic mechanism for HRI in mobile robotics, because hide-and-seek requires the robot to navigate, search, interact on and predict actions of the opponent. Hide-and-seek can be seen as a simplification of search-and-track, and we, thus, can use this research as a first step towards searching.

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 preprogrammed, 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. For this reason, we chose the MOMDP model [Araya-Lopez et al., 2010, Ong et al., 2010], a variant of POMDPs [Braziunas, 2003, Hauskrecht, 2000]. 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 an action following its own strategy to win the game.

In POMDP research, the hide-and-seek game often has been used as a benchmark to compare policy solvers. The computational complexity is a major problem when calculating policies for POMDPs, because it depends on the number of states, and grows exponentially with the search depth. In Section 2.3.1.1, we already have commented several works in which the game *tag* was used; for about 3000 states (an area of 55 cells), the policy calculation time of an MOMDP was half an hour [Ong et al., 2010].

In this chapter, we analyze and apply an off-line MOMDP and on-line Hierarchical MOMDP model. The off-line model worked very well for maps with a small number of grid cells, but it becomes intractable (PSPACE-hard, Papadimitriou and Tsisiklis [1987]) for a large number of grid cells. For this reason, we proposed an on-line MOMDP model that computes a locally near-optimal policy at every step, which 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, which 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, however, the policy is calculated at the top level MOMDP. The on-line method can also be applied to navigation problems, or problems where a high resolution map is used. Finally, an automated heuristic greedy seeker is tested for comparison. All seeker methods were tested in simulation and in real-life experiments, using the real robot Dabo (Section 3.3) against a human hider in a simple urban environment.

The chapter starts with an explanation of the used models, then the hide-and-seek game is described in Section 4.3 and an overview of the architecture is given in Section 4.4. The experimental settings are explained in Section 4.5 and the sections 4.6 and 4.7 explain the off-line and on-line methods respectively; and Section 4.8 explains the greedy methods. Section 4.9 shows the simulations done, and Section 4.10 shows the real-life experiments and we finish with a discussion in Section 4.11 and conclusions in Section 4.12.

4.2 Background

Reinforcement Learning (RL) can be used to learn actions, when only the goal is known, and when a greedy search through the possible actions is not enough. In our case, for example, we want to learn the best behaviour for the robot to find the person as quickly as possible. Markov Decision Process (MDP) models are often used to learn the best actions to do per state, for example a location of the person, which is guided by the rewards. To learn these actions, i.e. the policy, a solver is used to calculate it. First, we explain the MDP model, where the state is always known. However, in our problem the state is not always known, such as the location of the person, therefore a Partially Observable Markov Decision Process (POMDP) must be used. Finally, we explain a combination of the MDP and POMDP, the Mixed Observable Markov Decision Process (MOMDP) model, which allows for faster policy calculation.

4.2.1 Markov Decision Process

MDPs [Bellman, 1957, Sutton and Barto, 1998] are probabilistic models that are used to obtain the best actions to do in different scenarios. An MDP can be written as the tuple:

$$\langle S, A, T, R, \gamma \rangle \tag{4.1}$$

where S is the set of states that represent the world; A contains the actions that can be done by the agent; $T(s, a, s')$ defines the transition probabilities $P(s'|s, a)$ that indicates the probability of going from state s to s' with action a; R is the reward function that indicates a score for a state ($R(s)$), and possibly new state and action ($R(s, a, s')$); and, γ is the discount factor that reduces the importance of the expected rewards in the future.

As an example, we can take the problem of an agent having to navigate to a goal. The states represent the location of the agent as a discrete grid map, and the actions are, for example, the movement in four directions. The transition probability directly depends on the action, so we can say that the agent moves one grid cell in the direction of the action. Finally, the goal state should have a high reward, and has to be marked as *final state*.

Algorithm 4.1 The *Value Iteration* algorithm for an MDP.

```
1: ∀_{s∈S} : V(s) = 0
2: repeat
3:     Δ = 0
4:     for all s ∈ S do
5:         v = V(s)                                   ▷ old value
6:         V(s) = max_{a∈A} [R(s,a) + γ Σ_{s'∈S} T(s,a,s')V(s')]   ▷ updated value
7:         Δ = max(Δ, ‖v − V(s)‖)
8:     end for
9: until Δ < ε
```

The actions to take for each state are stored in the policy, and in order to generate the best policy (indicated by π^*), the expected reward—the Value Function—has to be calculated, which is done using the Bellman Equation [Bellman, 1957]:

$$V(s) = \max_{a \in A} \left(R(s,a) + \gamma \sum_{s' \in S} \left[T(s,a,s') V(s') \right] \right) \tag{4.2}$$

where $\gamma < 1.0$ is the discount factor that reduces the influence of future expected rewards, normally $\gamma \geq 0.9$. The Value Function is learned by either *Value Iteration* or *Policy Iteration* [Sutton and Barto, 1998], which will be explained next.

Value Iteration: Algorithm 4.1 shows the *Value Iteration* algorithm, which calculates the Value Function by iteratively applying Eq. 4.2. To obtain the optimal policy V^*, an infinite number of iterations have to be done, instead however, the Value Function is approximated by doing iterations until V does not change anymore: $\max_{s \in S} \|V_{t+1}(s) - V_t(s)\| < \epsilon$, where ϵ is a small number.

The policy can be defined, once the Value Function has been calculated:

$$\pi(s) = \arg\max_{a \in A} \left(R(s,a) + \gamma \sum_{s' \in S} \left[T(s,a,s') V(s') \right] \right) \tag{4.3}$$

Policy Iteration: For *Policy Iteration*, see Algorithm 4.2, the Value Function $V(s)$ and policy (π) are initialised randomly, and then two steps are iterated over until the policy does not change any more: *Policy Evaluation* and *Policy Improvement*.

Algorithm 4.2 The *Policy Iteration* algorithm for an MDP.

1: $\forall_{s \in S} : V(s) = \text{RANDOM}(\mathbb{R})$, $\pi(s) = \text{RANDOM}(A)$
2: **repeat**
3: ▷ **Policy Evaluation**
4: **repeat**
5: $\Delta = 0$
6: **for all** $s \in S$ **do**
7: $v = V(s)$ ▷ old value
8: $V(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V(s')$ ▷ new value
9: $\Delta = \max(\Delta, \|v - V(s)\|)$
10: **end for**
11: **until** $\Delta < \epsilon$
12: ▷ **Policy Improvement**
13: policy-stable = **true**
14: **for all** $s \in S$ **do**
15: $a = \pi(s)$ ▷ old policy action
16: $\pi(s) = \arg\max_{a' \in A} \left(\sum_{s \in S} T(s, a', s') \left[R(s) + \gamma V(s') \right] \right)$ ▷ new action
17: **if** $a \neq \pi(s)$ **then**
18: policy-stable = **false**
19: **end if**
20: **end for**
21: **until** policy-stable

First, the policy is evaluated from line 4, where in line 8 the Value Function is updated using the action of the policy π. Next, from line 13, the policy is checked on changes, and it is accepted if it has not been changed.

4.2.1.1 Temporal-Difference Learning

Whereas the previously discussed RL method requires a model (i.e. transition probabilities), TD Learning [Sutton and Barto, 1998] methods do not. They make use of Monte-Carlo methods and Dynamic Programming to find a policy; examples are Sarsa, Q-learning, and Actor-Critic.

In *Q-learning* [Sutton and Barto, 1998] an action-value function $Q(s, a)$ is learned based on the current and previous rewards:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a \in A} Q(s_{t+1}, a) - Q(s_t, a_t) \right] \tag{4.4}$$

where t is the time of the learning step, s_{t+1} the new state reached with the executed action a_t and received reward r_t, α is the learning rate, and γ the discount factor.

4.2.2 Partially Observable Markov Decision Process

In MDPs the state is always known, but in our problem the person's position is not always known, which is part of the state; to handle this, a Partially Observable Markov Decision Process (POMDP) [Thrun et al., 2005, White, 1991] can be used. The partially observable model POMDP has two main differences with the fully observable model MDP: first, the current state is not known, but a belief (probability map) represents the probability of being in each state; second, the belief is updated through observations.

A POMDP is a tuple:

$$\langle S, A, O, T, Z, R, b_0, \gamma \rangle \tag{4.5}$$

that contains, like an MDP (discussed before in subsection 4.2.1) a set of states (S), actions (A), rewards (R), and a state transition function T, which defines the probability of going to s' from s with action a: $T(s, a, s') = P(s'|s, a)$. Instead of knowing the current state, observations (O) are used in POMDPs, and an observation probability function Z, which defines the probability of observing o from new state s': $Z(o, s', a) = P(o|s', a)$. Also an initial belief b_0 has to be given since the state is not known initially.

The belief ($\mathcal{B}$) is the probability of being in each possible state; Figure 4.1 shows the belief in two situations: the person is hidden (*top*) and visible (*bottom*). To update the belief the *observation* o and action a are used:

$$b_a^o(s') = \frac{Z(o, s', a) \sum_{s \in S} T(s, a, s') b(s)}{\Omega(o|b, a)} \tag{4.6}$$

where $b_a^o(s')$ is the probability of being in state s' after having done observation o and action a; b is the previous belief. A normalisation is done by dividing by $\Omega(o|b, a)$, the probability of observation based on the belief and action:

$$\Omega(o|b, a) = \sum_{s' \in S} \left(Z(o, s', a) \sum_{s \in S} \left[T(s, a, s') b(s) \right] \right) \tag{4.7}$$

The initial belief b_0, has to be given in advance; for example, a uniformly distributed probability over all the locations where the person might be hidden (such as shown in Figure 4.1(b)). Thereafter, the belief is updated using the observation and the probability functions.

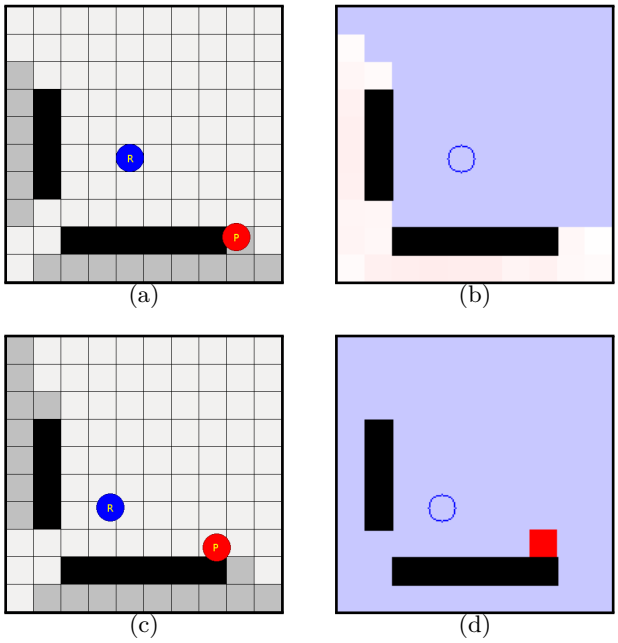


Figure 4.1: The simulation of two sequential positions is shown. Left the maps are shown with the blue circle being the robot (R), and the red the person (P). Black squares are obstacles, and dark grey squares indicate locations which are not visible to the robot. The right images show a distribution of the belief (without noise), where red indicates a high, white a low and light blue zero probability.

The reward is given for a state and action $R(s, a)$, and is used to calculate the best action to do in each (belief) state using the Value Function per action and belief:

$$Q(b, a) = \rho(b, a) + \gamma \sum_{o \in O} \Omega(o|b, a) V(b_a^o) \tag{4.8}$$

where $\rho(b, a) = \sum_{s \in S} [b(s)R(s, a)]$ is the reward for the belief $b \in \mathcal{B}$ and action a; γ is the discount factor; and b_a^o is the next belief state, defined in Eq. 4.6. Now, we can define the Value Function (like Eq. 4.2):

$$V(b) = \max_{a \in A} Q(b, a) \tag{4.9}$$

The policy π, the best action to take in each state, is:

$$\pi(b) = \arg\max_{a \in A} Q(b, a) \tag{4.10}$$

4.2.2.1 Policies

The Value Function for a POMDP can be defined as a linear convex function [Pineau et al., 2003], which is represented by a list of α-vectors: $V_n = \{\alpha_0, \alpha_1, \cdots, \alpha_n\}$, where an α-vector is an $|S|$-dimensional hyper-plane containing the values for a certain action a.

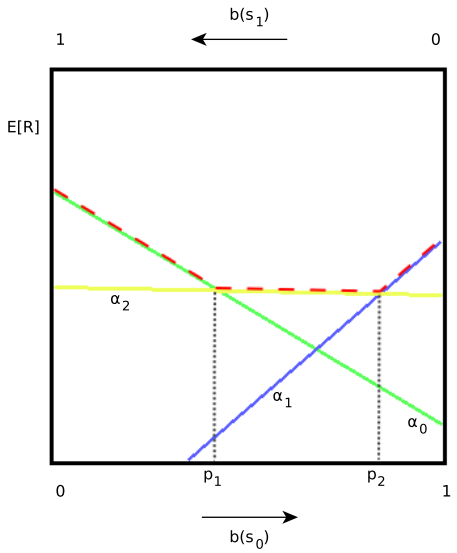


Figure 4.2: The value function of the POMDP is represented by a list of α-vectors, in this example, there are three: $V = \{\alpha_0, \alpha_1, \alpha_2\}$. Horizontally, the belief $b(s_0)$ is represented below, and $b(s_1)$ up. The vertical axis indicates the expected reward. The red dashed line represents the maximum value for each belief.

Figure 4.2 shows an example of a value function V with three α-vectors. In this example there are only two states: $S = \{s_0, s_1\}$; their belief is shown horizontally: $b(s_0)$ below, and $b(s_1)$ up; note that $b(s_1) = 1.0 - b(s_0)$, since there are only two states. Vertically, the expected reward is shown, and each of the α-vectors represents an action: $A = \{a_0, a_1, a_2\}$. The dashed red line indicates the best value for each belief, which would result in the following policy:

$$\pi = \begin{cases} a_0, & \text{if } 0 \leq b(s_0) < p_1 \\ a_2, & \text{if } p_1 \leq b(s_0) < p_2 \\ a_1, & \text{if } p_2 \leq b(s_0) \leq 1.0 \end{cases} \tag{4.11}$$

Another way of representing a policy is with a tree; Figure 4.3 shows a partial tree of depth one (a complete policy tree would be much larger). The root of the policy tree represents the current situation, which is a state for, the MDP, and a belief for the POMDP. The trees in Figure 4.3 show that the POMDP's policy tree is wider than the MDP's tree, because it also contains the observations.

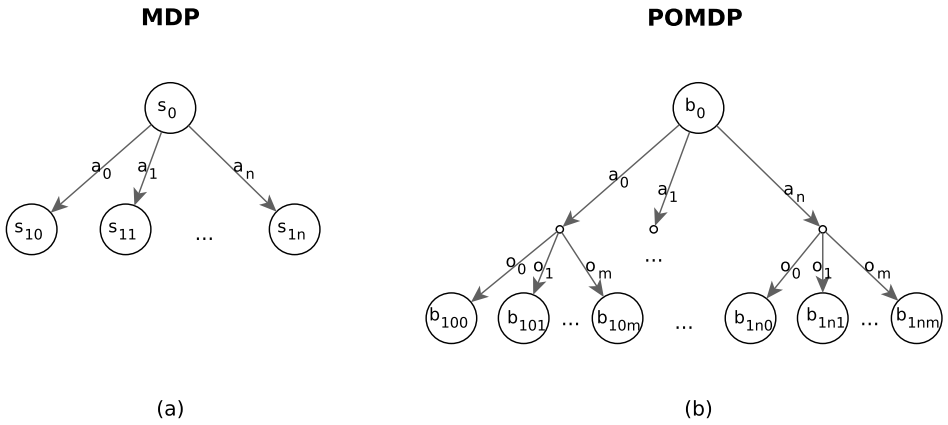


Figure 4.3: The policy tree of an MDP and a POMDP model; note that the figures only show a depth of one, and that in (a) the MDP, the state is known, whereas for (b) the POMDP, the beliefs contains a probability of each state.

Executing the policy is done by finding the maximum approximated expected reward:

$$V(b) = \max_{\alpha \in \Gamma}(\alpha \cdot b) \tag{4.12}$$

where Γ is the list of α-vectors and b is the belief point.

4.2.2.2 Policy Calculation

Finding POMDP policies has the problem of being complex—intractable (PSPACE-hard) [Papadimitriou and Tsisiklis, 1987] to find the exact policy—and furthermore, it is known to suffer from the *curse of dimensionality* and the *curse of history* [Pineau et al., 2003, Silver and Veness, 2010], because of the infinite continuous belief space.

Exact Value Iteration The calculation of the Value Function V is also noted as $V = HV'$ [Pineau et al., 2003], where V' is the previous version of the Value Function,

and H is the backup operator. Algorithm 4.3 shows the *Exact Value Iteration* (see for example [Pineau et al., 2003]) for which the Value Function (Eq. 4.8) is calculated in several steps for all the actions and observations. Firstly, in line 1 the direct reward is calculated for all actions a, and it is stored in the set $\Gamma^{a,*}$; secondly, in line 2 the discounted future reward is calculated for each action a and observation o, and all existing α-vectors in V'. Thirdly, the values are summed based on the actions and observations (line 3); finally, all sets of α-vectors are joined in line 4.

It can be seen, that the set Γ grows exponential if no pruning is done, the new set V has, at worst case, $|A||V'|^{|O|}$ α-vectors; and the time complexity is $|S|^2|A||V'|^{|O|}$ [Pineau et al., 2003].

Algorithm 4.3 The steps of the *Exact Value Iteration* for a POMDP [Pineau et al., 2003].

1: $\Gamma^{a,*} \leftarrow \alpha^{a,*}(s) = R(s,a)$
2: $\Gamma^{a,o} \leftarrow \alpha^{a,o}(s) = \gamma \sum_{s' \in S} [T(s,a,s')Z(o,s',a)\alpha'_i(s')], \forall \alpha'_i \in V'$
3: $\Gamma^a = \Gamma^{a,*} \oplus \bigoplus_{o \in O} \Gamma^{a,o}$
4: $V = \bigcup_{a \in A} \Gamma^a$

Curse of Dimensionality: The policy of a POMDP does not define exactly which action to do in which state, however, which action to do in a certain belief state. Since the belief is probabilistic, this space is infinite with $|S| - 1$ dimensions, S being the set of states. For each added state, a new dimension is added to the belief, this is called the *Curse of Dimensionality* [Pineau et al., 2003]: it scales exponentially with the number of states.

Curse of History: When trying to find an optimal policy, learning is started with an initial belief, then all of the action-observation combinations have to be traced, which grows exponentially with the planning horizon (search depth), see Figure 4.3. This growth affects the POMDP value iteration far more than the *Curse of Dimensionality* [Pineau et al., 2003].

Algorithm 4.4 The backup function of the PBVI algorithm.

1: $\Gamma^{a,*} \leftarrow \alpha^{a,*}(s) = R(s,a)$
2: $\Gamma^{a,o} \leftarrow \alpha^{a,o}(s) = \gamma \sum_{s' \in S} [T(s,a,s')Z(o,s',a)\alpha'_i(s')], \forall \alpha'_i \in V'$
3: $\Gamma^a_b = \Gamma^{a,*} + \sum_{o \in O} \arg\max_{\alpha \in \Gamma^{a,o}}(\alpha \dot{b})$
4: $V = \arg\max_{\Gamma^a_b, \forall a \in A}(\Gamma^a_b \dot{b}), \forall b \in \mathcal{B}$

4.2.2.3 POMDP Solvers

Since finding the exact policy is hard to calculate, approximation methods are used. They sample the belief space in a smart way, to find a policy [Kurniawati et al., 2008, Pineau et al., 2003].

PBVI: Instead of exploring the whole belief space, it is more practical to only explore representative points of the whole space, this is done by the Point-Based Value Iteration (PBVI) solver [Pineau et al., 2003, Spaan and Vlassis, 2004], which explores only reachable belief points. Figure 4.3 shows that new belief points are reachable for the POMDP by doing an action and an observation, which both are limited sets.

PBVI extends the belief space $\mathcal{B} = \{b_0, b_1, .., b_n\}$, by searching reachable belief points that improve the coverage of the total belief. For each of those belief points, an α-vector is calculated, which has the same length as the number of states, and represents the expected reward of each state. The policy is stored as the collection of α-vectors, and for each α-vector the best action is stored.

Algorithm 4.4 shows the PBVI backup function. PBVI expands the set of belief points by greedily expanding the set that improves the worst-case density as fast as possible. It is an anytime algorithm, i.e. it can return a policy even though it has not yet converged. Here the final solution only contains $|\mathcal{B}|$ points, and has time complexity $|S||A||V'||O||B|$ [Pineau et al., 2003].

SARSOP: An improvement of PBVI is the anytime algorithm Successive Approximations of the Reachable Space under Optimal Policies (SARSOP) [Kurniawati et al., 2008], which limits the searched belief space to only optimally reachable belief points. These are the points that are reached by doing an optimal sequences of actions, starting from the initial belief b_0. The algorithm keeps track of a lower and upper bound, the

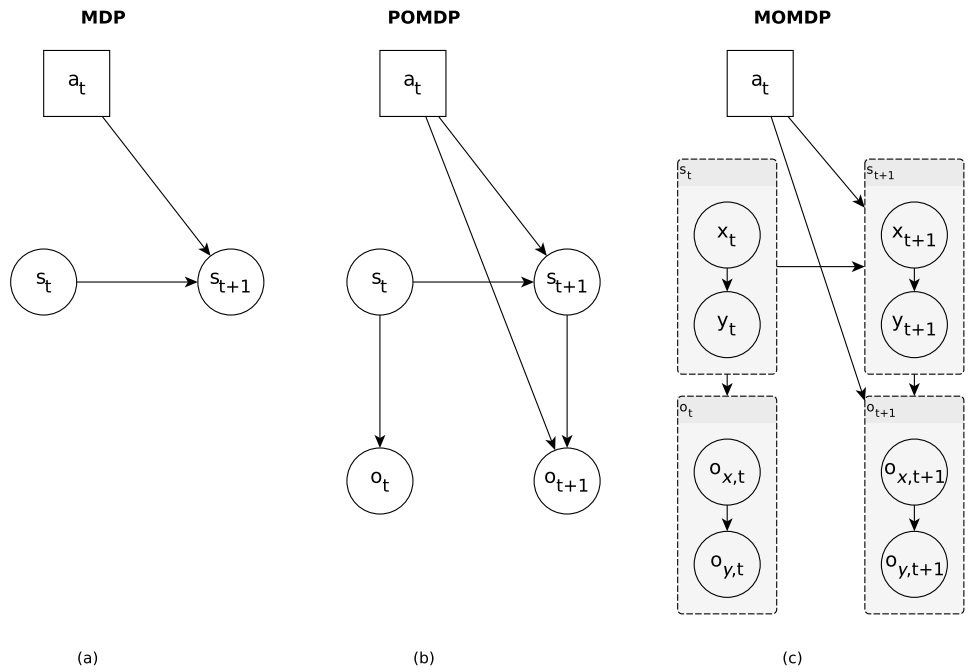


Figure 4.4: The figure shows the dependencies of the states and observations for the MDP, POMDP, and MOMDP models.

first is represented by the set of α-vectors Γ, and the second by a sawtooth approximation [Kurniawati et al., 2008].

4.2.3 Mixed Observable Markov Decision Process

In some problems, not the whole state space is partially observable, such as the problem of hide-and-seek, in which the hider is not always visible, but the seeker is. For these cases, the fully and partially visible state variables can be separated. A model that does this is the Mixed Observable Markov Decision Process (MOMDP) [Araya-Lopez et al., 2010, Ong et al., 2010].

An MOMDP is a tuple:

$$\langle \mathcal{X}, \mathcal{Y}, \mathcal{A}, \mathcal{O}_\mathcal{X}, \mathcal{O}_\mathcal{Y}, T_\mathcal{X}, T_\mathcal{Y}, Z_\mathcal{X}, Z_\mathcal{Y}, R, (x_0, b_{\mathcal{Y}0}), \gamma \rangle \tag{4.13}$$

where the states, observations, transition function, and observation function are split for a fully visible part $\mathcal{X}$, and a partially visible part $\mathcal{Y}$. The influence of the states and

observations is shown in Figure 4.4, compared to the MDP and POMDP.

Here we give a list of all the items of the MOMDP, Eq. 4.13:

- $\mathcal{X}$: the fully-observable state variables;
- $\mathcal{Y}$: the partially-observable state variables;
- A: the set of actions;
- $O_{\mathcal{X}}$: the list of observations for the fully visible state variables, in most cases: $\mathcal{O}_{\mathcal{X}} = \mathcal{X}$, note that this is added for completeness and in [Ong et al., 2010] x is used directly instead of o_x;
- $O_{\mathcal{Y}}$: the set of observations for the partially visible state variables;
- $T_{\mathcal{X}}$: $T_{\mathcal{X}}(x, y, a, x') = p(x'|x, y, a)$: the transition probabilities of the visible state variables, given an action;
- $T_{\mathcal{Y}}$: the transition probabilities of the partially observable state variables, $T_{\mathcal{Y}}(x, y, a, x', y') = p(y'|x, y, a, x')$;
- $Z_{\mathcal{X}}$: the observation probabilities for the visible state part, $Z_{\mathcal{X}}(x', y', a, o_x) = p(o_x|x', y', a)$;
- $Z_{\mathcal{Y}}$: the observation probabilities for the partially visible state variables, $Z_{\mathcal{Y}}(x', y', a, o_x, o_y) = p(o_y|x', y', a, o_x)$;
- R: the reward function $R(x, y, a)$;
- $(x_0, b_{\mathcal{Y}0})$: the initial belief;
- γ: the discount factor.

The belief space $\mathcal{B}$ is not one probability map for all states, but for all partially observable states, stored per fully observable state variable x: $\mathcal{B}_{\mathcal{Y}}(x)$. And although the total belief space $\mathcal{B}$ has $|\mathcal{X}| \times |\mathcal{Y}|$ dimensions, it is faster to calculate the Value Function compared to Eq. 4.12:

$$V(x, b_{\mathcal{Y}}) = \max_{\alpha \in \Gamma_{\mathcal{Y}(x)}} (\alpha \cdot b_{\mathcal{Y}}) \tag{4.14}$$

The advantage is that all the operations are done on the smaller subspace $\Gamma_{\mathcal{Y}}(x)$ instead over the whole set of α-vectors, Γ.

The new belief depends on the action a and observation $o = \langle o_x, o_y \rangle$, and Figure 4.5 shows all the reachable belief states, which depend on the number of observations ($|O_{\mathcal{X}}| \cdot |O_{\mathcal{Y}}|$) and actions ($|A|$). The belief update changes with respect to the POMDP (see Eq. 4.6):

$$b^o_{a,\mathcal{Y}}(y') = \frac{1}{\Omega(o|b,a)} Z_{\mathcal{X}}(x', y', a, o_x) Z_{\mathcal{Y}}(x', y', a, o_x, o_y) \times \sum_{y \in \mathcal{Y}} T_{\mathcal{X}}(x, y, a, x') T_{\mathcal{Y}}(x, y, a, x', y') b_{\mathcal{Y}}(y) \quad (4.15)$$

The main difference with respect to Eq. 4.6 is that the transition and observation probabilities are now separated; here also the belief is normalized by $\Omega(o|b,a)$ (similar to Eq. 4.7, but for an MOMDP).

Policy Calculation: The policy tree of the MOMDP, see Figure 4.5, adds one layer more ($O_{\mathcal{X}}$) to the tree with respect to the POMDP policy tree (Figure 4.3). But, only a belief is maintained over the partially observable states $\mathcal{Y}$, because the fully observable state $x \in \mathcal{X}$ is known.

The solver for the MOMDP model, to calculate a policy, is based on SARSOP (see subsubsection 4.2.2.3). It uses a lower bound and upper bound to sample the policy space, which is reachable and optimal. For the upper bound a sawtooth approximation [Hauskrecht, 2000] is used, and for the lower bound the Value Function is calculated as shown in Algorithm 4.5.

Algorithm 4.5 *Value Iteration* for MOMDPs, based on SARSOP. With input $T_R, \Gamma, (x, b_{\mathcal{Y}})$, and $o = \langle o_x, o_y \rangle$.

1: $\Gamma^{a,o} \leftarrow \alpha^{a,o}(s) = \arg\max_{\alpha \in \Gamma_{\mathcal{Y}}(o_x)} \left(\alpha b^o_{a,\mathcal{Y}} \right), \forall a \in A, o \in O_{\mathcal{X}} \times O_{\mathcal{Y}}$
2: $\Gamma^{a,y} \leftarrow \alpha^{a,o}(s) = R(x,y,a) + \gamma \sum_{o,x',y'} \left[T_{\mathcal{X}}(x,y,a,x') T_{\mathcal{Y}}(x,y,a,x',y') Z_{\mathcal{Y}}(x',y',a,o) \alpha^{a,o,x'}(y') \right], \forall y \in \mathcal{Y}, a \in A$
3: $a' = \arg\max_{a \in A} (\alpha^a b_{\mathcal{Y}})$
4: Insert $\alpha^{a'}$ into $\Gamma_{\mathcal{Y}}(x)$

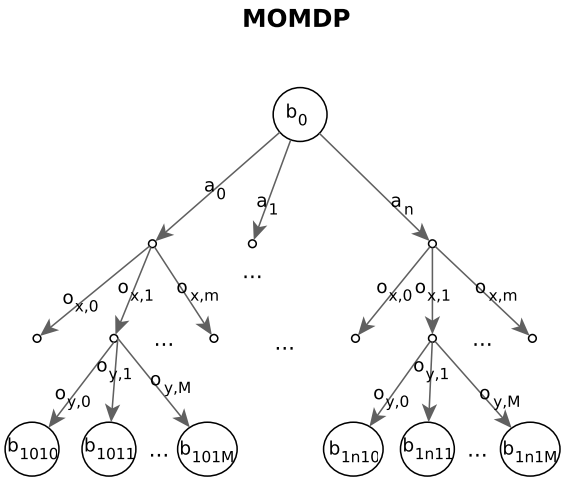


Figure 4.5: The policy tree of an MOMDP model, see Figure 4.3 for the trees of an MDP and an POMDP. Note that each node contains the belief $b_i = \langle o_x, b_y \rangle$; and, note that the policy is only of depth one.

4.3 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 (preferably not visible from the base). The agents are assumed 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 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 neighbour cell in an 8-connectivity neighbourhood (i.e. a maximum of nine actions for each player: eight moving and one standing still). The seeker wins, if (s)he approaches the hider sufficiently (we use a distance of one cell) and *catches* it. The hider wins if (s)he 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.

The MOMDP, presented in Sections 4.6 and 4.7, models the game from the point of view of the seeker, this is, we want to learn a policy for the seeker without knowing the hider's 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. The hider's position is identified if the hider is visible from the seeker's position, and otherwise it is *unknown* for the seeker.

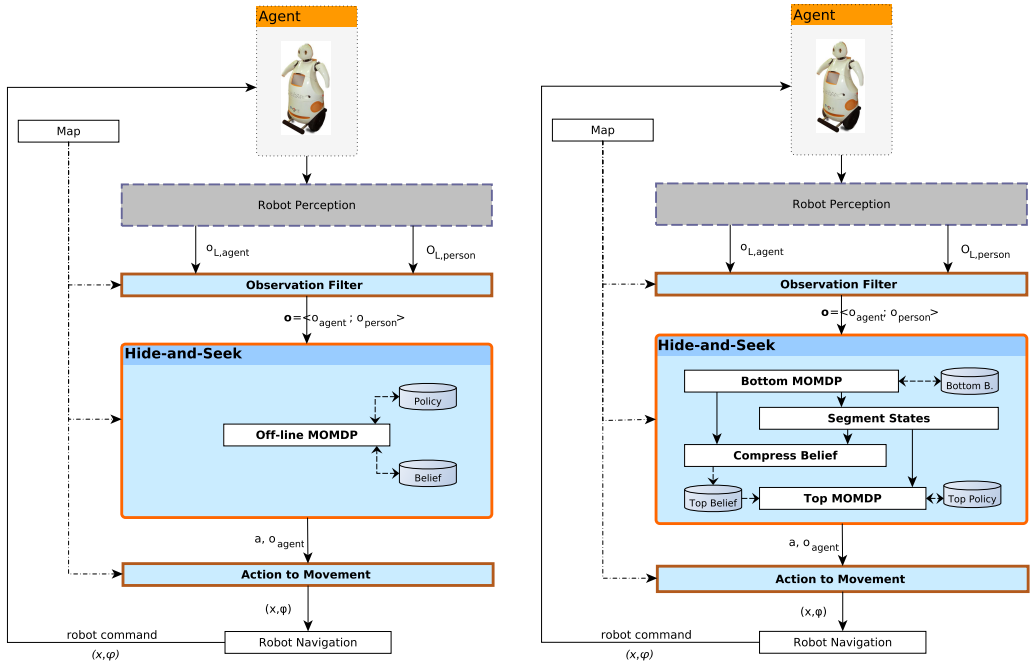
In the simulations, two virtual agents were involved: an automated seeker—applying the MOMDP policy or using a heuristic; and a random or a *smart* (heuristically driven) hider. In our real-life experiments, a physical robot (Dabo) had the role of the seeker, and played against a human opponent in the role of the hider.

4.4 Overview of the Approach

An overview of the presented method is shown in Figure 4.6 with the different layers of the system. Section 3.2 already explains the *Robot Perception* and *Robot Navigation* in detail. Here, we focus on the hide-and-seek model presented in this chapter, with two versions: the off-line MOMDP (Figure 4.6(a)) and the Hierarchical MOMDP (Figure 4.6(b)).

The hide-and-seek models have as input the *Observation Filter*'s output $\mathbf{o} = \langle o_{\text{agent}}, o_{\text{person}} \rangle$. And the output of the hide-and-seek method is an *action*, which is a step in one of eight directions or staying at the same location. The *Action to Movement* module uses the action a and the agent's position (o_{agent}) to calculate a new pose, which is sent to the *Robot Navigation* module.

The off-line MOMDP uses an MOMDP policy that is learned off-line, before doing the experiments. During the games, the belief is maintained and used to find the best action in the policy. The Hierarchical MOMDP has two layers: a *Bottom MOMDP*, which has the same resolution as the off-line MOMDP; and a *Top MOMDP*, which has less states. The *Top MOMDP's* states are created by segmenting states of the *Bottom MOMDP*, and the belief is compressed to the *Top Belief*. After having generated the *Top MOMDP*, a policy is learned on-line, which is then used to find the best action for the current *Top Belief*.



(a) The off-line MOMDP.

(b) The on-line Hierarchical MOMDP.

Figure 4.6: The schema of the different processes in the hide-and-seek methods presented in this chapter. The architecture of the hide-and-seek methods: (a) the off-line MOMDP, and (b) the Hierarchical MOMDP. The blocks are algorithms or groups of algorithms, the orange bold lined blocks were created by us.

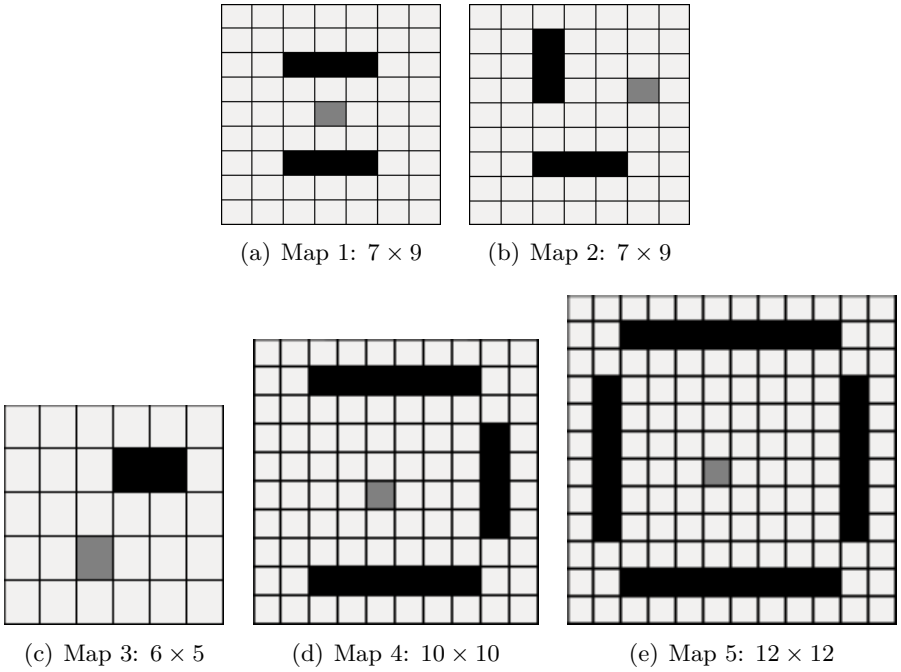
4.5 Experimental Settings

In the experiments, we used the robot Dabo to play hide-and-seek, and a marker to detect the hider.

4.5.1 Maps

In order to do the simulations and real-life experiments, we created discretized 2D grid maps of the FME environment, where the players could move in one of the eight directions. Two setups of the environment were created with different obstacle layouts: map 1 and map 2 (Figures 4.7(a,b)). The size of the grid cells is 1 m, which implies a grid size of 7×9 cells. The base and two obstacles, with a length of three cells (3 m), have been placed on different positions.

We used the maps which can be seen in Figure 4.7; for the first simulations, we used the same maps but with a higher resolution: 9×12; for the second part of the simulations, we used different sized maps (maps 3-5 in Figure 4.7c-e).



(a) Map 1: 7×9 (b) Map 2: 7×9

(c) Map 3: 6×5 (d) Map 4: 10×10 (e) Map 5: 12×12

Figure 4.7: The maps used in the simulated and real-life experiments. Black cells are obstacles, the dark gray cell is the base.

4.5.2 Hardware and Software

The real-life experiments were done with the robot Dabo (see Section 3.3). And the simulations were done on a stand alone PC with 8 GB of RAM and an Intel Core^{TM}i5 CPU 760 @ 2.80 GHz with 4 cores and Ubuntu 12.04 as OS.

To generate the policies, we used the Approximate POMDP Planning Toolkit as solver for the POMDPs and MOMDPs, described in [Kurniawati et al., 2008, Ong et al., 2010]. We used `APPL Offline 0.95`, which is available on http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/index.php?n=Main.Download.

4.6 Off-line MOMDP Model for Hide-and-seek

The hide-and-seek game can be cast as an off-line MOMDP model (subsection 4.2.3), 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. In our game, we assume that the robot's position is fully observable, and the hider's position is not always visible (partially observable). This results in a compact lower-dimensional representation of its belief space.

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 m $\times$ 10 m implies 10 000 MOMPD states). The hide-and-seek game is modelled as an MOMDP, thus, we have to define the items of the tuple Eq. 4.13:

- $\mathcal{X}$: the fully-observable state variable contains the seeker's position $x = (x_{\text{seeker}}, y_{\text{seeker}})$.
- $\mathcal{Y}$: the partially-observable state variable contains the hider's position $y = (x_{\text{hider}}, y_{\text{hider}})$.
- A: the nine 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. Note that the hider also can move at the same time, but we assume it to be independent on the seeker's movement.
- $O_{\mathcal{X}}$: $\mathcal{O}_{\mathcal{X}} = \mathcal{X}$, since $o_x = x$ for all states of the seeker.
- $O_{\mathcal{Y}}$: $\mathcal{O}_{\mathcal{Y}} = \mathcal{Y} \cup \{unknown\}$, 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.
- $T_{\mathcal{X}}$: 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 either 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 will result in not modifying the seeker's state variable. Reaching the final state also results in staying in the same state.

- T_y: the transition probabilities of the hider's state, given a seeker's action and locations of the seeker and hider. 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 the next subsection. Also here the probability will be 1 if a final state has been reached.
- $Z_{\mathcal{X}}$: the observation probabilities are 1 if $o_x = x'$ and 0 otherwise, since $\mathcal{X} = O_{\mathcal{X}}$.
- Z_y: the observation probabilities depend on the map, and the locations of the seeker and hider. The probability is 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 is 0.
- R: two reward functions have been tested (see subsection 4.6.2).
- γ: the discount factor is set to 0.95.

Final states are defined as either the seeker catching the hider ($x_{\text{final}} = y_{\text{final}}$), or the hider reaching the base without being caught. When a final state has been reached, the transition functions are defined to stay in the same state:

$$\begin{aligned} p(x_{\text{final}}|x_{\text{final}}, y_{\text{final}}, a) &= 1.0 \\ p(y_{\text{final}}|x_{\text{final}}, y_{\text{final}}, a, x_{\text{final}}) &= 1.0 \end{aligned} \tag{4.16}$$

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

4.6.1 Hider Transition Function

As a first option to model the hider's movement, we used a uniform probability of the person choosing any of the nine actions the seeker also can choose: staying at the same

position or moving in one of the eight directions. Another method is to base these probabilities on real-world data, where the number of actions per state is stored and normalized to generate the transition function T_y. However, it requires a large number of games, since each state is the combination of the hider's and seeker's position.

4.6.2 Reward Function

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 [Araya-Lopez et al., 2010, Ong et al., 2010]. These functions are:

- *Simple reward*: non-zero values only for final states:

$$R_{\text{simple}}(s,h,b) = \begin{cases} 1, & \text{if } s = h \\ -1, & \text{if } h = b \\ 0, & \text{otherwise} \end{cases} \tag{4.17}$$

  Where s, h, and b are respectively the positions of the seeker, the hider and the base.

- *Triangle reward*: this reward makes use of the three important distances—all measured as shortest path distance—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 (d_{sb}):

$$R_{\text{tri}}(s,h,b) = \begin{cases} D_{\max} - d_{sh}, & \text{if } d_{hb} > d_{sb} \\ -d_{sh}, & \text{otherwise} \end{cases} \tag{4.18}$$

  where $D_{\max}$ is a maximum distance constant, depending on the map size, see Figure 4.8.

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. For the triangle reward, the distances have to be calculated, which increases the complexity. Therefore, either the distances should be precalculated for each state—implying a higher memory cost—or the computation time is increased considerably if calculated at each step. We chose for the first option, since the used maps were small.

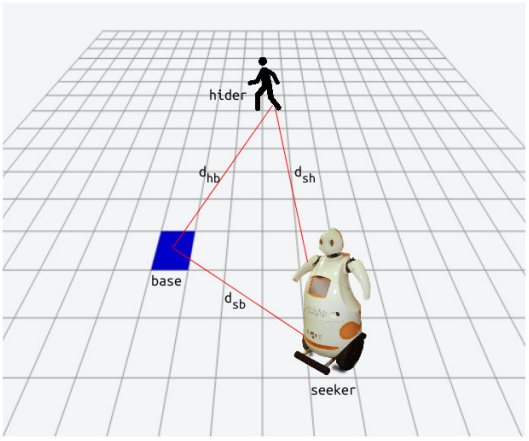


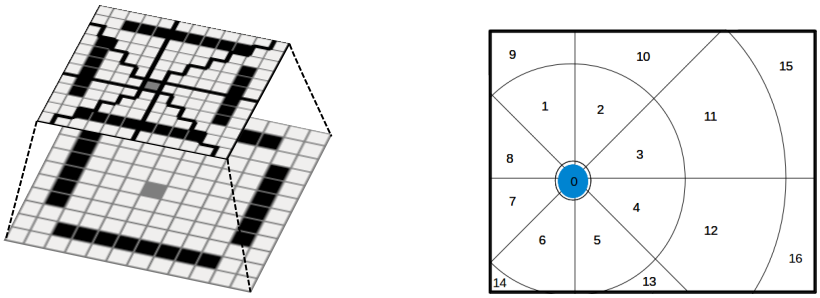
Figure 4.8: The *Triangle reward* (Eq. 4.18) is calculated based on the distances (d_{sh}) between the seeker, hider, and base.

4.7 On-line MOMDP Model for Hide-and-seek

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[1]). Furthermore, the time and memory complexity grow with the number of states due to the curse of history and dimensionality (see subsubsection 4.2.2.2); therefore, we present a method that reduces the number of states.

We present a hierarchical model, based on [Foka and Trahanias, 2007], as shown in Figure 4.9(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 are calculated, based on the probabilities and belief in the bottom MOMDP. A policy is calculated for the top MOMDP on-line, and directly thereafter, the policy is 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.

[1] Using the CPU described in subsection 4.5.2



(a) Layered segmentation (b) Robot centred segmentation

Figure 4.9: 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 bold lines in the top level indicate the top level states. The *robot centred* segmentation (b) centres on the robot's location (0 in the figure), and from there on creates segments based on the direction and distance.

4.7.1 Bottom-level MOMDP

The bottom level is a full MOMDP (Eq. 4.13), defined in the same way as described in Section 4.6, 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 the bottom-level belief b_y is updated using Eq. 4.15, 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.

4.7.2 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 $\psi(y_T) = \{y_0, y_1, \cdots, y_k\}$ gives the set of bottom-level adjacent states $y_0, y_1, \cdots, y_k$, which are covered by 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 reduce the time 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 [Stockman and Shapiro, 2001], 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.

4.7.2.1 Robot Centred 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. The big advantages of this method are that it is easy to calculate, and the number of segments are low and independent of the map size.

Figure 4.9(b) shows the robot centred 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 centre. This segmentation focuses on the direction and not the exact location; which is sufficient because a new robot centred 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 represent only one cell in the bottom level.

4.7.2.2 Top MOMDP

The top-level MOMDP can be defined as follows:

$$\langle \mathcal{X}_T, \mathcal{Y}_T, \mathcal{A}_T, \mathcal{O}_{\mathcal{X},T}, \mathcal{O}_{\mathcal{Y},T}, T_{\mathcal{X},T}, T_{\mathcal{Y},T}, Z_{\mathcal{X},T}, Z_{\mathcal{Y},T}, R_T, \gamma \rangle \tag{4.19}$$

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;
- $Z_{\mathcal{X},T} = Z_{\mathcal{X}}$.

Therefore, the Top MOMDP reduces to the following tuple:

$$\langle \mathcal{X}, \mathcal{Y}_T, \mathcal{A}, \mathcal{O}_{\mathcal{X}}, \mathcal{O}_{\mathcal{Y}}, T_{\mathcal{X},T}, T_{\mathcal{Y},T}, Z_{\mathcal{X}}, Z_{\mathcal{Y},T}, R_T, \gamma \rangle \tag{4.20}$$

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

$$T_{\mathcal{X},T}(x, y_T, a, x') = p(x'|x, y_T, a) = \frac{1}{|\psi(y_T)|} \sum_{y \in \psi(y_T)} T_{\mathcal{X}}(x, y, a, x') \tag{4.21}$$

$$\begin{aligned} T_{\mathcal{Y},T}(x, y_T, a, x', y'_T) &= p(y'_T|x, y_T, a, x') \\ &= \frac{1}{|\psi(y_T)|} \sum_{y' \in \psi(y'_T)} \sum_{y \in \psi(y_T)} T_{\mathcal{Y}}(x, y, a, x', y') \end{aligned} \tag{4.22}$$

Note that in our implementation we have not made the seeker's position (x) dependent on the hider's position (y), therefore, Eq. 4.21 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:

$$\begin{aligned} p(x_f|x_f, y_{T,f}, a) &= 1.0 \\ p(y_{T,f}|x_f, y_{T,f}, a, x_f) &= 1.0 \end{aligned} \tag{4.23}$$

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:

$$Z_{\mathcal{Y},T}(x', y'_T, a, o_x) = p(o_y|x', y'_T, a) = \frac{1}{|\psi(y'_T)|} \sum_{y' \in \psi(y'_T)} Z_{\mathcal{Y}}(x', y', a, o_x, o_y) \tag{4.24}$$

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

$$b_{\mathcal{Y},0,T}(y_T) = \sum_{y \in \psi(y_T)} b_{\mathcal{Y}}(y) \tag{4.25}$$

Segment $\mathcal{X}$ and O When both state variables $\mathcal{X}$ and $\mathcal{Y}$ and the observations $O_{\mathcal{X}}$ are segmented with functions ψ_X, ψ_Y and ψ_O respectively, then the transition and observation probabilities change slightly:

$$\begin{aligned} T_{\mathcal{X},T}(x_T, y_T, a, x'_T) &= p(x'_T|x_T, y_T, a) \\ &= \frac{1}{|\psi_X(x_T)||\psi_Y(y_T)|} \sum_{x' \in \psi_X(x'_T)} \sum_{x \in \psi_X(x_T)} \sum_{y \in \psi_Y(y_T)} p(x'|x, y, a) \end{aligned} \tag{4.26}$$

$$\begin{aligned} T_{\mathcal{Y},T}(x_T, y_T, a, x'_T, y'_T) &= p(y'_T|x_T, y_T, a, x'_T) \\ &= \frac{\sum_{y' \in \psi_Y(y'_T)} \sum_{x' \in \psi_X(x'_T)} \sum_{x \in \psi_X(x_T)} \sum_{y \in \psi_Y(y_T)} p(y'|x, y, a, x')}{|\psi_X(x_T)||\psi_Y(y_T)|} \end{aligned} \tag{4.27}$$

$$\begin{aligned} Z_{\mathcal{Y},T}(x'_T, y'_T, a, o_{T,Y}) &= p(o_{T,Y}|x'_T, y'_T, a) \\ &= \frac{\sum_{x' \in \psi_X(x'_T)} \sum_{y' \in \psi_Y(y'_T)} \sum_{o_y \in \psi_O(o_{T,Y})} p(o_y|x', y', a)}{|\psi_X(x'_T)||\psi_Y(y'_T)|} \end{aligned} \tag{4.28}$$

Note that ψ_X and ψ_Y could be different, but we used the same functions when we segmented both state variables. ψ_O is the same as ψ_Y, but has the *unknown* value added.

To speed up the process of finding a good policy, the final state can be defined as staying in the same super state independent of the action a: $p(x_{T,f}|x_{T,f}, y_{T,f}, a) = 1.0$ and $p(y_{T,f}|x_{T,f}, y_{T,f}, a, x_{T,f}) = 1.0$, where $(x_{T,f}, y_{T,f})$ is a final state. The final state is defined as either $y_{T,f}$ being on the base, or if $\exists x \in \mathcal{X} : x \in \psi_Y(y_{T,f}) \wedge x \in \psi_X(x_{T,f})$, i.e. the seeker is in the same superstate as the hider.

Top Reward The top reward function $R_T(x_T, y_T, a)$ (with $x_T = s_T$, the seeker's position; and $y_T = h_T$, the hider's position) can be defined as an average of the rewards of the bottom states:

$$R_{\text{avg},T}(s_T, h_T, a) = \frac{\sum_{s \in \psi_X(s_T)} \sum_{h \in \psi_Y(h_T)} R(s, h, a)}{|\psi_X(s_T)||\psi_Y(h_T)|} \tag{4.29}$$

However, also an explicit reward was tested:

$$R_{\text{simple},T}(s_T, h_T, a) = \begin{cases} 1, & \text{if } \exists s \in \psi_X(s_T) : s \in \psi_Y(h_T) \\ -1, & \text{if } b \in \psi_Y(h_T) \\ 0, & \text{otherwise} \end{cases} \tag{4.30}$$

4.7.3 The On-line Algorithm

The algorithm for solving the on-line two-level MOMDP is based on [Foka and Trahanias, 2007], and uses SARSOP to generate a policy [Kurniawati et al., 2008, Ong et al., 2010]. 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 [Ross et al., 2008]. Algorithm 4.6 and Figure 4.6(b) show 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 there on, the algorithm is run until a final condition is reached: some player has won or the time has passed. The segmented hider states are calculated in line 3; here we apply the robot centred segmentation (Figure 4.9). In line 4, the belief is compressed up using Eq. 4.25, thereafter, 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 Π_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 in line 10.

Algorithm 4.6 On-line two-level MOMDP planner.

1: $(b_Y, x) = \text{INITBELIEF}(M)$
2: **while not** finished game **do**
3: $\quad S_T = \text{SEGMENTSTATES}(M, x)$
4: $\quad b_{Y,T} = \text{COMPRESSBELIEF}(M, S_T, b_Y, x)$
5: $\quad M_T = \text{GENERATETOPLEVEL}(M, S_T)$
6: $\quad \Pi_T = \text{SOLVETOPLEVEL}(M_T, b_{Y,T}, x)$
7: $\quad a = \text{GETBESTACTION}(M_T, \Pi_T, b_{Y,T}, x)$
8: $\quad \text{DOACTION}(a)$
9: $\quad (o_y, x) = \text{DOOBSERVATION}(\ )$
10: $\quad (b_Y, x) = \text{UPDATEBELIEF}(b_Y, x, o_y)$
11: **end while**

4.8 Smart Seeker and Hider

As a comparison method we have created a greedy heuristic seeker and hider.

4.8.1 Smart Seeker

Using the previously defined triangle reward (Eq. 4.18) an automated heuristic greedy 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:

$$\pi_{\text{smart}}(s, h, b) = \arg\max_{a \in A} Q(\text{MOVE}(s, a), h, b) \tag{4.31}$$

where s is the seeker position, h the hider position, b the base, Q the score function Eq. 4.34, and MOVE the function that moves the seeker to a new state s' using an action (moving in one of eight directions, or staying at the same place). One action gets the seeker to a position, which can be used to calculate R_{tri} (Eq. 4.18); but, at the same time the hider can make a move, which we take into account by averaging the score over these possible (nine) moves:

$$q(s, h, b) = \sum_{h' \in \text{MOVES}(h)} R_{\text{tri}}(s, h', b) / |\text{MOVES}(h)| \tag{4.32}$$

where MOVES returns the list of possible moves by the hider at location h. 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 q is calculated for every possible hider's position and then averaged:

$$q_{\text{hidden}}(s, b) = \sum_{h \in \text{HIDDEN}(s)} q(s, h, b) / |\text{HIDDEN}(s)| \tag{4.33}$$

where HIDDEN returns the list of positions not visible to the seeker in the current map, i.e. the potential locations of the hider.

Finally, we can define the score function Q:

$$Q(s, h, b) = \begin{cases} q_{\text{hidden}}(s, b), & \text{if h=}\textit{hidden} \\ q(s, h, b), & \text{otherwise} \end{cases} \tag{4.34}$$

4.8.2 Smart Hider

Also a greedy heuristic hider has been created, the Smart Hider, which makes use of a similar greedy policy as Eq. 4.31:

$$\pi_{\text{smart}}(h, s, b) = \arg\max_{a \in A} Q(\text{MOVE}(h, a), s, b) \tag{4.35}$$

And in Eq. 4.32, instead of the reward R_{tri}, the following score is used:

$$R_{\text{hider}}(s, h, b) = D_{\max} - d_{hb} + \upsilon d_{sh} + noise \tag{4.36}$$

where $D_{\max} = \text{rows} \times \text{cols}$, d_{hb} is the shortest path distance between the hider and the base, and d_{sh} is the distance between the seeker and the hider, tuned by υ, which was found to work well for $\upsilon = 0.4$. The *noise* is uniform, has a maximum value of 2 m and is reduced as soon as the distance is less than 3 m, 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.

4.9 Simulations

In this section, the results of simulations with the different models are shown, and we explain the results of using different variants of the problem.

Movement The cells in the map are discrete, and are either an *obstacle*, *free* or the *base*. In the simulations, the agents do a step of 1 cell in one of eight directions, or they stay in the same location, and they are only allowed to be in free space inside the map. The simulations do not include neither acceleration, nor friction, nor collision, for simplicity.

Hiders As opponents we use a Random Hider, which moves randomly, and the Smart Hider as explained in subsection 4.8.2.

Maximum time To prevent the game from running endlessly, a maximum number of time steps was set, relative to the map size: $2(rows + cols)$, since in bigger maps it requires more steps to win. Reaching the maximum time without a winner is counted as a tie.

4.9.1 POMDP vs. MOMDP

Ong et al. [2010] mentioned as one of the biggest advantages of the MOMDP model and their used solver to be the short policy calculation time. And for the hide-and-seek problem we also found this to be true, and already for small maps of 5×6 cells.

For five different configurations of the map (i.e. different locations of obstacles), the policy calculation for the MOMDP was at least more than 140 times faster than for the POMDP policy.

4.9.2 Learn Transition Probabilities

Instead of using a uniform distribution of the transition probabilities for the hider, we can also use transition probabilities based on real experiments. In [Georgaraki, 2012], the probabilities were calculated for five small maps (5×6 cells) using data of hide-and-seek games played by humans against a *Smart* and *Random* hider. For each seeker-hider location the probabilities of the nine actions (moving in one of the eight directions or standing still) were calculated. Since not all transitions were covered, the missing ones were given a uniform probability. The results showed no significant difference (using Fisher's exact test), therefore, for the simulations and experiments we have used uniform transition probabilities for the hider's movement.

4.9.3 Two Simple Maps

More than 5000 simulated games were done on maps 1 and 2 (Figure 4.7) with two different resolutions (7×9 and 9×12), and with a maximum of 32 and 42 discrete time steps respectively (resulting in a *tie*). The opponents were the two automated hiders: Random Hider and Smart Hider. As models we used: the off-line MOMDP with the *simple* (Eq. 4.17) and the *triangle* (Eq. 4.18) reward, and the on-line Hierarchical MOMDP with the simple top reward (Eq. 4.30; the averaged reward will be tested in subsection 4.9.4), and the Smart Seeker.

4.9.3.1 Results

Table 4.1 shows that the off-line model with the *triangle* reward works best ($p < 0.001$; Fisher's exact test, two-sided, this has been used to check all the win statistics). 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

Table 4.1: 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	Reward	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	top	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	top	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	top	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	top	95.5%	4.3%	0.5%	400
		smart	–	95.0%	0.0%	5.0%	361
Total				96.9%	0.9%	2.2%	4475

difference was found in winning for the two map configurations (Figures 4.7(a) and 4.7(b)), nor for their sizes.

Table 4.2 shows the win statistics 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 or 42 (for the larger map) actions resulted in a *tie*. The off-line MOMDP model used the least amount of 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 also found that the seeker needed more steps on map 1 (Figure 4.7(a)) than on map 2 (Figure 4.7(b); $p < 0.001$; 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 was highest (see last column of Table 4.2), because the MOMDP model was calculated and a policy was learned at each time step; the average step time was lowest for the heuristic method ($p < 0.001$; Wilcoxon ranksum test, 2-sided).

The durations of the calculation of the off-line policies are shown in Table 4.3. Although the off-line method with triangle reward worked better than the on-line method, we can also see from Table 4.3, that the calculation of an off-line policy with triangle reward took relatively much more time. The on-line method required to calculate a policy at every time step, which for the 9×12 on average was 6.7 s, and on average it

Table 4.2: 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	Reward	Win	Lose	Tie	Total	Num. Act.	Dur./Act. [s]
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	top	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	top	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

Table 4.3: The time it took to calculate the policies off-line for the different maps, using the triangle or 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

took 12 steps to win (see Table 4.2), which resulted in approximately 78 s to complete a game. This is quite less than the calculation of the off-line policy for the triangle reward.

4.9.4 Larger Maps

In this section, we explain simulations done for a small map (map 3, Figure 4.7(c)) and larger sized maps (maps 3-4 in Figure 4.7). The *simple* (Eq. 4.17) and the *triangle* (Eq. 4.18) reward were used for the off-line MOMDP, and the *average* of the bottom *triangle* reward (Eq. 4.29) and *simple* top reward (Eq. 4.30; later referred to as *top rew.*) for the on-line Hierarchical MOMDP. The policy of the off-line MOMDP method was calculated beforehand with a maximum time of one hour.

4.9.4.1 Results

An overview of the win percentages of the different seekers can be seen in Table 4.4; in these simulations only maps until a size of 12×12 were used, because of the model's current limitations, and for the on-line MOMDP methods, a maximum learning time of 300 s per step was allowed. When we compare the different seeker models, the off-line MOMDP model won the most games, next the Smart Seeker and finally, the on-line MOMDP models ($p < 0.001$; Fisher's exact test, two-sided, this has been used to check all the win percentages). The off-line MOMDP with the triangle reward and the Smart Seeker won more often against the *Smart* hider, while the off-line method with the simple reward and the on-line methods won more against the Random Hider ($p < 0.01$). In all the results, the Smart Hider won most often, like expected, with a 12% win against a 1% of the Random Hider ($p < 0.001$).

Segmenting $\mathcal{X}$ For the on-line methods several parameters were tested. First, a reduction of the number of states, by also segmenting the fully observable space ($\mathcal{X}$), this gave no significant difference in the number of wins.

Learning time Policy When we limited the on-line policy learning time to 10 s, the win percentage were significantly less than when we limited them to 300 s.

More Segmentation Areas In the robot centred segmentation we, by default, used only the direction—and the location of the seeker, base, and hider if known—which gave at maximum 11 segments. We tested to add a layer more based on the distance (see Figure 4.9), which increased the number of segments to 19 at maximum; this gave marginally significant better results.

Map Size The map size did influence the results, the bigger the map the lower the win percentage, as can be seen in Table 4.5. Furthermore, for bigger maps the on-line MOMDP methods needed more time. Since the on-line MOMDP methods learn a policy on every time step, they take more time for bigger maps. The on-line MOMDP method with top rewards (1 or 0) took significantly longer on the 6×5 and 10×10 maps, but on the 12×12 maps the on-line method with average reward took more time

Table 4.4: The simulation statistics for the different models against the different hiders.

Seeker	Reward	Hider	Win	Lose	Tie	Total
off-line	simple	random	1224 (98.6%)	3 (0.2%)	15 (1.2%)	1242
		smart	1051 (86.6%)	162 (13.4%)	0 (0.0%)	1213
off-line	triangle	random	1317 (96.8%)	0 (0.0%)	43 (3.2%)	1360
		smart	601 (99.0%)	5 (0.8%)	1 (0.2%)	607
on-line	average	random	293 (93.9%)	15 (4.8%)	4 (1.3%)	312
		smart	109 (63.4%)	58 (33.7%)	5 (2.9%)	172
on-line	top	random	294 (96.7%)	3 (1.0%)	7 (2.3%)	304
		smart	124 (56.9%)	88 (40.4%)	6 (2.8%)	218
smart seeker	–	random	758 (89.5%)	0 (0.0%)	89 (10.5%)	847
		smart	455 (95.0%)	0 (0.0%)	24 (5.0%)	479
Total			6226 (92.2%)	334 (4.9%)	194 (2.9%)	6754

($p < 0.05$, Wilcoxon ranksum test, 2-sided; see Table 4.5). Games on the 40×40 maps were only tested against the Smart Seeker, because for the other methods the number of states was too big to be able to calculate a policy.

Time Segmenting also the $\mathcal{X}$ states seemed to take more time, but this was not significant. Using the on-line MOMDP with rewards redefined at the top took less time than the MOMDP with rewards averaged from the bottom level, but this is marginally significant. Using the extra layer of the robot centred segmentation method took significantly more time than only segmenting based on the direction.

The off-line MOMDP and the Smart Seeker took less time than the on-line MOMDP models ($p < 0.001$,Wilcoxon ranksum). Nonetheless, we should take into account that the off-line MOMDP method requires to learn the policy beforehand; this took from 1.4 s on average for the 6×5 maps to 1 hour (the set maximum) for bigger maps. The off-line MOMDP method won the games in, statistically less, steps than any of the other methods ($p < 0.001$,Wilcoxon ranksum; see Table 4.5). And using the simple reward resulted in winning in less steps than using the triangle reward ($p < 0.001$).

4.10 Real-life Experiments

Since the seeker (Dabo), was 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 to the hide-and-seek game. First of all, the robot and the person were only

Table 4.5: The win statistics per map size and seeker type played against both hiders. The last columns show the average ± standard deviation of the number of actions and the duration per action for won games.

Map Size	Seeker	Win	Lose	Tie	Total	Num. Act.	Dur. Act. [s]
6 × 5	off-line	97.0%	1.2%	1.8%	2249	5.49 ± 4.21	0.15 ± 0.13
	on-line	93.3%	6.7%	0.0%	268	5.06 ± 3.49	0.69 ± 0.62
	on-line (t.r.)	93.0%	6.6%	0.4%	242	5.71 ± 4.22	0.88 ± 1.36
	smart seeker	91.7%	0.0%	8.3%	360	7.59 ± 4.99	0.13 ± 0.09
10 × 10	off-line	96.6%	2.1%	1.4%	1407	11.48 ± 7.11	0.11 ± 0.08
	on-line	78.9%	17.6%	3.5%	142	14.67 ± 7.57	14.73 ± 17.69
	on-line (t.r.)	81.4%	15.3%	3.4%	118	13.49 ± 7.84	64.63 ± 69.38
	smart seeker	91.6%	0.0%	8.4%	856	13.97 ± 9.22	0.12 ± 0.1
12 × 12	off-line	85.2%	14.8%	0.0%	766	9.87 ± 6.0	0.11 ± 0.1
	on-line	54.1%	40.5%	5.4%	74	14.43 ± 12.41	92.83 ± 63.01
	on-line (t.r.)	59.9%	35.2%	4.9%	162	15.06 ± 11.26	70.53 ± 61.63
	smart seeker	92.0%	0.0%	8.0%	100	15.68 ± 12.11	0.1 ± 0.07
40 × 40	smart seeker	55.4%	0.2%	44.4%	448	44.59 ± 24.94	0.1 ± 0.15

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 neighbourhood, 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 and changed to the closest location on the map that was not an obstacle.

The observations were entered into the model, which then calculated the action to do. Then the person was 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 (32 steps, based on the map size). The win condition for the robot was adapted to its size, the robot won if it was within one cell distance of the hider.

The games were done in the FME environment (see Section 3.1) on maps 1 and 2 (Figure 4.7), and we used the off-line MOMDP method with both reward functions, the on-line hierarchical method with top reward and the Smart Seeker.

4.10.1 Results

With the real robot a total of 44 games were played against 15 adults, from which 12 games were won by the hider (person), and 32 by the seeker robot, see Table 4.6 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.

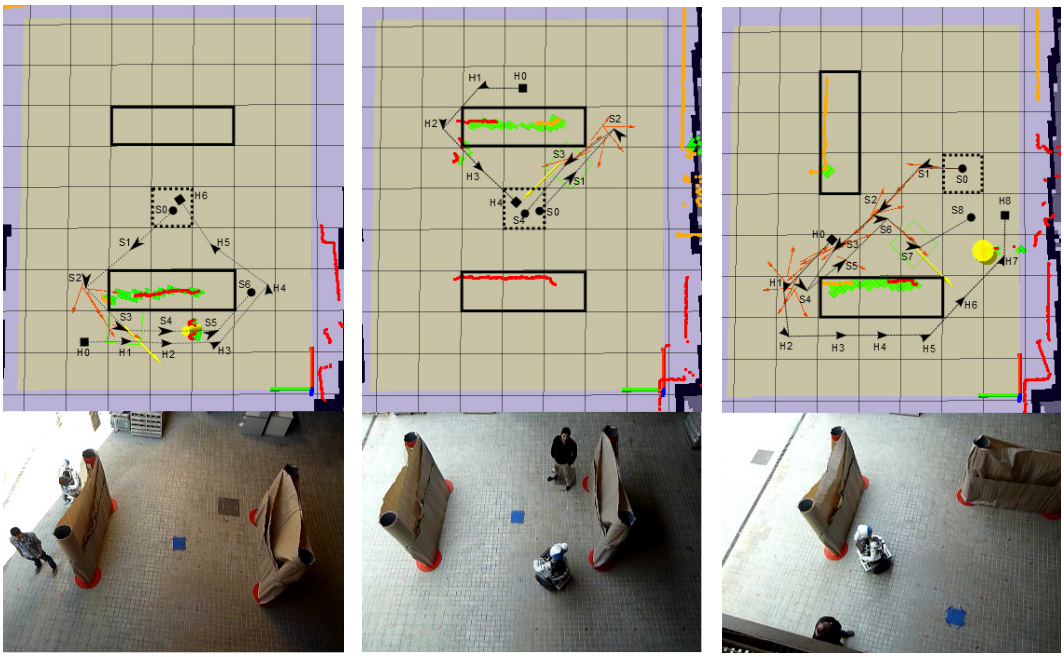
The average number of actions for won and lost games is shown in the last columns of Table 4.6, The only significant difference in the game results for map 1 and map 2 is that the games were won in significantly less steps on map 1 ($p < 0.001$; Mann-Whitney test).

Like in the simulations (subsection 4.9.3), 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 special strategies used by the human players.

Table 4.6: The results of the real-life experiments with the 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 him/her. The last two columns show the average number of actions it took the seeker to win or lose the game respectively.

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

Three games are shown in Figure 4.10, which show the map and the laser collision detections (of obstacles and walls). The light yellow area represents the game field. On top of the map the obstacles are shown as black rectangles and the base is shown as a dashed square. For both the seeker and hider steps are indicated with arrows and lines; S_0 is the seeker start place, which was always the base, and H_0 the hider's start place.



(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.

Figure 4.10: 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 the triangle reward was used in (b) and (c). 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 robot's goal position and the red arrows the robot's previous odometry.

The first image (Figure 4.10(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 4.10(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 4.10(c)) shows a game where the hider tried to get the seeker to follow him but when the distance to the base was too far, the seeker returned, and finally, caught the hider before reaching the base. Two videos in which the robot plays with the simple and triangle reward can be found on: http://alex.goldhoorn.net/thesis/hs-momdp/.

4.11 Discussion

In this chapter we have started to tackle the hide-and-seek problem by using RL methods. First, we confirmed that the MOMDP model, with the SARSOP solver, resulted in faster policy learning times than learning the policy for a POMDP. This is because of the separation of fully and partially observable state variables.

Next, we found that learning all the transition probabilities is not feasible, because we need a large amount of experimental data and there are many locations which were not visited during the experiments done by the human subjects. Therefore, we decided to continue using a uniform probability over all actions.

When we look at the artificial opponent in the simulation, the Smart Hider and the Random Hider, we found that the Smart Seeker and the MOMDP models with the *triangle* reward won most. While for the Random Hider, using the simple reward resulted in a higher win rate. This can be explained because the *triangle* reward is based on the same heuristic as the Smart Hider, whereas the simple reward only takes into account the final states.

The on-line hierarchical method was proposed to reduce the number of partially observable states, and thereby tackling the *curse of dimensionality*. Even though the 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 they used a strategy in which they "misled" the robot by leading him around an obstacle, and thereby they won. Another disadvantage of the on-line method was its policy calculation time, which, for the bigger maps, resulted in too slow reaction times to work in real-life. The off-line policy calculation resulted in much faster reaction times, but was limited by the state space.

4.12 Conclusions

This chapter has explained the first step towards a search-and-track method, specifically, we have analysed 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 were extensively tested in simulation, and initial experiments were 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 well, and the best method was the off-line MOMDP model with the triangle reward.

To reduce the state space, we proposed an on-line hierarchical method, which allows to handle larger state spaces, since it segments the state space in larger state groups. In simulation, the on-line method was found to give good results, but in the real-life experiments the results were much worse than for the other methods. Furthermore, the on-line calculation of the policy resulted in a slow robot.

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 working in 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 larger real-world environments.

Chapter 5

Search-and-Track in Large Continuous Environments

While in the previous chapter we have focused on the hide-and-seek game in a small environment, here, we extend it to a realistic problem of search-and-track in real urban environments with our robot Dabo. We propose several methods that can handle real-time searching and tracking of people in an urban area, and which are shown by real-life experiments.

The methods do an estimation of the position of the person in a continuous state map. One method is based on the Reinforcement Learning method Partially Observable Monte-Carlo Planning (POMCP), which we apply while making use of continuous states. The second method uses a Particle Filter to estimate the person's position. This location estimate is then used to decide where the robot has to go to, to search for the person.

Finally, also dynamic obstacles—that represent other people or objects moving in the environment—are introduced, not only for simulation, but also to predict the location of the person.

The methods presented in this chapter brought the following contributions:

- The CR-POMCP, which is a POMCP that generates a policy in continuous space, runs in real-time, has always a minimum number of belief points, and runs in large state spaces [Goldhoorn et al., 2014, 2017b].

Figure 5.1: Dabo performs the search-and-track task with a target (wearing a tag for recognition) in different urban environments.

- Using the Highest Belief (HB) of the belief of the CR-POMCP allowed us to do search-and-track in a real-life environment with Dabo [Goldhoorn et al., 2014, 2017b].
- The HB-CR-POMCP Searcher & Tracker improves the previous method when the person is visible [Goldhoorn et al., 2014, 2017b].
- Furthermore, a PF method was introduced to keep track of the location of the person, which resulted in the methods HB-PF and HB-PF Searcher & Tracker [Goldhoorn et al., 2017b].
- Dynamic obstacles are simulated and used in the models to improve the prediction of the person's location [Goldhoorn et al., 2017b].
- Real-life experiments were done [Goldhoorn et al., 2014, 2017b].

5.1 Introduction

In this chapter, we start by changing the previously used RL method, MOMDP, which can only be used for small state spaces due to the computational and memory complexity. The presented methods in this chapter, however, are able to work in continuous space, large environments (tested until an area of 2188 m^2) and in real-time.

Our goal is to search and track a person in an urban environment that has static and dynamic obstacles. We have already seen in Chapter 2 that there exist different good techniques for searching or tracking only, but few which do both simultaneously.

We make use of a Monte-Carlo method, POMCP, that creates a policy by doing many simulations. The CR-POMCP method works in continuous state space and with large maps, where the belief is represented by a limited number of points. To improve the method in the real-life experiments, it is extended to use the Highest Belief (HB) points, which are the most probable locations of the person according to the belief. As next method, we present another way of generating the probability map (belief) of the person's location by using a Particle Filter (PF).

Our presented methods are able to work in urban environments with dynamic and static obstacles; furthermore, they take into account sensor uncertainty, false detections of the person and lack of a person's detection. Additional considerations are required to make the system work properly, for example, Gaussian sensory noise is inevitable in real-life situations, and false negative and false positive detections tend to occur. The presented methods take the first two problems into account, and can handle situations with short false positive detections.

Finally, the validation of the method is accomplished throughout an extensive set of simulations and real-life experiments, see Figure 5.1. For the later, we accomplished more than three km of autonomous navigation, with almost six hours of successful results during several weeks of testing and experimentation.

We start this chapter by explaining some background about the used methods (POMCP and PF), then an overview of the approach is given. In Section 5.4 some additional information about the experimental setup is given, and then we start with the explanation of the methods: the CR-POMCP in Section 5.5, the HB-CR-POMCP Searcher & Tracker in Section 5.6 and the HB-PF Searcher & Tracker in Section 5.7. Next, the simulations and real-life experiments are discussed, and finally a discussion and the conclusions are given.

5.2 Background

In Section 4.2, POMDPs were explained in detail, in this section, models with continuous state space will be described, first Continuous POMDPs, then POMCPs, and finally, we will explain two position estimation methods: the Kalman Filter (KF) and the Particle Filter (PF).

5.2.1 Continuous POMDP

The problem of using continuous states for POMDPs is that the belief over a discrete state space is already continuous. There are mainly two ways that solve this problem, either by using the Gaussian Mixture Model (GMM), which is a finite list of Gaussians, or by using particles and Monte-Carlo simulations.

Porta et al. [2006] presented a framework to find a policy for POMDPs with continuous states, actions and observations. They show that the Value Function for the continuous POMDP is convex in the beliefs over continuous states. Authors define the continuous states, observations, transition function, and reward function using a GMM, and for the belief a GMM or particles.

Van Den Berg et al. [2012] introduced a method in which the belief is modeled using Gaussians, the belief dynamics are approximated using an EKF (see subsubsection 5.2.3.1). The approach has a complexity of $O(n^4)$ for n dimensions, it however has some limitations, such as requiring Gaussian distributions, and the dynamics, observations and rewards should be smooth.

Bai et al. [2014] used Monte-Carlo simulations (based on the MCVI [Bai et al., 2011], which will be discussed in the next section) to handle continuous state and observation spaces. They present a Generalized Policy Graph (GPG) which is a policy graph, with each node having an action and edges representing observations. The GPG is constructed by applying the Bellman backup equation, which is done using Monte-Carlo sampling, to cope with continuous state and observation spaces.

5.2.2 Partially Observable Monte-Carlo Planning

In the previous chapter, we have seen that the main problem of the POMDP models is that calculating an optimal policy is computationally too complex for large state spaces, since the search and memory requirements grow exponentially.

Instead of calculating the exact policy, we have seen that approximations, such as [Kurniawati et al., 2008] can be used. Another approach is used by Monte-Carlo Value Iteration (MCVI) [Lim et al., 2011], in which the expected reward is calculated over a random set of samples instead over all states. MCVI is used in [Bai et al., 2011] to calculate a policy for POMDPs. They generate a policy graph to avoid having to

handle the continuous belief space; instead of going through all states to calculate the expected reward, they randomly sample a number of states.

Silver and Veness [2010] go even a step further by using Monte-Carlo simulations to generate a policy, without exactly calculating the expected reward (Eq. 4.8). They present the Partially Observable Monte-Carlo Planning (POMCP), which is based on Monte-Carlo Tree Search (MCTS) [Kocsis and Szepesvári, 2006] and POMDPs. MCTS is a planning algorithm that estimates the expected reward by doing Monte-Carlo simulations, whereas MCTS is made for the fully observable MDPs, POMCP handles partial observability. Instead of defining the complete transition probability T and observation probability Z, the POMDP simulator $(s', o, r) = \mathcal{G}(s, a)$ is used; which returns a new state s', observation o, and reward r based on a current state s, and action a. Moreover, the belief of the POMCP is represented by a list of n_{belief} states instead of a belief probability for each possible state. To create the policy, n_{sim} Monte-Carlo simulations are done, the best action is chosen, the belief is updated and, finally, the learning process continuous with the new belief.

The big advantages of POMCPs are the complexity reduction, and the easier and more flexible definition of the POMDP model. The POMCP's complexity does not depend on the number of states nor observations, but it depends on the POMDP simulator. Furthermore, the POMCP algorithm tackles the *curse of history* and the *curse of dimensionality* by doing Monte-Carlo simulations using a POMDP simulator, instead of the fully defined model. The first curse is solved by doing a limited amount of Monte-Carlo simulations that each follow a single path—until an end state or limited path depth—without trying other paths on the policy tree, and therefore, the search space does not grow exponentially with the search depth. The second curse is solved by the limited number of samples that represent the belief. Convergence of the POMCP policy with finite horizon is proven in [Silver and Veness, 2010], and can be extended to the infinite case as shown in [Kocsis and Szepesvári, 2006].

In the next subsection the algorithm is explained in more detail for the discrete case, as shown in [Silver and Veness, 2010].

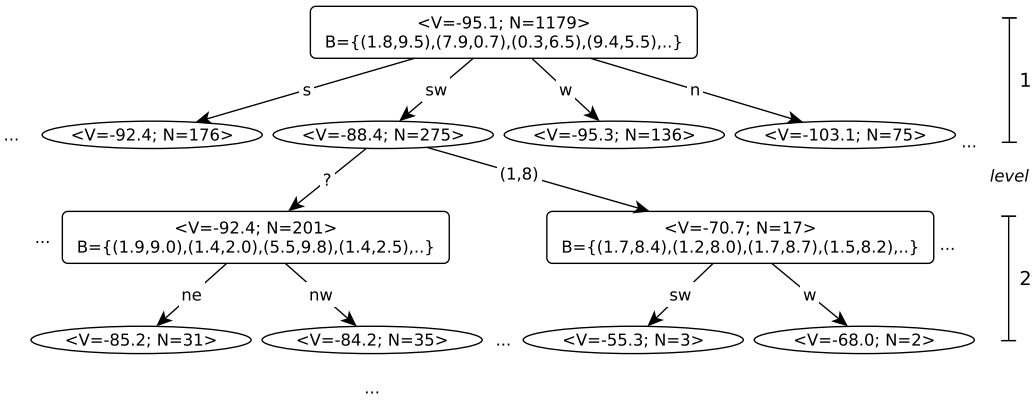


Figure 5.2: A part of a policy tree generated by the POMCP solver during the search-and-track task for the situation in Figure 4.1-*top*. *Belief nodes* are rectangular, *action nodes* are oval. For each action (here representing a movement: **n**orth, **n**orth**e**ast, **s**outh, etc.) a child node is reached. From the *action nodes* a *belief node* is reached with an observation ('?' being *hidden*). All nodes contain an expected value V and the number of times N this value has been updated. B is the belief and is only maintained in *belief nodes*. For clarity, in this figure only the person's position is shown in the observations and beliefs, and not the robot's position.

5.2.2.1 Algorithm

The POMCP algorithm generates a policy tree, which has two types of nodes: *belief nodes*, representing a belief state; and *action nodes*, which are the belief nodes' children, and are reached by doing an action. The root is a *belief node* and contains the belief of the current situation. Figure 5.2 shows part of the policy tree of the situation in Figure 4.1-*top*. In POMDPs, as discussed previously, the belief is the probability of being in each of the possible states. In POMCPs the belief is represented by a list of states, which represents the possible real state (see Figure 5.2). When a state is more probable, then the state is repeated more times in the belief.

Whereas POMDP solvers mainly use Value Iteration (Eq. 4.8) to create a policy, the POMCP solver does (n_{sim}) Monte-Carlo simulations. Each node in the tree keeps track of the average expected reward V, and the number of times N, a simulation passes through the node.

The learning process is summarized in Figure 5.3, it starts with an initial observation

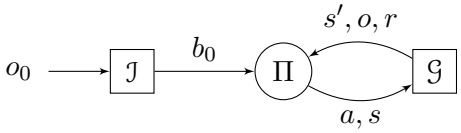


Figure 5.3: Schema of the POMCP learning process.

Table 5.1: The parameters used in the POMCP algorithm.

Parameter	Description
γ	Discount factor.
n_{sim}	Number of Monte Carlo simulations.
n_{belief}	Minimum number of states in the belief.
c	Exploration constant.
e_{count}	Expand count.
d_{max}	Maximum tree depth.

o_0. The simulator function $\mathcal{I}(o_0)$ generates an initial belief b_0, which is stored in the tree root's node. With the states from the initial belief, the current policy π is used to choose the action. The POMDP simulator $\mathcal{G}$ outputs a new state, observation and reward, which are used to update the policy, and to continue the simulation (explained in detail later on in Section 5.5).

Before each robot's step, the policy tree is generated and updated as shown in Algorithm 5.7. It starts with LearnTree(*Root*), which does n_{sim} simulations. An overview of the parameters used in the algorithm is shown in Table 5.1.

Each step in the Monte-Carlo simulation starts at a randomly chosen state s from the root's belief (line 3). SimNode simulates a step by first choosing the action that gives the highest reward (line 10). To prevent the model from only exploiting the current policy, the last term in the equation in line 10 is introduced, which is weighted by the exploration constant c. The knowledge is greedily exploited when $c = 0$, by choosing always the action with the highest value; and with $c > 0$, different actions are tested. These are both applied in MCTS [Kocsis and Szepesvári, 2006] and POMCP [Silver and Veness, 2010]. The exploration constant was set to $c = r_{\text{hi}} - r_{\text{low}}$, as suggested by Silver and Veness, where r_{hi} and r_{lo} are respectively the highest and lowest reward values returned. Action nodes, which have not been explored yet, are given a value of ∞ in line 10, to ensure that they are chosen at least once.

Next, in line 11, the belief of the second level *belief node* (the root being first) is

Algorithm 5.7 The POMCP solver. Accessing child nodes is noted as $Node[a]$ (for action a for example).

function LEARNTREE($Node$)
  for $i = 1$ to n_{sim} **do**
    $s \sim Node.\mathcal{B}$
    SIMNODE($Node$,s,0)
  end for
end function
function SIMNODE($Node$,s,$depth$)
  if depth $> d_{\max}$ **then return** 0
  else
    $a = \text{argmax}_a \left(Node[\text{a}].V + c\sqrt{\frac{\log(\text{Node}.N)}{\text{Node[a].N}}} \right)$
    if depth $= 1$ **then** $Node.\mathcal{B} = Node.\mathcal{B} \cup \{s\}$
    $(s', o, r_{\text{immediate}}) = \mathcal{G}(s, a)$
    if s' is **not** final **and not** $Node$[a][o] exists **and**
        $Node[\text{a}].N \geq e_{\text{count}}$ **then**
      Add $Node$[a][o]
    end if
    if s' is **not** final **then**
      if $Node$[a][o] exists **then**
        $r_{\text{delayed}} =$ SIMNODE($Node$[a][o],s',$depth$+1)
      else
        $r_{\text{delayed}} =$ ROLLOUT(s',$depth$+1)
      end if
    else
      $r_{\text{delayed}} = 0$
    end if
    $r_{\text{total}} = r_{\text{immediate}} + \gamma r_{\text{delayed}}$
    $Node[\text{a}].N = Node[\text{a}].N + 1$
    $Node[\text{a}].V = Node[\text{a}].V + \frac{r_{\text{total}} - Node[\text{a}].V}{Node[\text{a}].N}$
    $Node.N = Node.N + 1$
    $Node.V = Node.V + \frac{r_{\text{total}} - Node.V}{Node.N}$
    return r_{total}
  end if
end function
function ROLLOUT(s,$depth$)
  if depth $> d_{\max}$ **then return** 0
  else
    $a \sim \pi_{\text{rollout}}()$
    $(s', o, r) = \mathcal{G}(s, a)$
    return $r + \gamma$ ROLLOUT(s',$depth$+1)
  end if
end function

expanded with the passed state s. The *belief nodes* of a deeper level are not created, because the amount of nodes grows exponentially with the depth, and for those nodes the probability of being pruned out is too high.

The simulator is called with the current state and action in line 12. With action a and observation o the new node in the tree is reached (see Figure 5.2), if it exists, otherwise it is added (lines 13-16) if the new state s' is not final and the parent action node has been visited at least e_{count} (expand count) times; this last criterion is to prevent the tree from growing too fast. The new node's value V is initialized to 0, but a heuristic value can be used.

If the new state s' is not final, it continuous the simulation in line 19 if the new node exists, otherwise a *rollout* is done. The ROLLOUT function (line 34) is executed when there is no node in the policy tree for the observation and action yet, it then uses a *rollout* policy π_{rollout} [Silver and Veness, 2010] until the maximum depth has been reached. This policy can be uniformly random action selection (as suggested by [Silver and Veness, 2010]), but it can also be based on a heuristic.

After the simulation step, the reward r_{total} for the current belief and action node are calculated (line 26), where the influence of the future reward r_{delayed} is weighted with the discount factor γ to reduce the importance of possible future rewards.

For the POMDP models, the expected reward for each of the possible actions, normally is calculated using the Bellman equation Eq. 4.8. In POMCPs, these are calculated by averaging the rewards (line 26) obtained by doing Monte-Carlo simulations. The action and observation node's number N and value V are updated in lines 27-28, and lines 29-30 respectively.

The simulation continues until a maximum amount of steps (d_{max}, line 8 and 35), or until s' is a final state. Finally, after the policy tree has been learned, the action to execute can be chosen from the tree: $a = \text{argmax}_a Root[\text{a}].V$. When the robot has executed its action and has done the new observation, a new root is chosen, this is called the *update* process.

Using the already learned policy tree we can get the child node through action a (the planned one—or better, the one really done—since due to, for example obstacle avoidance, the action can be different) and observation o. If, for example, action *sw*

(south-west) has been executed, and observation *hidden* ('?') has been done for the tree in Figure 5.2, then this would reach the *belief node* left-under. This new node is then taken as root, and if this child *belief node* does not exist, a new *belief node* is created.

During the simulations the beliefs of the second level observation nodes already have been grown (line 11 in Algorithm 5.7), therefore, the newly chosen belief will (most probably) already have some belief points. However, to assure a minimal spread of the belief over the possible person's locations, a minimum number of n_{belief} states in the belief are maintained. If the new root's belief includes less than n_{belief} states, then new states are generated for the belief using the POMDP simulator: $(s', o_{\text{sim}}, r) = \mathcal{G}(s, a)$, where s is sampled from the previous root's belief. The state s' is stored in the new root's belief if the real observation matches: $o = o_{\text{sim}}$. If there is no belief in the new root, an initial belief can be calculated from the new observation with $\mathcal{J}$, but this results in losing the knowledge about the state (belief).

Finally, a new learning process is started again from the newly chosen root, thereby, continuing until the final state has been reached.

5.2.3 Filters

Bayesian Filters have been used for many years to estimate the state (position) of an agent, based on an observation with sensor noise, for example, in robotics [Thrun et al., 2005]. A *Bayesian Filter* [Thrun et al., 2005] is used to estimate the state s_t, based on an observation o_t, action a_t and previous state s_{t-1}:

$$\bar{b}(s_t) = \int P(s_t|a_t, s_{t-1}) b(s_{t-1}) ds_{t-1} \tag{5.1}$$

$$b(s_t) = \eta P(o_t|s_t) \bar{b}(s_t) \tag{5.2}$$

where η is a normalization: $\eta = 1/\int P(o_t|s)\bar{b}(s)ds$. As can be seen, there are two phases, first a *prediction* (Eq. 5.1) is done based on the done action a_t at the previous state s_{t-1}, then a *correction* (Eq. 5.2) is done using the previous estimate and the observation o_t.

Examples of Bayesian filters are Kalman Filters and Particle Filters, which will be explained next.

5.2.3.1 Kalman Filter

A Kalman Filter (KF) [Thrun et al., 2005] assumes that the state is linear stochastic:

$$s_t = A_t s_{t-1} + B_t a_t + \epsilon_t \tag{5.3}$$

where A_t describes the state movement from one time step to another, B_t describes the control step change, and ϵ_t is an independent and normally distributed noise with covariance R_t. The measurement variable is defined as:

$$o_t = C_t s_t + \delta_t \tag{5.4}$$

with C_t being a matrix indicating how to map the state to an observation, and δ_t is an independent normally distributed noise with covariance Q_t.

Algorithm 5.8 A Kalman Filter.

1: **function** KalmanFilter($\mu_{t-1}, \Sigma_{t-1}, a_t, o_t$)
2: ▷ **Prediction**
3: $\bar{\mu}_t = A_t \mu_{t-1} + B_t a_t$
4: $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$
5: ▷ **Correction**
6: $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$
7: $\mu_t = \bar{\mu}_t + K_t(o_t - C_t \bar{\mu}_t)$
8: $\Sigma_t = (I - K_t C_t)\bar{\Sigma}_t$
9: **return** μ_t, Σ_t
10: **end function**

The KF starts with an initial mean (μ_0) and covariance (Σ_0), and like in the *Bayes Filter*, the KF also has a *prediction* and *correction* phase, as shown in Algorithm 5.8. First, the mean (μ) and covariance (Σ) are updated in lines 3 and 4. Then *correction* is done by first calculating the Kalman gain K (line 6), which takes into account the projection of the state to observation.

In cases where linearity cannot be guaranteed, the Extended Kalman Filter (EKF) can be used, which does not have a linear Gaussian transition and sensor model. The EKF models the states and observations as functions (g and h) instead of the linear model in Eq. 5.3 and Eq. 5.4:

$$s_t = g(a_t, s_{t-1}) + \epsilon_t \tag{5.5}$$

$$\Sigma_t = h(s_t) + \delta_t \tag{5.6}$$

The EKF does linearisation, and does not preserve normality of the distributions of the state and measurement, therefore it is only reliable for almost linear systems.

5.2.3.2 Particle Filter

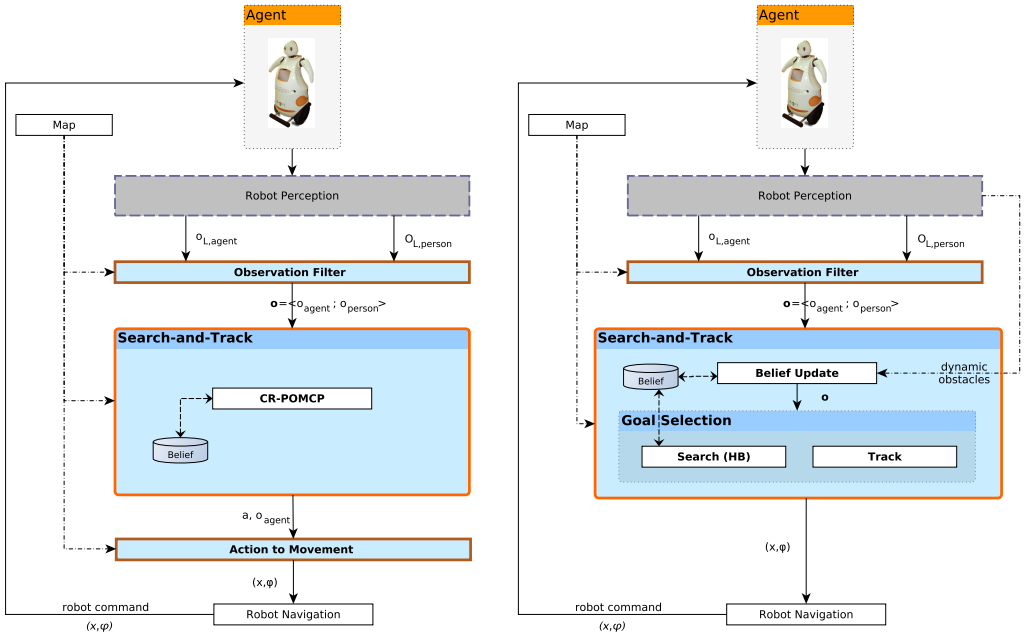
A Monte-Carlo method of doing state estimation is the Particle Filter (PF) [Thrun et al., 2005], which uses particles that represent the state, and the state is updated in two phases: a *prediction* and an *update* phase.

Algorithm 5.9 shows that there are different steps, first a *prediction* step (line 5), then the weight is calculated based on the observation probability given the state (line 6), and finally, resampling is done (line 11) based on the weight.

Algorithm 5.9 A basic Particle Filter.

1: **function** ParticleFilter(S_{t-1}, a_{t-1}, o_t)
2: $\bar{S}_t = S_t = \emptyset$
3: ▷ **Prediction**
4: **for** $i = 1$ to $n_{\text{particles}}$ **do**
5: sample $s_t^i \sim p(s_t|s_{t-1}^i, a_{t-1})$
6: $s_{t,w}^i = p(o|s_t^i)$
7: $\bar{S}_t = \bar{S}_t \cup \{s_t^i\}$
8: **end for**
9: ▷ **Update**
10: **for** $i = 1$ to $n_{\text{particles}}$ **do**
11: sample $s_t^i \in \bar{S}_t$ with probability $s_{t,w}^i$
12: $S_t = S_t \cup \{s_t^i\}$
13: **end for**
14: **return** S_t
15: **end function**

One of the advantages of a PF is that the complexity is linear with the number of particles $n_{\text{particles}}$. Furthermore, it is a non-parametric method that puts no requirements on the distribution of the localizations, whereas the KF requires a Gaussian distribution.



(a) The CR-POMCP.

(b) The Highest Belief methods (using CR-POMCP or PF).

Figure 5.4: The schema of the different processes in the search-and-track methods presented in this chapter. (a) shows the RL method CR-POMCP, (b) shows the HB method, which uses either the CR-POMCP's or PF's belief.

5.3 Overview of the Approach

In this chapter, several methods are presented to search-and-track a person using a robot. An overview of the architecture is shown in Figure 5.4, where (a) uses the RL method CR-POMCP, and (b) uses the belief to find the location with the Highest Belief. The *Robot Perception* and *Robot Navigation* have already been explained in Section 3.2.

The first proposed method CR-POMCP (Figure 5.4(a), Section 5.5) creates a policy on-line, and uses it to find the best action (a) to execute. This action is transformed into a location by the *Action to Movement* module.

The other methods make use of a belief, the probability map of the person, see Figure 5.4(b). We use two methods to calculate the belief: the CR-POMCP (Section 5.5) and the PF (Section 5.7). Then, the robot goal is selected, which depends on the ob-

servation of the person; if he/she is visible, the robot does tracking using the observed position, otherwise *search* is done. The *search* uses the belief to calculated the location with the highest belief, which is then selected as goal (see Sections 5.6 and 5.7).

Finally, the *Robot Navigation* module calculates the path to the selected goal, which is sent as motion commands to the robot.

5.4 Experimental Setup Improvements

In Chapter 3, the experimental settings have been commented, but with respect to Chapter 4, we have improved some of the simulations, such as adding a visibility probability, and the use of dynamic obstacles.

5.4.1 Visibility Probability

Whereas previously, we modelled the visibility to be infinite, only being blocked by obstacles; here we introduce a visibility probability function, as shown in Figure 5.5. The probability of seeing s_1 from s_2 is given by:

$$P_{\text{vis}}(s_1, s_2) = \begin{cases} 0, & \text{if } \textsc{ray}(s_1, s_2) \text{ not free} \\ p_{\text{v,max}}, & \text{if } d < d_{\text{v,max}} \\ \max(0, p_{\text{v,max}} & \\ \quad -\alpha_{\text{vis}}(d - d_{\text{v,max}})), & \text{otherwise} \end{cases} \tag{5.7}$$

where RAY is the ray tracing function, which makes use of the discrete map; $d = \|s_1 - s_2\|$; $p_{\text{v,max}}$ is the maximum visibility probability; $d_{\text{v,max}}$ is the distance until which it has a maximum visibility probability; and α_{vis} is the slope with which the probability reduces. The parameters values were tuned based on real world data, and are shown in Table 5.6.

The visibility probability is also used to simulate observations, i.e., an observation with the real person's location is returned with a probability of $P_{\text{vis}}(s_1, s_2)$, otherwise an empty observation is returned.

In the case the ray tracing algorithm detects an obstacle before the area we want to detect, it is given a probability of 0. This is done for static obstacles, but can also be done for dynamic obstacles when they are detected, for example for other people walking around.

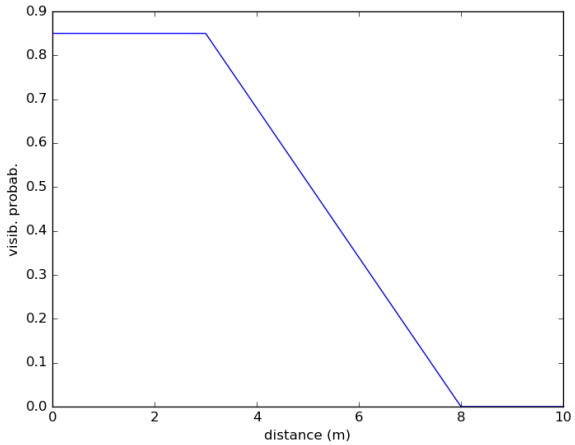


Figure 5.5: The visibility probability of seeing a person from a specific distance in an obstacle free area, as indicated by Eq. 5.7 (using the parameter values from Table 5.6).

5.5 Continuous Real-time POMCP

There are different ways of handling continuous state space in POMDPs, as commented in subsection 5.2.1, but these either model the state space using a GMM or by using particles. Since the use of a GMM is not that obvious nor easy, we have chosen the particle method. Furthermore, the particle method, POMCP, is able to handle large state spaces, and does not require a complete POMDP model to be defined, since only a simulator of the POMDP is required (as explained in subsection 5.2.2). This spares on having to define the transitions as a probability function T, and avoids handling a belief space of infinite dimensions.

In order to use continuous states in POMCPs, three things have to be done:

1. define the continuous states;
2. create a POMDP simulator that handles these states;
3. assure that the observations are discretized.

When states represent a position on a map, then in the discrete case the position is a grid cell, whereas in the continuous case they can have any value. Only the actions and observations have to be discrete to prevent the policy tree from growing too wide,

and to assure that the values in the nodes can converge. Balancing the observation discretization is important, because a too detailed discretization of the observations would result in a too wide policy tree, and therefore would require more simulations. On the other hand, less detail might cause problems in the precision of planning.

In the case of using positive rewards for each state, running through more states will be seen as being better, since it gives a higher expected reward. To prevent this, we present the use of an averaging instead of summation for line 26 of Algorithm 5.7:

$$r_{\text{total}} = \frac{r_{\text{immediate}} + \gamma r_{\text{delayed}}}{(1+\gamma)} \tag{5.8}$$

5.5.1 Search-and-track POMDP Simulator

For the search-and-track problem, states $(s_{\text{agent}}, s_{\text{person}})$ are defined as the position of the agent (s_{agent}) and person we want to find (s_{person}). There are nine actions which can be done: eight directions and staying at the same position. The observations are the same as the states, except when the person is not visible, then the person's observation (o_{person}) is a special predicate: *hidden*. Our reward function, $-d_{ap}$, is increasing when the agent-person distance d_{ap} is decreasing.

As mentioned previously in subsection 5.2.2, POMCPs do not require the definition of all the probability matrices of the POMDP, but they need a POMDP simulator with two functions: one to create an initial state ($\mathcal{I}$), and another for the next step ($\mathcal{G}$).

The initialisation of the belief is shown in Algorithm 5.10, which for the search-and-track problem is calculated based on the visibility of the person (Eq. 5.7). Initial states are generated with the function $s = \mathcal{I}(o_0)$, where o_0 is the initial observation. The generated state is equal to the observation (with added Gaussian noise), but when the observed person's position is *hidden*, then s_{person} is chosen randomly from hidden positions from o_{agent}, thereby using the visibility probability P_{vis}.

The second function generates a new state based on the current state and action: $(s', o, r) = \mathcal{G}(s, a)$, see Algorithm 5.11. The new agent's position s'_{agent} depends on the action and the agent's position s_{agent}. The person's movement is modelled as random, because we do not know where the person is going to, but its movement could be modelled as heuristic or based on experience. The function MOVE moves the agent

Algorithm 5.10 The initialisation algorithm for the belief, with as input observation o_0.

```
1: function I(o_0)
2:     B_0 = ∅
3:     for i = 1 to n_belief do
4:         if o_{0,person} = hidden then
5:             do
6:                 sample s_person from the free cells of map
7:             while P_vis(s_person, o_{0,agent}) > PRAND( )
8:         else
9:             s_person = N(o_person, σ_person I)
10:        end if
11:        B_0 = B_0 ∪ {⟨o_{0,agent}, s_person⟩}
12:    end for
13:    return B_0
14: end function
```

with action a; MoveRandom moves the person in a random direction, both a distance of at maximum 1 cell. The observation o equals the new state s', except for o_{person}, being *hidden* when the person is not visible according to a ray tracing algorithm. The reward function is the negative shortest path distance (i.e. taking into account obstacles) between the robot and person d_{ap}.

To make the POMDP simulator resemble the real world, we have added Gaussian noise to the positions in the new state s' and observation o. Different standard deviations are defined for the next seeker state (σ_{agent}), next person state (σ_{person}), seeker observation ($\sigma_{\text{obs,agent}}$) and person observation ($\sigma_{\text{obs,person}}$). The noise in the new state s' simulates the not perfect actuators of the robot, and a general difference in speed of both agents; whereas the observation noise represents the noise of the sensors. Adding the noise also prevents particle deprivation, which is present in most particle filters [Thrun et al., 2005].

Besides noisy signals, *false negative* and *false positive* detections can occur, the first are simulated by converting a person's observation o_{person} in *hidden*, with probability $p_{\text{false\_neg}}$, and the second by returning a position for o_{person} when it is *hidden*. A *false positive* could have occurred because of sensory noise, or because the person was visible through an obstacle which did not block the robot's view completely. Therefore, the person's position is returned with a probability of $p_{\text{see\_occ}}$ and otherwise, a random

Algorithm 5.11 The POMDP simulator used to learn the POMCP policy. PRAND generates a uniformly distributed random number between 0 and 1.

function $\mathcal{G}_{\text{S\&T}}(s, a)$
  $s'_{\text{agent}} = \text{MOVE}(s_{\text{agent}}, a)$
  $s'_{\text{person}} = \text{MOVERANDOM}(s_{\text{person}})$
  $o_{\text{agent}} = s'_{\text{agent}}$
  if $\text{VISIBLE}(s'_{\text{agent}}, s'_{\text{person}})$ **then**
    if $\text{PRAND}(\,) < p_{\text{false\_neg}}$ **then**
      $o_{\text{person}} = \textit{hidden}$
    else
      $o_{\text{person}} = s'_{\text{person}}$
    end if
  else
    if $\text{PRAND}(\,) < p_{\text{see\_occ}}$ **then**
      $o_{\text{person}} = s'_{\text{person}}$
    else if $\text{PRAND}(\,) < p_{\text{false\_pos}}$ **then**
      $o_{\text{person}} = \text{RANDOMPOSITION}()$
    else
      $o_{\text{person}} = \textit{hidden}$
    end if
  end if
  $r = -\text{SHORTESTPATHDIST}(s'_{\text{agent}}, s'_{\text{person}})$
  $s'_{\text{agent}} = \mathcal{N}(s'_{\text{agent}}, \sigma_{\text{robot}} I)$
  $s'_{\text{person}} = \mathcal{N}(s'_{\text{person}}, \sigma_{\text{person}} I)$
  $o_{\text{agent}} = \mathcal{N}(o_{\text{agent}}, \sigma_{\text{obs,agent}} I)$
  $o_{\text{person}} = \mathcal{N}(o_{\text{person}}, \sigma_{\text{obs,person}} I)$
  return (s', o, r)
end function

position is returned with a probability of $p_{\text{false\_pos}}$.

The complexity of the simulator $\mathcal{G}$ is $O(1)$, because we assume the visibility check, distance check and probability visibility check to be of constant time (because they are cached; see Section 3.6). The complexity of the update function of the POMCP (Algorithm 5.7) is therefore $O(n_{\text{belief}})$.

5.6 HB-CR-POMCP Searcher & Tracker

Experiments with the CR-POMCP showed that our robot, Dabo, moved slowly, and therefore, was not able to search and track the person in real-time. These issues were

because of:

1. *Small step actions*: the RL methods return an action which is a step of about 1 m, which resulted in acceleration and deacceleration before having reached a reasonable speed, therefore, the final movement per step was slow.
2. *A limited amount of movement directions*: the discrete actions allowed the robot only to move in eight directions, see Figure 5.6(a).
3. *Quickly changing actions*: the actions—and thereby the directions—were often changed from one step to another.

The first two limitations are enforced by the planning method, because increasing the number of actions causes an exponential growth of the search tree. The third issue was due to the noise present in the policy learning—specifically the limited number of Monte-Carlo simulations—and a relatively small difference of an action like *north* and *northeast*, as can be seen in Figure 5.6(a).

One way of avoiding these small steps and moving more fluidly would be to adapt the path planner used, however, this still would not solve the other two problems. Instead, we suggest to select goals (location where the robot has to go to) farther away, which are selected based on the belief. The HB-CR-POMCP takes as goal the point with the Highest Belief, and the robot's path planning algorithm is used to reach it. The goal is updated every t_{update} s (or after a number of iterations for the simulations), or when the person is visible. A search limit of $d_{\text{max\_search}}$ cells is given to prevent the robot from choosing a too far goal.

In order to find the highest belief, a probability matrix for the map has to be calculated. Since the CR-POMCP belief is a list of continuous states, first a 2D histogram is made of the position of the person (s_{person}). It contains the number of particles per cell, which are then divided by $n_{\text{particles}}$, to get the probability of the person being there, such as can be seen in Figure 4.1-*right*. From the matrix, the cell with the highest probability is used as goal. Choosing the right dimension of the matrix is important, because a large histogram matrix results in a high number of cells, which requires a higher number of belief points (n_{belief}), and therefore, also requires more simulations (n_{sim}). A too small belief matrix results in a loss of precision for goals, which mainly

is important when the person is visible. The size we chose for the belief maps depends on the size of the map, and the precision we require. For the smaller environments, such as at the FME, we opted for the same resolution as the grid map, but for larger maps, such as Telecos Square, we reduced the resolution five times from 75 × 69 cells to 15 × 14 cells.

5.6.1 Tracker when Visible

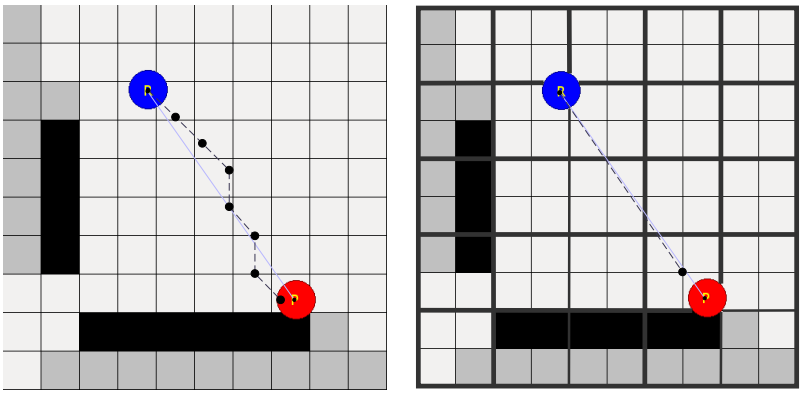
The CR-POMCP algorithm and HB-CR-POMCP work very well, but have some limitations which are mainly present when the person is visible. Figure 5.6 visualizes these limitations by comparing the planned path to the straight line from the robot to the person. The CR-POMCP algorithm has only eight directions to which it can go to, and therefore, when the person is visible, the actions taken by the planner are most of the time not completely direct and cause a zigzag movement, as can be seen in Figure 5.6(a). The HB-CR-POMCP method gives imprecise goals when the resolution of the histogram matrix is low; Figure 5.6(b) shows that the goal for the robot is set in the centre of the belief matrix' cell in which the person is located. The belief matrix is shown with bold lines, and in this example, has a lower resolution than the normal grid.

Assuming that the observation of the person's position is detailed enough, we use this as goal when the person is visible. At the same time, the belief of the POMCP model (used by both CR-POMCP and HB-CR-POMCP) is updated, and when the person is not visible the HB-CR-POMCP is used again.

Note that we also could have used a PF or KF to do tracking, but these need several observations in order to reach to a good estimate, furthermore, we do not need an exact location of the person since we only want to approach it.

5.7 HB-PF Searcher & Tracker

In Section 5.5, a RL method to search-and-track has been presented. Next, this method has been adapted in Section 5.6 to use only the belief. In this section, we present an adapted Particle Filter (PF) that can generate a belief, i.e., probability of the location of the person we are looking for.



(a) The CR-POMCP's zigzag issue. (b) The HB's low resolution issue.

Figure 5.6: The two issues with the explained methods. (a) shows the movement (shown as a dashed line from the robot [left top circle] to the person) of the robot using CR-POMCP, which uses (1 cell length) steps with eight different directions. (b) HB-CR-POMCP moves directly to the highest belief, which is the centre of a belief matrix cell (shown with thick lines). The light blue line shows the direct movement from the robot to the person.

A KF can be used to track a person, but the method lacks two main requirements for searching. Firstly, the person should be visible in the beginning, and secondly, when the person is not visible anymore, we do not have an observation, and therefore, are not able to do the *innovation step* to improve the certainty about the person's location. For longer periods, the prediction will get invalid, since the person is not always going straight. We have chosen to use PFs, since they have been proven to work well for tracking, are able to represent different types of distributions, are easy to adapt to our problem and have a low computational complexity.

5.7.1 Modifications of the Basic PF for Search-and-track

To make the PF suitable for searching, we have made two important changes. First, the initial distribution is based on the initial observation, taking into account the locations where the person could be hidden. Second, we also use the lack of observation to update the particles.

The initialization of the $n_{\text{particles}}$ particles is done by sampling, based on the observation $o = (o_{\text{agent}}, o_{\text{person}})$, and the visibility probability like the initialization of the CR-POMCP in Algorithm 5.10.

In the prediction step, we assume the person has moved around 1 cell (with Gaussian noise) in a random direction (θ):

$$s_t = s_{t-1} + \mathcal{N}(1.0, \sigma_{\text{person}}) \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix}, \theta \sim \mathcal{U}(0,1) \tag{5.9}$$

Algorithm 5.12 Changed update step that also takes into account cases where there is no observation, using $w(s,o)$, see Eq. 5.10.

function UPDATE(o,S,$n_{\text{particles}}$)
  $\forall_{s \in S} : s_w = w(s,o)$
  $\forall_{s \in S} : s_w = s_w / \sum_{k \in S} k_w$ ▷ normalize
  $\bar{S} = \emptyset$
  for $i = 1$ to $n_{\text{particles}}$ **do**
    sample $\bar{s}$ from S with probability $\bar{s}_w$
    $\bar{S} = \bar{S} \cup \{\bar{s}\}$
  end for
  return $\bar{S}$
end function

For the update step (Algorithm 5.12), we also calculate the weight, even if no observation is available of the person:

$$w(s,o) = \begin{cases} 0, & \text{if } \neg \text{ISVALID}(s) \\ e^{-\|o_{\text{person}} - s\|^2 / \sigma_w^2}, & \text{if ISVALID}(s) \wedge o_{\text{person}} \neq \emptyset \\ w_{\text{cons}}, & \text{if ISVALID}(s) \wedge o_{\text{person}} = \emptyset \wedge P_{\text{vis}}(o,s) = 0 \\ w_{\text{inc}}(1 - P_{\text{vis}}(o,s)), & \text{otherwise} \end{cases} \tag{5.10}$$

where, ISVALID checks if the state is a valid free position in the map, and $\emptyset$ refers to the *hidden* value of the observation. If the person is visible, a probability is calculated based on the distance between the observed location and the particle location, where for a higher σ_w, higher distances are accepted; we set it to 1.0. If the person is not visible to the agent, and neither would the particle s be visible to the agent, then it is given a constant weight w_{cons} (0.01). Finally, if the person is not observed, but there is a probability of seeing the particle $P_{\text{vis}}(o,s)$ (Eq. 5.7 defined in subsection 5.4.1) then a lower weight $w_{\text{inc}} << w_{\text{cons}}$ is used (we set it to 0.001), since the observation is inconsistent with the expected value.

5.7.2 Person Location Estimation (HB Calculation)

The PF method gives us a probability map of the person's location. Like discussed in Section 5.6, we decided to use a grid to find the HB. In order to decide a goal for the agent, we can use the average position of the particles. This makes sense when only tracking is done, since the particles will be close to the target and normally distributed. Here however, this will not be the case when the agent has to search the person, because the real person location might not (yet) be at the centre of the particles.

The HB is calculated as in the previous method: the number of particles in a grid cell divided by the total gives the probability of the person being in that cell. To prevent the agent from changing too quickly from one point to another, the goal is maintained during several time steps t_{update}. In larger maps, the belief grid cells should not be too small in order to accumulate enough particles. Also a maximum search distance was set in order to have the robot not go too far; only in the case of not finding any HB in this area, the search space was increased to the whole map. Like in subsection 5.6.1, the observed location by the robot is used when the person is visible, such that the precision is higher and we do not depend on the HB grid.

5.7.3 Extension of the method to handle Dynamic obstacles

In previous subsections, we predicted the possible locations of the person based on the known map of the environment. For the method to work with dynamic obstacles, we have added false negatives to the prediction phase. Here, however, we also take into account dynamic obstacles that can occlude the person in the ray tracing function.

In the case of not taking into account dynamic obstacles, the system assumes, after not detecting anything, that the area in its surrounding is free. When dynamic obstacles are taken into account, particles behind them are given a higher weight w_{cons}, see Eq. 5.10; later on in subsection 5.8.3 some examples will be shown.

Handling false positive detections is more difficult, but they can be partly treated by the PF method. Several iterations are needed in which the incorrect observation is detected in order to concentrate all the particles to that location.

5.8 Simulations

In this section several simulations on several maps are discussed with the different presented methods.

Movement: Although we use grid maps for the obstacle locations, coordinates of the agents are continuous, and for each iteration the agents do a step of 1 cell distance (also in diagonal, thus not $\sqrt{2}$). The algorithms calculate a goal location for the agent, which can either be one step—in the case of the CR-POMCP—or a goal farther away. In the latter case, the agent is moved one step at a time in the goal's direction following the shortest path. The simulations include neither acceleration, nor friction, nor collision, for simplicity. Furthermore, the agents are not allowed to be neither outside the map nor on top of an obstacle.

Person: The tracked person is simulated by giving him/her a goal to go to, starting at a random position with a random goal. Each iteration the goal is approached one step, and when the goal is reached, a new random goal is chosen.

Dynamic obstacles: To generate a noisy environment, a large group (up to 100) of "people" have been simulated, which each use the same movement strategy as the tracked person (i.e. choosing the random goals). These moving people generate a noisy vision, since they block the visibility of the agent—without the agent knowing about the other people being there. For simplicity, the simulated dynamic obstacles are not avoided in the simulations, in the real-world experiments they are avoided using obstacle detection.

Followers: In the following simulations, two simple following methods were used in order to compare our methods: first Simple Follower in the last simulations See All Follower.

The Simple Follower simply goes to the position where the person was detected last [Garrell et al., 2013]. When the person is not visible any more the robot keeps going to the position where the person was last visible. When the person has not yet been

detected, it simply stays at the same position. The See All Follower can always see the person—independent of the distance or occlusion by obstacles—and therefore, is an upper bound method.

Simulation Runs: For all the simulations, a large amount of runs were done for all of the methods and experiment parameters (such as the map and number of dynamic obstacles) with random positions for the agent and person, always taking into account whether they should be visible or not. To make the comparison as fair as possible, all the algorithms and variation of parameters to compare were run for the same simulation (i.e. movement of the person and other persons (if present), and starting position of the agent). After all algorithms were run, another simulation was generated which then again was run with all algorithms. This is especially important because two random simulations can be very different in difficulty, in some simulations the person can be much closer to the agent than in others.

Belief Error: A measurement of the belief error ε_b has been introduced which indicates the error of the person's location in the belief with respect to the real location p. The value ε_b is a weighted distance between the person's location in the belief and in reality:

$$\varepsilon_b = \sum_{h \in H} b_h \|h - p\| \tag{5.11}$$

where H is the list of HB points, and b_h is the belief of cell h. An average of this value has been used to compare the beliefs; since we do not have ground truth in the real experiments, this value can only be calculated in simulation.

Below, we will discuss the different simulations done with the different algorithms. The last subsection details the latest simulations comparing the HB-PF Searcher & Tracker and HB-CR-POMCP Searcher & Tracker presented in [Goldhoorn et al., 2017b].

5.8.1 POMCP Hide-and-seek

First, we tested the HB-CR-POMCP Searcher & Tracker in the hide-and-seek game against the two automated hiders: the Smart Hider and the Random Hider. The Smart Hider minimized the distance to the base, but at the same time kept a minimum distance

to the seeker, as explained in Section 4.9. The Smart Hider used the same reward function Eq. 4.36, but with $D_{\max} = \max(rows, cols)$, $v = 0.5$, and $R_{\text{hider}}(s, h, b) = 0.2d_{hb}$ if $d_{sh} = 0$. The last case was added to reduce the probability of the Smart Hider choosing to go to the base (or be close to it) when the seeker was actually there.

Both algorithms were tested in discrete state space without noise and in continuous state space with noise. The games were played in turns, where in each turn both players had to do an action before the next turn started. The game ended in a tie if the maximum number of actions was reached, which was set to rows × cols. The used maps were BRL (50×10 cells) and FME (20×12 cells), see Section 3.1, with the base on the centre of the map.

Both the simple reward (Eq. 4.17) and the triangle reward (Eq. 4.18) were tried. In the first case, the reward aggregation was done with an average (Algorithm 5.7, line 26), and in the second case with Eq. 5.8. Furthermore, two different rollout policies were used: uniformly random action selection and the heuristic Smart Seeker (Section 4.8).

5.8.1.1 Results

No significant difference in the use of the random rollout policy and the Smart Seeker rollout policy has been found, therefore only the Smart Seeker policy has been used.

The Random Hider was the easiest opponent and in most games the seeker won against it, except for the POMCP with triangle reward ($< 50\%$ won games) and the Smart Seeker ($< 70\%$ won games); therefore, we concentrate on the Smart Hider.

The results of the games played against the Smart Hider are shown in Table 5.2, and demonstrate us that the POMCP planner with the simple reward worked better than using the triangle reward ($p < 0.01$; Fisher's exact test, two-sided, this has been used to check all the win statistics). The results in continuous space were better than in discrete space, which was probably because the Smart Hider was not tuned for continuous space. In the discrete cases, almost all games that used the triangle rule and the Smart Seeker ended in a tie, which shows that the games reached an equilibrium. The POMCP method worked significantly better ($p < 0.01$) than the Smart Seeker, except for case of continuous space in the smaller map (FME) where no significant difference was found.

Table 5.2: The results of simulations against the Smart Hider on the BRL (a) and FME (b). All lines are statistics of about 90 runs. *Num. Act.* are the average ± standard deviation number of actions for the won games. *Avg. Time* is the average step time for actions in won games (or tie if there are none).

(a) BRL (50 × 10 cells)

Seeker	Cont.	Reward	Win	Tie	Lose	Num. Act.	Avg. Time [s]
POMCP		simple	31.8%	68.2%	0%	102.6	0.54
POMCP		triangle	0%	100%	0%	–	0.59
Smart Seeker	–		0%	100%	0%	–	0.04
POMCP	✓	simple	70.9%	29.1%	0%	67.9	0.94
POMCP	✓	triangle	50.6%	47.1%	2.4%	22.8	0.81
Smart Seeker	✓		49.4%	50.6%	0%	17.9	0.04

(b) FME (20 × 12 cells)

Seeker	Cont.	Reward	Win	Tie	Lose	Num. Act.	Avg. Time [s]
POMCP		simple	31%	66.7%	2.3%	105.6	0.35
POMCP		triangle	2.3%	97.7%	0%	84.5	0.23
Smart Seeker	–		0%	100%	0%	–	0.04
POMCP	✓	simple	100%	0%	0%	8.2	0.53
POMCP	✓	triangle	71%	28%	1.1%	47.3	0.23
Smart Seeker	✓		96.4%	0%	3.6%	11.0	0.04

The step times of the Smart Seeker were about 0.04 s, and for the POMCP methods 0.9 s at maximum, but the times of the POMCP solver depend especially on the number of simulations (n_{sim}).

5.8.2 CR-POMCP Variants

In this section we test the CR-POMCP algorithm and its variants discussed in Sections 5.5 and 5.6:

- CR-POMCP: the RL method for which we learn a policy on-line, and then this is used to find the best action to do for the current belief.
- HB-CR-POMCP: chooses the HB location for the robot to go to, and uses the shortest path to reach it.
- CR-POMCP Searcher & Tracker: uses the action of the CR-POMCP when the person is not visible, otherwise it goes directly to the person.

- HB-CR-POMCP Searcher & Tracker: uses the HB-CR-POMCP when the person is not visible, otherwise goes directly to the person.
- Simple Follower: goes to the last location where the person was seen.

Comparison of the algorithms was done by looking at the average distance to the person, the time it took the robot to get close to him/her and the belief error (Eq. 5.11).

In these simulations the dynamic obstacles were not taken into account while planning, and no visibility probability was used, but an infinite visibility was assumed. A noisy crowded environment was simulated by adding a group of 10 or 50 people to the scene, to reduce the robot's visibility. Finally, we will show the results of simulations to which Guassian noise was added to the observations.

A total of about 4000 experiments were done, repeating each of the conditions at least 38 times. For each experiment 200 iterations were done, except for the smaller map FME, for which 100 steps were done.

5.8.2.1 Algorithm Parameter Values

The values of the parameters used are shown in Table 5.3. The value of the discount factor γ was set to 0.95, like in [Silver and Veness, 2010]. The exploration constant was set to $c = r_{\text{hi}} - r_{\text{low}}$ as suggested in [Silver and Veness, 2010], where r_{hi} and r_{lo} are respectively the highest and lowest reward values returned. Our reward is $r = -d_{\text{ap}}$ and therefore the highest value would be 0, and the lowest value would be the negative longest path: $-(rows \times cols)$. The expand count (e_{count}) was set to 2, to prevent the tree from being expanded at the first simulation, but on the other hand, trying to store information of as many simulations as possible. The maximum depth (d_{max}) was set to an estimation of when the robot should find the person. Although it also depends on the number of obstacles, using the size of the map is a good indicator.

In simulation, the smaller values of n_{sim} and n_{belief} were used for the small map only. These values were obtained experimentally: a higher n_{sim} and n_{belief} should improve the policy and belief, but at the same time require more calculation time. Especially when we want a real-time system, the timing is important. The other parameters tune the

noise of the states and observations returned by the simulator $\mathcal{G}$ (see subsection 5.5.1), these were tuned first in simulation, and later while doing tests with the real robot.

The HB-CR-POMCP method updated its belief every 3 s in the real experiments and every 3 iterations in the simulations. We did not limit the distance for the search of the highest belief ($d_{\max}$). The 2D histogram for the belief made by HB-CR-POMCP had the same scale for the FME map (17 × 12 cells), but for the BRL map it was reduced to 1 : 2 (from 80 × 15 cells to 40 × 8 cells) and for the Telecos Square it was scaled 1 : 5, because it is a large map (75 × 69 cells to 15 × 14 cells).

Table 5.3: The parameters values used. There is a column for the parameter values in simulation, and in the real experiments.

Parameter	Real	Sim.	Description
		CR-POMCP Parameters	
γ	0.95	0.95	discount factor
n_{sim}	1000	2500	number of simulations
n_{belief}	500	1000	number of belief points
c	$rows \times cols$		exploitation constant
e_{count}	2	2	expand count
$d_{\max}$	$2(rows + cols)$		maximum search depth
		Noise Parameters	
σ_{agent}	0.2 m	0.2 m	standard dev. of Gaussian noise for robot movement
σ_{person}	0.3 m	0.3 m	std. dev. of Gaussian noise for person movement
$\sigma_{\text{obs,agent}}$	0.1 m	0.1 m	std. dev. of Gaussian noise for robot observation
$\sigma_{\text{obs,person}}$	0.1 m	0.1 m	std. dev. of Gaussian noise for person observation
$p_{\text{false\_neg}}$	0.3	0.3	false negative detection probability
$p_{\text{false\_pos}}$	0	0.001	false positive detection probability
$p_{\text{see\_occ}}$	0.01	0.01	probability of seeing through an obstacle
		Highest Belief	
t_{update}	3 s	3 steps	wait time to re-calculate goal

5.8.2.2 Results

Table 5.4 shows the results of the simulations done in all maps, vertically it shows the three different maps, and the amount of people (dynamic obstacles) randomly walking in the area. On each line the average of at least 38 simulations is shown for the specific method. The first results column shows the average (± standard error) distance between the robot and the person it should search and track; the second column gives the average (and standard error) belief error Eq. 5.11, except for the Simple Follower, since it does not use a belief; the last columns show the percentage of the time in which the robot was closer than 1 cell or 5 cells to the person.

Table 5.4: The results of the simulation of the different CR-POMCP methods. On the left side the map, number of people occluding, and the method are indicated. The result columns: 1) *Avg. Distance*: indicates an average (± standard error) of the distance between the robot and the person; 2) ε_b: the *Belief Error* error (Eq. 5.11); 3) *CR* 1 and 5: Close Range, percentage of time that the robot was respectively at a distance of 1 or 5 grid cells, or closer.

		Method	Avg. Distance [m]	ε_b [m]	CR1 [%]	CR5 [%]
FME	0 People	Simple Follower	1.66 (±0.35)	–	85.08	95.19
		CR-POMCP	1.55 (±0.39)	0.51 (±0.13)	15.60	95.21
		HB-CR-POMCP	1.52 (±0.32)	0.47 (±0.17)	76.16	95.48
		CR-POMCP S.&T.	1.49 (±0.32)	0.48 (±0.19)	85.08	95.19
		HB-CR-POMCP S.&T.	**1.47 (±0.25)**	0.48 (±0.24)	85.53	95.50
	10 People	Simple Follower	3.11 (±0.47)	–	38.88	82.73
		CR-POMCP	3.26 (±0.36)	1.61 (±0.86)	11.20	84.55
		HB-CR-POMCP	2.64 (±0.39)	1.43 (±0.79)	36.84	87.78
		CR-POMCP S.&T.	2.79 (±0.39)	1.55 (±0.92)	41.82	85.05
		HB-CR-POMCP S.&T.	**2.54 (±0.36)**	1.33 (±0.81)	46.82	88.04
	50 People	Simple Follower	5.94 (±0.55)	–	7.03	48.59
		CR-POMCP	5.31 (±0.42)	5.39 (±0.51)	4.85	55.52
		HB-CR-POMCP	5.34 (±0.45)	5.42 (±0.62)	5.61	54.09
		CR-POMCP S.&T.	**5.14 (±0.44)**	5.01 (±0.67)	7.93	57.58
		HB-CR-POMCP S.&T.	5.45 (±0.46)	5.46 (±0.58)	7.39	52.77
BRL	0 People	Simple Follower	6.01 (±1.65)	–	69.04	80.79
		CR-POMCP	5.74 (±1.31)	1.81 (±0.65)	4.30	81.62
		HB-CR-POMCP	4.62 (±1.25)	1.24 (±0.58)	16.46	87.49
		CR-POMCP S.&T.	4.08 (±1.45)	1.34 (±0.61)	77.80	87.23
		HB-CR-POMCP S.&T.	**3.78 (±1.32)**	1.20 (±0.56)	78.99	88.54
	10 People	Simple Follower	7.25 (±1.68)	–	46.28	75.85
		CR-POMCP	6.81 (±1.58)	3.05 (±2.44)	4.45	76.65
		HB-CR-POMCP	6.50 (±1.67)	3.10 (±2.51)	14.80	77.38
		CR-POMCP S.&T.	5.83 (±1.60)	2.97 (±2.04)	54.45	78.53
		HB-CR-POMCP S.&T.	**5.51 (±1.54)**	2.51 (±2.11)	56.01	80.46
	50 People	Simple Follower	14.57 (±2.29)	–	12.17	39.69
		CR-POMCP	12.40 (±1.94)	9.94 (±2.09)	2.48	43.32
		HB-CR-POMCP	12.57 (±2.09)	10.39 (±2.14)	5.89	47.29
		CR-POMCP S.&T.	**11.89 (±2.02)**	9.94 (±1.99)	16.16	47.92
		HB-CR-POMCP S.&T.	12.47 (±2.04)	10.21 (±2.01)	13.71	45.00
Telecos Square	0 People	Simple Follower	14.44 (±3.14)	–	50.69	61.35
		CR-POMCP	7.79 (±2.08)	4.62 (±2.01)	3.86	77.07
		HB-CR-POMCP	10.14 (±1.98)	5.17 (±1.95)	2.56	50.40
		CR-POMCP S.&T.	6.59 (±2.02)	4.71 (±2.04)	67.72	79.56
		HB-CR-POMCP S.&T.	**5.61 (±1.95)**	3.73 (±1.87)	71.07	81.01
	10 People	Simple Follower	18.05 (±3.24)	–	29.80	49.71
		CR-POMCP	10.30 (±2.84)	6.99 (±2.27)	3.80	68.04
		HB-CR-POMCP	12.59 (±2.38)	8.18 (±2.31)	2.12	45.65
		CR-POMCP S.&T.	**8.59 (±2.41)**	6.19 (±2.18)	45.57	72.81
		HB-CR-POMCP S.&T.	8.79 (±2.34)	6.35 (±2.24)	48.80	67.22
	50 People	Simple Follower	23.76 (±2.91)	–	10.87	26.39
		CR-POMCP	15.99 (±2.79)	13.99 (±3.12)	2.69	43.81
		HB-CR-POMCP	19.14 (±2.78)	16.86 (±3.19)	1.38	28.36
		CR-POMCP S.&T.	**15.01 (±2.74)**	13.41 (±2.98)	16.47	46.29
		HB-CR-POMCP S.&T.	16.59 (±2.84)	14.42 (±3.01)	18.08	36.82

Statistical comparisons between the average robot-person distance of the methods were done with the Wilcoxon ranksum test (2-sided), because not all the data had a normal distribution according to the χ^2-test. When looking at the results we should take into account the size of the maps, in a small map—such as FME—it is easier and faster to find the person, but it easily gets crowded and therefore, we should focus more on the results on the other two maps.

The method which was significantly closest to the person, without any other people walking around and in all maps, was the HB-CR-POMCP Searcher & Tracker ($p < 0.001$); the second best was the CR-POMCP Searcher & Tracker ($p < 0.001$). When there were 10 people walking around the results slightly change, but the HB-CR-POMCP Searcher & Tracker still was significantly best in the FME and BRL Lab ($p < 0.001$). In the Telecos map there was no significant difference with the CR-POMCP Searcher & Tracker. Finally, for 50 people the CR-POMCP Searcher & Tracker was best ($p < 0.001$), except for the FME map where there was no significant difference with the HB-CR-POMCP Searcher & Tracker. In all cases the Simple Follower was worst ($p < 0.001$), except for the FME map when there were 10 or more people, then there was no significant difference with the CR-POMCP.

The high distances for large maps with many people walking around indicate that, on average, the robot was relatively far from the person. However, the close range percentage of at minimum 25% indicates that the robot was close to the person (5 cells or less) during at least 25% of the time. The remaining time, the robot was farther away, because it had to find the person first in the crowded environment.

Although no statistical tests have been done for the belief error and *Close Range*, their results are mostly consistent with the *Average Distance*. The lowest average distance to the person showed the lowest agent belief error and the highest percentage in the close range (5 cells). The only exception is the FME map with 0 people, but here the results do almost not differ.

A last measurement is the relative time in which the person was visible to the robot. Also here we see that the HB-CR-POMCP Searcher & Tracker scored best, i.e. had the highest percentage, and the *Simple Follower* was worst. The visibility percentages in the BRL map were respectively 95% and 86%, and for the Telecos map 89% and 67%.

Table 5.5: The results of the simulation of the different methods considering sensor noise ($\sigma = 1$m). On the left side the map, and the method are indicated. The result columns indicate: 1) *Avg. Distance*: an average (± standard error) of the distance between the robot and the person; 2) ε_b: *Belief Error* (Eq. 5.11); 3) *CR* 1 and 5: Close Range, percentage of time that the robot was respectively at a distance of 1 or 5 grid cells, or closer.

	Method	Avg. Distance [m]	ε_b [m]	CR1 [%]	CR5 [%]
FME	Simple Follower	3.41 (±0.72)	–	11.46	89.94
	CR-POMCP	3.08 (±0.30)	1.61 (±0.47)	10.36	91.60
	HB-CR-POMCP	2.91 (±0.51)	1.59 (±0.31)	11.27	92.53
	CR-POMCP S.&T.	2.87 (±0.57)	1.57 (±0.49)	11.90	92.71
	HB-CR-POMCP S.&T.	2.85 (±0.48)	1.56 (±0.34)	12.37	93.25
BRL	Simple Follower	13.76 (±3.88)	–	2.60	36.95
	CR-POMCP	6.77 (±3.72)	2.52 (±2.52)	2.94	67.96
	HB-CR-POMCP	7.33 (±2.28)	2.35 (±1.23)	3.62	55.35
	CR-POMCP S.&T.	8.10 (±2.46)	2.57 (±1.44)	3.42	45.03
	HB-CR-POMCP S.&T.	8.25 (±2.37)	2.53 (±1.34)	3.64	45.50
Tel. Sq.	Simple Follower	21.37 (±2.21)	–	1.64	28.61
	CR-POMCP	9.81 (±2.11)	6.22 (±2.06)	2.42	56.44
	HB-CR-POMCP	12.47 (±2.26)	7.77 (±2.19)	1.75	31.67
	CR-POMCP S.&T.	10.56 (±2.04)	6.37 (±1.98)	2.02	48.18
	HB-CR-POMCP S.&T.	11.50 (±2.18)	7.16 (±2.11)	2.41	38.86

The better working *Tracker* versions can be explained by the use of the Simple Follower as soon as the person was visible, because the movement was much more efficient, as discussed in subsection 5.6.1. When the number of people increases, there was a higher probability of the tracked person to be occluded by them. These dynamic obstacles, the other people, were not known by the POMDP simulator, used in the POMCP, and therefore the model could incorrectly assume that the person was not visible. This resulted in an incorrect belief, and caused the HB-CR-POMCP Searcher & Tracker to be worse, because it used the belief directly for its search strategy.

5.8.2.3 Observation Noise

Additionally, we analysed the effect of sensor noise in the proposed method. In Table 5.5, the results of simulations with a Gaussian sensor noise of $\sigma = 1$ m in all maps are shown, and here no dynamic obstacles were present (i.e. 0 people). As a comparison the no noise case ($\sigma = 0$ m) can be found in Table 5.4, in the rows with *0 people*.

In the first map (FME), the significantly best method—i.e. having the lowest average distance to the person—was the HB-CR-POMCP Searcher & Tracker ($p < 0.001$), and the second best was the CR-POMCP Searcher & Tracker ($p < 0.001$). In contrast,

in the BRL Lab the significantly best method was the CR-POMCP ($p < 0.001$), followed by the HB-CR-POMCP ($p < 0.001$). Finally, in the Telecos Square environment, the significantly best method was the CR-POMCP ($p < 0.001$), and the second best was the CR-POMCP Searcher & Tracker ($p < 0.001$). The better working CR-POMCP in large maps can be because the steps taken by this method are from the policy which is learned by doing a large number of simulations. The *Simple Follower*, however, also used by the *Tracker* methods, goes directly to the noisy observed location, if the person is seen. Finally, the belief error and *Close Range* are largely consistent with the results of the *Average Distance*.

5.8.3 HB-CR-POMCP Searcher & Tracker and HB-PF Searcher & Tracker

Finally, we evaluate the PF method HB-PF Searcher & Tracker and compare it to the method found to be best so far: the HB-CR-POMCP Searcher & Tracker. Furthermore, the methods were tested both with and without the use of dynamic obstacles, and here we do take into account a probabilistic visibility with maximum (Eq. 5.7). As a reference method, we used the See All Follower which always sees the person.

A noisy crowded environment was simulated by adding a group of 10 and 100 dynamic obstacles to the scene. A total of more than 8000 experiments were done, repeating each of the conditions at least 140 times.

The experiments were separated in *search* and *track*. In the first, the person started hidden from the agent and stayed there, and the simulation stopped when the agent found the person and was close to it. Simulations were stopped if the maximum time passed: 500 steps for the FME map and 5000 for the Tel. Square map. Here we measured the time (steps) the agent needed to see the person. In the track experiments the agent started seeing the person, doing 500 steps for the FME map and 1000 for the Tel. Square map. In the *track* simulations we measured the distance to the person, the time of visibility, and the recovery time, which is the time the robot needs to recover the person's position after not having seen him/her.

5.8.3.1 Algorithm Parameter Values

The values of the parameters were obtained experimentally; the parameter values are shown in Table 5.3, and the new parameters are shown in Table 5.6. The algorithms updated the particles or belief every iteration, and the highest belief points were calculated every t_{update}=3 s in the real experiments and every 3 iterations in the simulations, if the person was not visible. For the HB-CR-POMCP Searcher & Tracker a tree search depth ($d_{\max}$) of 1 was used to only do simulations with a depth of one step, because we did not use the policy, only the belief.

Table 5.6: The parameters values used during the real experiments and simulations.

Parameter	FME	Tel.Sq.	Description
		CR-POMCP Parameters	
n_{belief}	1000	5000	number of belief points
$d_{\max}$	1	1	maximum search depth
		Particle Filter Parameters	
$n_{\text{particles}}$	1000	5000	number of particles
w_{cons}	0.01	0.01	PF weight if no obs., but consistent (Eq. 5.10)
w_{inc}	0.001	0.001	PF weight if no obs., but inconsistent (Eq. 5.10)
σ_w	1.0 m	1.0 m	normalization of PF weight (Eq. 5.10)
		Highest Belief	
cells size	1 m×1 m	3.8 m×3.8 m	cell size for HB grid
$d_{\text{max\_search}}$	10 m	25 m	maximum distance to search HB
		Visibility Parameters	
$p_{\text{v,max}}$	0.85	0.85	maximum probability visibility Eq. 5.7
α_{vis}	0.17	0.17	reduction factor Eq. 5.7
$d_{\text{v,max}}$	3.0 m	3.0 m	maximum distance full visibility Eq. 5.7

5.8.3.2 Results

The results of the simulations, the 140 repetitions, are shown in Tables 5.7 and 5.8 for the FME and Tel. Square map respectively. We did search and track simulations separately, the measurements in time are discrete time steps, and for the distance, the cells are converted to meters. The cell size for the FME was 1.0 m and for the Tel. Square 0.8 m, and the average speed was 1 cell per time step. Although the See All Follower was always given the location of the person, we do use the real visibility (i.e. taking into account distance and obstacles) when we calculate the measurements (*First visible step*, *Visibility*, and *Recovery time*), which gives their best possible values.

In the search phase, the *first visible step* (the discrete time until the person was found) was measured, for which only a significant difference was found with the See

All Follower. The high standard deviation is due to the large difference in the starting position of the robot and person for the simulations. The distance to the person during the *tracking* phase was found to be significantly less for the HB-PF Searcher & Tracker ($p < 0.001$; Wilcoxon Ranksum test) in comparison with the HB-CR-POMCP Searcher & Tracker, except for in the FME map without people walking around. When looking at the belief error (Eq. 5.11), there is no clear difference between the methods.

The visibility was found to be significantly higher ($p < 0.001$; Fisher's exact test) for the HB-PF Searcher & Tracker, except for the case of 100 random people walking around. Finally, for the *tracking* phase, the *recovery time* shows the average time the agent needed recover the visibility of the person after having lost him/her. It shows that after the See All Follower, the HB-PF Searcher & Tracker worked best for both maps ($p < 0.01$; Wilcoxon Ranksum test) without dynamic obstacles. With dynamic obstacles the See All Follower had better results, but there is no significant difference between the other methods. For the HB-PF Searcher & Tracker algorithm, the use of the detected dynamic obstacles in the algorithm had a positive effect on the *distance to the person* when tracking, on the bigger map ($p < 0.01$; Wilcoxon Ranksum test). Also for the *visibility* this had a positive influence.

The run time of the algorithms is not significantly different, for search the average was about 250 ms/iteration, and for track about 216 ms/step. Note that the run time mainly depends on the number of particles for the HB-PF Searcher & Tracker or on the number of belief points for the HB-CR-POMCP Searcher & Tracker.

5.8.3.3 Dynamic obstacles

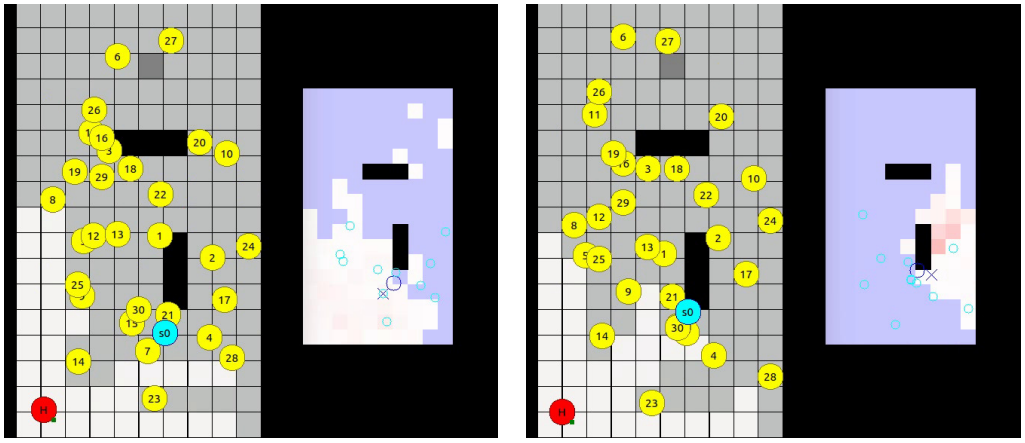
Using the dynamic obstacles, which have been detected to update the probability map, has a great advantage in certain situations, as shown in Figure 5.7. Left can be seen the situation where there are particles maintained behind the detected dynamic obstacles, whereas in Figure 5.7(b) that area is already cleaned. In this simulation the first method needed 21 steps to find the person, whereas the latter needed 366 steps, because it went around the obstacle, thinking that the person could only be hidden behind obstacles. See a demonstration video on http://alex.goldhoorn.net/thesis/dynobst/.

Table 5.7: This table shows the average and standard deviations for the measurements of the simulations in the FME. The different algorithms are shown in columns, where *(d)* indicates the methods that take into account dynamic obstacles. The type of measurement is indicated in the first column, including the used unit and the type of task (search or track). The *First visible step* is the first time (discrete time) when the person was visible to the agent. The *visibility* indicates the amount of time (% of the whole simulation) in which the person was visible to the agent. The *distance to the person* is the average distance to the person, and the *belief error*, Eq. 5.11, indicates how well the belief represents the real location of the person. The *recovery time* shows the average time the agent needed to return to see the person after having lost him/her. [1]The *recovery time* is identical to the *recovery distance* since the agent moves 1 m per time step on average.

Type of Measurement	**Num. Pers.**	*See All Follower*	*HB-PF Searcher & Tracker*(d)	*HB-PF Searcher & Tracker*	*HB-CR-POMCP Searcher & Tracker*	*HB-CR-POMCP Searcher & Tracker*
First visible step	0	2.4 ± 3.2	4.5 ± 5.8	4.7 ± 8	5.3 ± 6.9	5 ± 6.5
[time steps]	10	3.1 ± 3.7	7.2 ± 9.1	6.9 ± 10.7	7.7 ± 11.8	7.4 ± 10.1
(search)	100	8.4 ± 6.4	67.4 ± 71.1	90 ± 111.1	65.2 ± 78.8	59.9 ± 67.5
Visibility	0	100%	93.7%	93.4%	94.5%	93.9%
[%]	10	100%	59.3%	56.5%	59.5%	58.4%
(track)	100	100%	2.4%	1.8%	2.8%	2.8%
Distance to pers.	0	1.0 ± 0.4	2 ± 1.4	2 ± 1.4	1.9 ± 1.5	1.9 ± 1.5
[m]	10	1.0 ± 0.4	3.2 ± 2.5	3.4 ± 2.6	3.3 ± 2.8	3.5 ± 2.9
(track)	100	1.0 ± 0.4	5.6 ± 2.9	5.4 ± 2.8	6 ± 3.2	6 ± 3.2
Belief Error	0		4.2 ± 3	4.4 ± 3.3	5.1 ± 3.1	5 ± 3.2
(ϵ_b) [m]	10		5.1 ± 3.2	5.4 ± 3.4	6 ± 3.1	5.7 ± 2.9
(search)	100		7.4 ± 1.9	8 ± 3.4	7.7 ± 1.8	7.5 ± 1.8
Belief Error	0		1.1 ± 1.1	1.1 ± 1.1	0.8 ± 1.3	0.9 ± 1.4
(ϵ_b) [m]	10		2.5 ± 2.5	2.8 ± 2.7	2.5 ± 2.9	2.6 ± 3
(track)	100		6.4 ± 1.9	6.3 ± 2.9	6.6 ± 2	6.7 ± 2
Recovery time	0	1.2 ± 0.5	2.4 ± 1.9	2.5 ± 2.1	2.4 ± 3.2	2.7 ± 3.8
[time steps]	10	2.2 ± 3.8	3.5 ± 4.4	3.9 ± 4.6	3.6 ± 5.2	3.8 ± 5.5
(track)	100	3.2 ± 5.7	48.8 ± 54.5	63 ± 67.2	46.5 ± 55.8	46.7 ± 53.9

Table 5.8: This table shows the same measurements as Table 5.7, but for the Tel. Square map.

Type of Measurement	Num. Pers.	*See All Follower*	*HB-PF Searcher & Tracker*(d)	*HB-PF Searcher & Tracker*	*HB-CR-POMCP Searcher & Tracker*	*HB-CR-POMCP Searcher & Tracker*
First visible step	0	13.0± 16.2	110± 336.2	106.2± 369.6	114.6± 264.3	110.4± 233.2
[time steps]	10	13.8± 21.7	105.6± 285.5	96.5± 273.8	93.7± 189.3	107.4± 237.1
(search)	100	14.4± 17.2	115.6± 264.9	127.7± 265.9	121.2± 300.5	117.5± 246.9
Visibility	0	100%	62.7%	61.6%	58.5%	60.4%
[%]	10	95.6%	51.2%	45.6%	48.8%	49.0%
(track)	100	70.5%	13.8%	12.8%	15.9%	15.7%
Distance to pers.	0.0	0.8 ± 0.4	8.2 ± 9.1	8.6 ± 9.5	8.5 ± 9.0	8.3 ± 9.1
[m]	10.0	0.8 ± 0.4	9.4 ± 9.4	11.0 ± 10.4	9.8 ± 9.6	9.6 ± 9.4
(track)	100.0	0.8 ± 0.4	13.6 ± 9.4	15.4 ± 10.4	13.8 ± 9.7	13.8 ± 9.6
Belief Error	0.0		25.4 ± 11.8	26.1 ± 9.4	23.8 ± 6.9	23.4 ± 6.5
(ϵ_b) [m]	10.0		25.9 ± 10.0	26.5 ± 9.8	23.0 ± 6.9	23.4 ± 7.1
(search)	100.0		25.4 ± 9.3	25.2 ± 9.2	23.2 ± 6.2	23.3 ± 6.9
Belief Error	0.0		7.5 ± 10.1	7.8 ± 10.2	7.7 ± 9.4	7.4 ± 9.6
(ϵ_b) [m]	10.0		9.0 ± 10.2	10.8 ± 11.3	9.2 ± 10.0	9.0 ± 9.9
(track)	100.0		14.9 ± 9.2	16.7 ± 10.4	14.1 ± 9.3	14.2 ± 9.2
Recovery time	0	1.2 ± 0.5	14.9 ± 31.6	15.3 ± 32.5	19.5 ± 34.2	19.1 ± 34.8
[time steps]	10	2.2 ± 3.8	13 ± 27.9	15.6 ± 35.5	15.4 ± 31.6	15.5 ± 32.2
(track)	100	3.2 ± 5.7	22.2 ± 42.7	24.8 ± 48.6	19.7 ± 41.2	20.1 ± 41.9
Recovery dist.	0.0	1.0 ± 0.4	11.9 ± 25.3	12.2 ± 26.0	15.6 ± 27.4	15.3 ± 27.8
[m]	10.0	1.8 ± 3.0	10.4 ± 22.3	12.5 ± 28.4	12.3 ± 25.3	12.4 ± 25.8
(track)	100.0	2.6 ± 4.6	17.8 ± 34.2	19.8 ± 38.9	15.8 ± 33.0	16.1 ± 33.5



(a) Agent using dynamic obstacles. (b) Agent not using dynamic obstacles.

Figure 5.7: (a) The simulated agent using dynamic obstacles, and (b) not using dynamic obstacles. The left image of the image pair shows the person as red circle, the blue circle as the robot, and yellow circles are other people walking around. The black cells are obstacles, light grey are cells visible to the person, and dark grey are not visible cells. The right part shows the probability map, i.e. belief, of where the person could be where red is a high probability, white low, and light blue zero.

5.9 Real-life Experiments

Different real-life experiments were done with our mobile robot Dabo in different urban environments. Volunteers were used as person to track, and others as dynamic obstacles obstructing the robot's vision by passing in front of the robot or by standing in a group of people. The robot did not move quickly for security reasons, and therefore, the tracked person was also asked not to move too fast, furthermore, the minimum distance was set to be 80 cm. To be detected, the person wore a marker, see Figure 5.9, such that the robot was able to recognise him or her.

All the statistics are based on the data measured by the robot, so the distance travelled by the person is more than the values reflected in the table.

5.9.1 POMCP Hide-and-seek

The experiments presented in the previous chapter with the hide-and-seek game, were done in the FME (6 m × 9 m) environment, but the time and space were discrete. However, the experiments reported in this chapter, were done in continuous space,

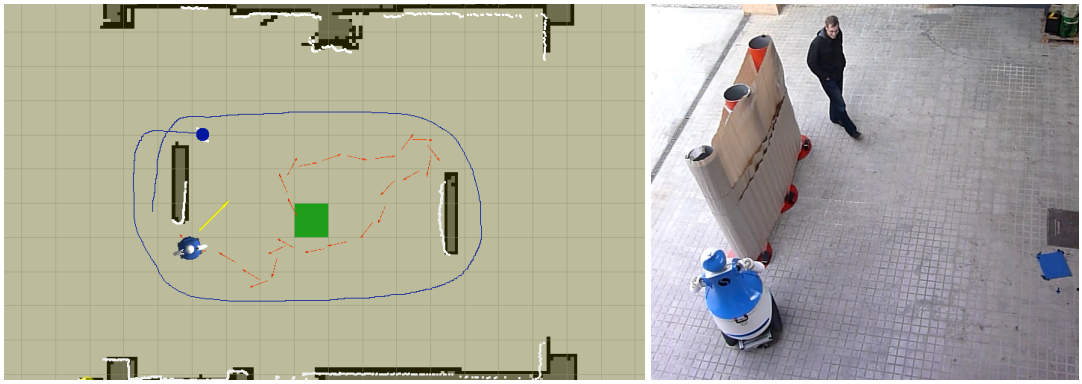


Figure 5.8: Hide-and-seek experiments in the FME with Dabo. The red arrows show the past trajectory of the robot, and the yellow its next goal. The blue line shows the trajectory of the person with the circle being the last location. The photo shows the current situation. The green square at the left and the blue are at the right represent the base.

meaning that both players were allowed to be at any location on the field, not limited to grid cells; moreover, there were no discrete time steps. The robot continuously received updates of the person tracking algorithm, and used this to plan the next action, which caused the robot to move continuously. The methods run at maximum 100 Hz, the Smart Seeker in practice run on average at 31.3 Hz, and the CR-POMCP at 1.2 Hz. The only limitation we set for the person was to not walk too fast, since the robot moved slowly.

Preliminary experiments were done on the BRL map, both with the Smart Seeker and the CR-POMCP, for which the triangle reward was used. The latter showed a protective behaviour, tending to stay near the base. The Smart Seeker did leave the base once the person was visible, trying to catch him.

After the previous issues were solved, more extensive experiments were done on the smaller map, see Figure 5.8. A compilation of some experiments is shown in a video available on http://alex.goldhoorn.net/thesis/hs-pomcp. Here the simple reward was used, since in simulation this gave better results. More than 25 experiments were done with 10 persons (playing as glshider), from which 21 games succeeded and four stopped because of an error on the robot. When we look simply at the win results, then the outcomes can be seen as similar for both seekers. Both methods won one time, and both ended in a tie two times. One tie was due to reaching the maximum time, and

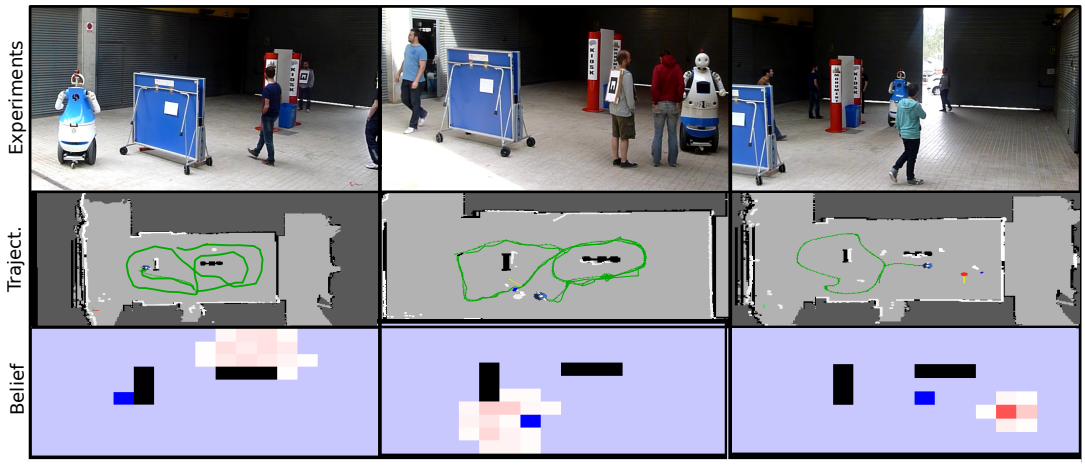


Figure 5.9: Dabo performs real-life experiments at the FME environment. *Top:* Some scenes of the performed experiments. *Center:* The trajectory during the experiments, indicated by the line. *Bottom:* The belief of the person's location (red represents a high probability). Note that the maps are scaled differently.

three times the hider and seeker reached the base at the same time. The other times the hider won, but in most of the cases this was because the person walked too fast. The movement of the robot was relatively slow and the input of the people tracker was sometimes erroneous, which caused the robot to turn often and slow down more. The experiments with the CR-POMCP took slightly more time (103 $\pm$ 85 s) than those with the Smart Seeker (75 $\pm$ 21 s).

5.9.2 HB-CR-POMCP Searcher & Tracker

In this section, we discuss the results of the search-and-track experiments done in the three urban environments (see Section 3.1): FME, BRL and *Telecos Square* (1 and 1a in Table 3.1). We tested search, track, and search-and-track as a whole for the HB-CR-POMCP Searcher & Tracker. The used parameters are shown in Table 5.3, and the visibility in the algorithm is assumed to be infinite (i.e. not probabilistic), nor were the dynamic obstacles taken into account for planning. The videos are available on: http://alex.goldhoorn.net/thesis/st-pomcp/.

In simulation, the algorithms were run in time steps, in which all players executed an action at the same time; for the real-life experiments the algorithm was running continuously, that is, the planning was done real-time, while the robot was moving. First,

an observation was done by the robot (i.e. receiving the robot's and person's position), this was fed to the planning algorithm, and finally, the resulting new robot position was sent to the navigation planner (see Figure 5.4). To prevent the planning algorithm from running too quickly, a minimum time of 400 ms was set between iterations. In practice, the algorithm ran on average 2.1 ± 8.3 Hz in the smallest environment, FME, in the BRL environment at 0.9 ± 1.2 Hz, and in the Telecos Square at 0.9 ± 2.0 Hz.

5.9.2.1 Analysis

Experiments of more than one week were done with Dabo in the different environments (Figures 5.9-5.11). The total trajectory of the robot was around 3 km for all the successfully executed experiments, summing up to more than 3 hours. Some statistics of the experiments are shown in Table 5.9. All these data were obtained from the observations of the robot, and therefore, might include a small measure error. The path length of the person, and the average robot-person distance were calculated when the robot saw the person. The table shows per map and in total, the time of recorded experiments, the path travelled by the robot and the person, their average speed, and the part of time the person was visible to the robot.

Experiments in the FME environment showed that, when using the Simple Follower, the robot did not move most of the time (50.4%; $p < 0.001$, Fisher's exact test), therefore, the robot was not able to follow the target. In contrast, when using the HB-CR-POMCP Searcher & Tracker, the robot moved all the time when the person was not visible, except for 3.7% of the time, and therefore it could follow the target. The CR-POMCP method alone showed promising results in simulation, but in the real-life experiments it moved too slowly to follow the person, for which we extended the method to the HB-CR-POMCP Searcher & Tracker, as explained in Section 5.6.

Extensive experiments with the HB-CR-POMCP Searcher & Tracker were done in the three environments. We started doing experiments in the FME environment, Figure 5.9, because this is a relatively small, and therefore relatively easily controllable. However, the task is still challenging because there were two artificial obstacles present, there were other people walking in front of the robot, and due to sensor noise, or bad light conditions the robot was not able to recognise the marker always. In the three snapshots, shown in Figure 5.9, can be seen that only in the last the robot recognised

Figure 5.10: *Top:* Some scenes of the performed experiments in the BRL. *Center:* The trajectory during the experiments, indicated by the line. *Bottom:* The belief of the person's location (red represents a high probability). Note that the maps are scaled differently.

the marker (indicated by the red dot in the centre map). The second snapshot shows that the person with marker was in front of the robot, but the robot did not see the marker, and nevertheless, the position of the person was stored in the belief, which is shown in the image below. As indicated in Table 5.9, more than two hours of recorded experiments were done covering over 800 m. The lower visibility percentage of the person is because the person, while walking through the environment, frequently got hidden behind an obstacle, whereas in the other environments there are larger open spaces.

The BRL environment, see Figure 5.10, was a good exercise to have long trajectories in which the robot could find, and then follow the person during a longer time. The average robot-person distance (Table 5.9) here was lower, because of the longer experiments in which the robot was able to follow the person continuously. In some cases, while following and temporarily losing the person, the robot did not go in the right direction. In the left snapshot of Figure 5.10 the robot did not see the person anymore, but still kept track of her location through the belief map. The belief was still propagated, but when the robot was at the location of the person (as shown in the snapshot), the person went to the right, but the robot's belief was higher to the left. After going left the robot lost the person, but started searching for her again, and found her.

The last environment, Telecos Square, as shown in Figure 5.11, was probably the

Figure 5.11: *Top:* Some scenes of the performed experiments in the Telecos Square environment. *Center:* The trajectory during the experiments, indicated by the line. *Bottom:* The belief of the person's location (red represents a high probability). Note that the maps are scaled differently.

most complete environment, since it combines indoors, outdoors, columns, a square with trees, etc. More than an hour of experiments were recorded in which the robot run more that 800 m, and the robot was on average 3.8 m from the person. Note, that although this is not close, it is an average of the whole experiments in which the person started far from the robot. We tried several situations in which the robot started inside, and had to find the person outside, and other situations in which the robot followed the person who walked behind a group of people. In this environment there were some small trees (middle snapshot of Figure 5.11), which were modelled as being obstacles (of 0.8 m $\times$ 0.8 m) to avoid the robot going there, even though they were much smaller and therefore only slightly obstructed the vision. When the person was in that area, and not behind a tree, the belief was still propagated behind the trees. Also in this case the robot was able to find the person again.

5.9.3 HB-CR-POMCP Searcher & Tracker and HB-PF Searcher & Tracker

In these experiments the HB-PF Searcher & Tracker has been compared to the HB-CR-POMCP Searcher & Tracker for search and track. In the FME environment three

Table 5.9: Some statistics of the real-life experiments with the HB-CR-POMCP Searcher & Tracker, per environment: the total time, length of the path of the robot, and person, average distance between the robot and person, average speed of the robot and person, and the visibility of the person. The *total* row shows the sum for the first three columns, and the average over all the data for the other columns. *Measurements which include the person were only measured when the person was visible to the robot.

Map	Time [min]	Robot distance [m]	Meas. dist. person* [m]	Avg. dist. to person* [m]	Avg. speed robot [m/s]	Avg. speed pers.* [m/s]	Visibility [%]
FME	135	834.8	414.7	3.6 (± 1.8)	0.25 (± 0.93)	0.56 (± 3.82)	47.6
BRL	19	188.7	187.5	2.7 (± 1.9)	0.16 (± 0.26)	0.35 (± 0.58)	56.8
Telecos Sq.	72	844.6	725.8	3.8 (± 2.8)	0.21 (± 0.53)	0.43 (± 1.08)	57.0
Total	226	1868.2	1328.0	3.5 (± 2.4)	0.21 (± 0.34)	0.45 (± 1.75)	53.0

experiments were done and all of them were successful (i.e. without errors). The total robot movement was 160 m, the total person movement was 35 m (when visible) and the total experiment duration was 702 s. In the Tel. Square environment 18 experiments were done, of which 11 were successful (the robot was able to search and track the person), and in 7 experiments several problems occurred. The issues were not related to the search-and-track method, but to software failure or hardware problems (e.g. temperature or low battery) that stopped the robot. In the 11 experiments the robot moved 1173 m, the person moved 148 m (when visible) and the total duration of the experiments was 5925 s (more than 1.5 h). Table 5.10 shows an overview of all the experiments' statistics, which are based on the data measured by the robot's sensors, so the distance traveled by the person was more than the values reflected in the table. The videos of the experiments and more information can be found on http://alex.goldhoorn.net/thesis/st-pf/.

We were not able to get the same statistics as the simulations (Tables 5.7-5.8) for the real-life experiments, due to two main reasons. First, the ground truth should be known to have correct evaluation measurements like in the simulations, and since most global person localization methods have a low precision, we decided not to use any. Instead, we used the person location obtained with the leg and AR marker detectors (see Section 3.5), which allow us to calculate the *distance to the person* when the

Table 5.10: Summary of the real-world experiments with the HB-PF Searcher & Tracker. [1]When the person is visible to the robot, because we used the robot's sensors.

Measurement	FME	Tel.Sq.
Average distance to person[1] [m]	2.8 ± 1.4	4.2 ± 2.4
Person visible time [%]	19.6%	3.2%
Average number of dynamic obstacles	4.0 ± 1.7	3.2 ± 2.4
Max. number of dynamic obstacles	8	16

person is visible. Second, several problems during the experiments caused the robot to stop a few seconds, extending therefore the time measurements. Nevertheless, we are able to show some descriptive statistics, as shown in Table 5.10, and we were able to obtain more specific statistics for the search and search-and-track experiments shown in Table 5.11.

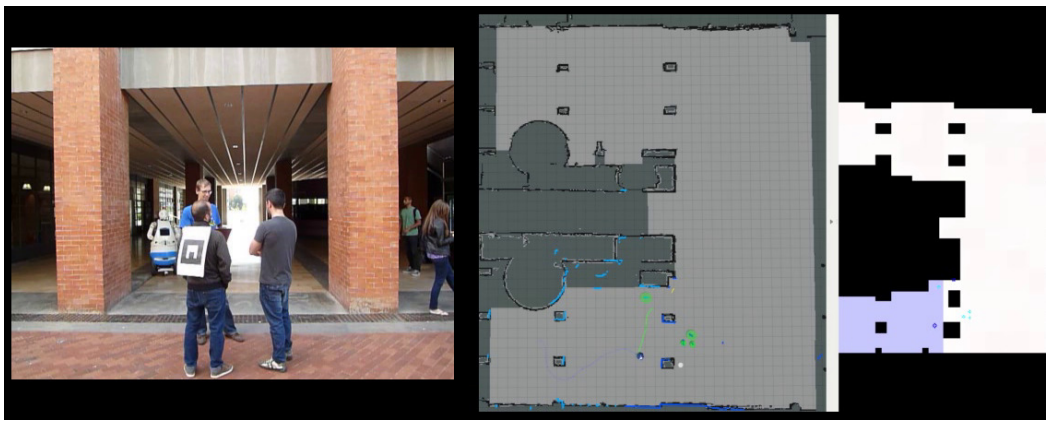
For the search experiments we calculated the time to first seeing the person; the tracking is counted from when the person is found, and in this case, we measure the average distance to the person, visibility, and the recovery time. Note that the *Belief Error* can not be calculated since it requires us to know the ground truth.

5.9.3.1 Smaller map – FME

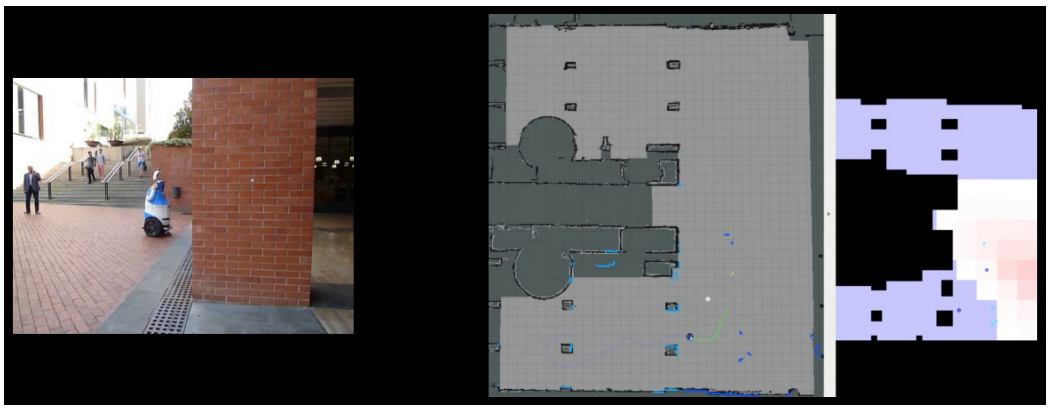
The FME map (Figure 5.12(a)) was slightly easy for the robot, since the area is relatively small, however, there were always two fixed obstacles where the person could hide. Furthermore, it can be seen in Figure 5.12(a) that the people standing in front of the robot also were used as obstacles, because there was a belief (i.e. particles) behind them.

In Table 5.11, for the FME, we only show one experiment because this was a complete search-and-track experiment, and in this experiment there were four people walking around. We can see that the distance to the person and also the recovery distance (comparing it to the recovery time) are comparable to the results of the simulation (Table 5.7).

(a) Experiments at the FME (17 m × 12 m).



(b) Experiment at the Telecos Square (60 m × 55 m).



(c) Other experiment at the Tel. Square.

Figure 5.12: The experiments using the HB-PF Searcher & Tracker in two environments. The center image shows the map used for localization with the robot (blue), detected people (green), and the laser range detections. The right image shows the probability map, i.e. belief, of where the person could be where red is a high probability, white low, and light blue zero. The blue circle indicates the location of the robot, the light blue circles are the locations of other nearby people, and the cross is the robot's goal.

5.9.3.2 Bigger map – Telecos Square

In the bigger map, the robot needed more time to explore the environment, because of its size and the relatively low speed of the robot. Compared to the experiments in subsection 5.9.2, here we also includes searching in the presence of dynamic obstacles, which has been shown to be important in several experiments. In various experiments, the robot looked for the person behind one or more dynamic obstacles, such as for example in Figure 5.9.3(a) and (b), where three people in front occlude part of the map, which otherwise would be visible to the robot.

In some occurrences of false positive detections, the robot stayed longer than required, because the robot assumed that the person was near to it. Then, the robot recovered either because it detected the real person or the false positive detection disappeared. The false detections were because other person's legs were detected in combination with a falsely detected AR marker.

In Table 5.11, we include for the real-life experiments the same measurements that we did in simulation. The *first visible step* should be compared (between simulation and real-life experiments) using the distance and not the time, since the robot stopped several times without moving due to external problems. Since in the real experiments there were around 5 to 10 other people walking around in the scene, we should compare it to the 105.6 ± 285.5 time steps in simulation, which is about 84.5 ± 228.4 m (0.8 m/cell). In the real experiments these values were lower, which might be because only a few search start positions were tried, whereas in simulations many completely random positions were tried. The visibility (in the tracking phase) was lower in the real experiments, mainly because the robot's vision system is worse than in the simulated version, which for example assumes a full panoramic view; note that in Table 5.10 the visibility was very low, because it includes exploration experiments in which the person was not present. The average distance to the person when tracking for the simulations was 7.5 ± 7.5 m, which is more than in the real world, because there it could only be calculated when the person was visible. The lower real-world recovery distance could be explained by the low speed of the person.

The real experiments showed us that the HB-PF Searcher & Tracker is able to search and track the tracked person, also when other people are partly blocking the

Table 5.11: Here the average values are shown for the real-world experiments. [1]Only person locations are used of measured points, i.e. when visible to the robot. [2]The data for the FME has only one experiment.

Measurement	FME[2]	Tel.Sq.
First visible time [s] (search)	4	132.5 ± 84.8
First visible distance [m] (search)	0.35	37.6 ± 23.9
Visibility [%] (track)	25.2%	16.0%
Distance to person[1] [m] (track)	2.9 ± 1.4	4.2 ± 2.5
Recovery time [s] (track)	17.1 ± 18.6	22.9 ± 40.0
Recovery distance [m] (track)	3.0 ± 5.5	4.3 ± 6.6
Robot movement/experiment [m]	105.5	62.5
Person movement[1]/experiment [m]	32.0	13.2

field of view.

5.9.3.3 Examples of the Experiments

Finally, we will explain shortly some of the real-life experiments using the HB-PF Searcher & Tracker, the videos can be found online.

Searching and tracking in the FME with Dynamic obstacles: Figure 5.13 shows the experiment done in the FME with up to four other people walking around (as dynamic obstacles); it can be seen that the agent assumed the tracked person could be hidden behind the people. The experiment started (a) with the person with tag being hidden behind two other people; then quickly the robot detected the person (b) when he became visible enough; but after not detecting him for a longer time, the belief propagated behind the obstacles (c), but later on the person was recovered again (see the video); at the end of the experiment (d) the AR marker detection algorithm falsely detected a marker in the corner where another group of people was sitting.

Searching and tracking in a larger environment: An experiment done at the Telecos Square is shown in Figure 5.14, which again started with the person being occluded by two people in (a); next (b) the robot found the person and; (c) followed the person, but lost him since he walked a bit faster, nevertheless, the belief was spread in the area where the person was; and a bit later the person was found again (d).

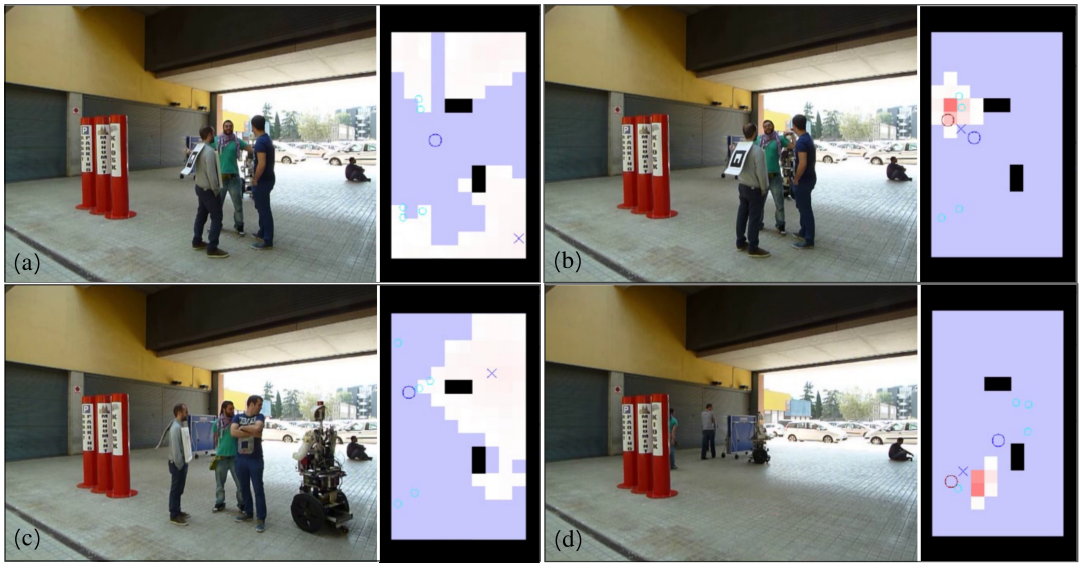


Figure 5.13: An experiment at the FME, where the person started hidden behind two other people.

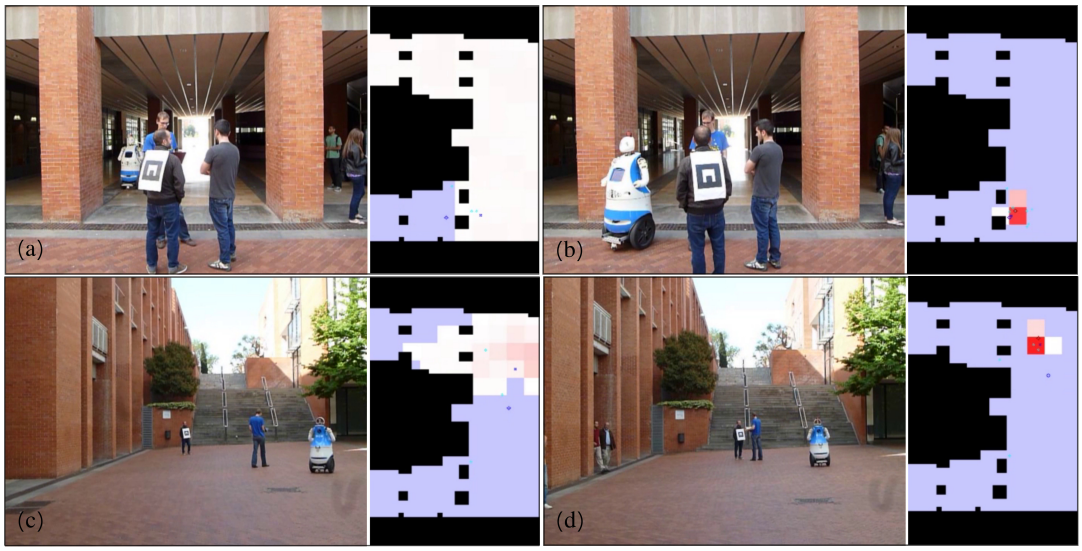


Figure 5.14: An experiment at the Telecos Square, where the person was hidden behind two other people, then discovered, and later lost temporarily due to the distance.

5.10 Discussion

The first presented method, CR-POMCP, was found to work well in simulation for hide-and-seek in continuous state space, but when we used it with the real robot however, we

encountered several problems. This led to the HB-CR-POMCP, and finally, we added a tracker to go to the exact person's location, which led to the HB-CR-POMCP Searcher & Tracker. In the HB-CR-POMCP method, the POMDP is only used to maintain the belief and the RL policy is not used anymore. Since the POMCP belief update method works the same way as a PF, we introduced the HB-PF Searcher & Tracker. The latest method worked as well as the CR-POMCP based method, and furthermore, is simpler. In these methods, the Highest Belief is used to localize the person.

5.10.1 Comparison of Methods

The final methods, the HB-PF Searcher & Tracker and the HB-CR-POMCP Searcher & Tracker, implement a different belief update method, but have in common the HB calculation and the *goal selection*. However, when we look in detail to both methods, the belief update is very similar because they both use Monte-Carlo simulations to propagate the particles (or belief points). This, therefore, explains their similar results.

Our methods work on-line and are able to both search and track a person, and do not require an initial observation, in contrast to trackers as [Fleuret et al., 2008, Lian et al., 2015].

In Volkhardt and Gross [2013], they searched for a person, and to detect him/her, they looked at their legs, face, body-shape and motion, which is a more realistic method than our marker; however, the marker allowed us to do experiments in a large outdoor environment. Their search was guided by a list of predefined guide points, and they assumed there to be only one, not moving, person.

Granata and Bidaud [2012] used a fuzzy controller to decide on the use of different behaviours, such as returning to the last seen location, whereas we keep track of the person's location through a belief map, which our method also uses to search for the person. Furthermore, they did experiments in a relatively small indoor environment, whereas we did outdoor experiments in large urban environments, where we focused on searching.

5.10.2 Noise

Noise handling is done by adding noise to the belief points and particle filters. The CR-POMCP adds Gaussian noise to the observations and state in the POMDP simulator $\mathcal{G}$ in Algorithm 5.11. Our PF method adds noise to the state when doing the step and choosing the direction (Eq. 5.9).

5.10.3 False Positive and Negative Detections

False positive and negative detections depend on the sensors and environment, but they are important to take into account, since they do occur. Monte-Carlo based methods can cope with noisy signals due to the simulated noise, which is done by the POMCP and PF methods.

CR-POMCP: For the CR-POMCP method, the *false positive* and *false negative* handling depends mainly on the POMDP simulator $\mathcal{G}$, shown in Algorithm 5.11. The simulator is used to generate new belief points through the internal simulations. After having received the new observation, it is used to choose the next belief from the policy tree (see Figure 5.2). Next, $\mathcal{G}$ is used again to fill the belief $\mathcal{B}$ until it has n_{belief} points. The simulator function $\mathcal{G}$ has several internal parameters which allow us to handle false positive or negative detections.

In the case of a *true positive* or *true negative* detection, the observation is consistent, and therefore the belief choice will be correct. When a *false negative* detection occurs, the observation is *hidden*, while it should have been the person's location. This is coped with (in Algorithm 5.11) by generating the observation *hidden*, even though the person should have been visible.

A *false positive* detection can occur for two reasons: first, because the sensors detect something incorrectly as person; or second, because the obstacle is bigger on the map as in reality (due to the lower resolution), and therefore, the person is visible while this was not expected (according to the ray tracing function applied on the map). The latter case results in the correct person location, but the first case is problematic, since we cannot recognise a *false positive* detection, and it causes the belief to be focused on the incorrect location. Seeing the person on the correct position (i.e. the second

case) is handled in line 12 (in Algorithm 5.11), by returning the location of the person as observation, and for the first case, a random position is generated in line 14. This results in a policy tree where an observation o_{rand} of this randomly chosen person position actually results in a belief B_o, which contains the expected state s'. Note, however, that the belief B_o will contain much more states that represent observation o.

Particle Filter: For the PF, the *false positive* and *false negative* detection handling is done by the weight calculation (Eq. 5.10), because the weight represents the probability of the particle to be propagated. In the case of a negative detection (i.e. *hidden*), a *true negative* will result in the weight w_{cons} (third case of Eq. 5.10), since the visibility function P_{vis} (Eq. 5.7) returns 0 probability if the particle is not visible from the agent's location. If P_{vis} returns a higher value, we assume it to be a *false negative* detection and therefore give it a lower value (fourth case of Eq. 5.10). In the case of a positive value (i.e. we observe a position), the weight is calculated using the exponential function (second case). The weight will be higher if the particle is closer to the observed location, which is good if it is a *true positive*, but if it is a *false positive* this will disturb the probability distribution.

5.10.4 Obstacle Modelling

Obstacles were modelled as discrete blocks of about 1 m × 1 m and being higher than the agents. In our first methods we modelled the obstacles to block the complete vision, however, some obstacles only result in partial occlusion, such as a small tree. Therefore, in the CR-POMCP we introduced a probability of seeing "through" an obstacle: $p_{\text{see\_occ}}$. The PF method tackles this problem by letting particles behind the obstacle have some weight (Eq. 5.10).

The first visibility check method assumed to have infinite vision, but later on, we introduced a visibility probability function (Eq. 5.7) that takes into account the distance. This probability function allowed us to simulate the real environment more realistically, probabilistically instead of deterministically.

Finally, dynamic obstacles were included in the model, taking them into account when calculating the visibility probability. This allowed us to better predict the visibility, and thereby, the potential locations of the person.

5.11 Conclusions

In this chapter, we have improved the method to search-and-track with respect to the previous hide-and-seek methods in various ways. First, the methods allow for a much larger space to work on; second *continuous state space* is used; and third, the algorithm run in *real-time*.

Instead of a POMDP, we have used a model that does Monte-Carlo simulations to generate a policy: the CR-POMCP that makes use of the POMCP. In this method, each time step an action is planned, however, in real-life experiments with the robot we found that this lead to not efficient movements, therefore, we decided to use the belief of the CR-POMCP to find the most probable location, i.e. the Highest Belief. Finally, the HB-CR-POMCP was combined with a tracker that works in the case of seeing the person, to assure a precise localization of the person: the HB-CR-POMCP Searcher & Tracker.

Next, a modified Particle Filter, the HB-PF Searcher & Tracker was created and is able to work without observing the location of the person. The PF model was used to keep track of the belief, and by obtaining the HB, the method is also able to search and track the person.

Comparing the HB-PF Searcher & Tracker method with the HB-CR-POMCP Searcher & Tracker method, the first method is simpler, because it is easier to modify the algorithm strategy in the PF than in the POMCP, even though the simulation results are similar.

Furthermore, we have presented a method that takes into account dynamic obstacles while searching and tracking a person. This, as expected, improves the search in crowded areas, because it maintains a probability of the person being behind any of the dynamic obstacles.

Chapter 6

Search-and-Track with Multiple Seekers

Using multiple robots instead of a single robot has several advantages, such as faster execution of the task and robustness, however, in order to do reach these, the robots should be coordinated correctly and efficiently. In this chapter, we use several agents to search and track the person. The methods are based on the HB-CR-POMCP Searcher & Tracker and HB-PF Searcher & Tracker, presented in Chapter 5, but here, we extend them to be used on multiple robots that communicate the observations and use an exploration method to divide the goals between the robots.

With the, in this chapter, presented methods we brought the following contributions:

- The Multi-agent HB-CR-POMCP Explorer and Multi-agent HB-PF Explorer were presented; they allow a group of robots to perform search-and-track [Goldhoorn et al., 2017a].
- To distribute the search task, the robots use an explorer to decide the goals for all agents [Goldhoorn et al., 2016, 2017a].
- Real-life experiments were done [Goldhoorn et al., 2016, 2017a].

6.1 Introduction

Even though it may seem obvious that the search by multi-agents should be faster than by a single agent, an efficient and robust cooperation is not that evident. Within the

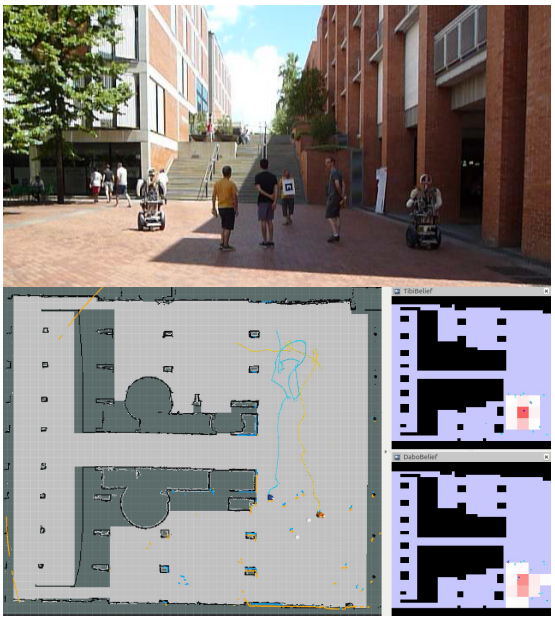


Figure 6.1: The robots search and track the person while other people are walking around obstructing the robots' vision. In the lower map left the localization of the robots can be seen (orange and blue robots), and at the right the probability maps of the person's location of both robots are shown.

area of social and cooperative robots, the nature of interactions between people and a set of accompanying robots has become a primary point of interest. Our findings are based on the behaviour of a team of robots, which operate cooperatively, to localize people in an urban area and are able to follow them. The person can hide out in the environment, while the robots, using a map, have to look for him/her.

We present a decentralized multi-robot approach that can search and track using one or more robots that cooperate, communicating their observations and most probable locations of the person. Internally the agents use a probability map of the person's location generated by either the *Multi-agent HB-CR-POMCP Explorer* or *Multi-agent HB-PF Explorer*, for which the observations of all robots are used. Thereafter, the most probable locations are marked and sent to all robots, such that each robot can choose the best location to explore. In the case of a visible person, the robots track the person while updating the probability map, since the observation could be a false positive.

In Chapter 5, two methods were introduced to search and track the person: HB-

CR-POMCP Searcher & Tracker and HB-PF Searcher & Tracker, which in contrast to other previous approaches can run in real-time in large environments in continuous state space, but in this chapter we go one step further. We make use of a team of cooperative robots that work in dynamic environments and share their sensors information to increase the performance of the task. Moreover, the robots are able to plan the strategies and to localize people in a cooperative way. Their search strategy takes into account the probability of a person being at a certain location, the distance to that location, and whether the location is close to a search goal already assigned to another robot. Even though, we make use of cooperative robots, they are also able to execute the task alone, in case of communication failure for example.

Finally, the validation of the approach is accomplished by an extensive set of simulations and a large amount of real experiments in an urban campus environment with dynamic obstacles, using our mobile social robots Tibi and Dabo. In the remainder of this chapter, we start with the system overview in Section 6.2 and the specific experimental settings in Section 6.3. In Section 6.4 we start explaining the methods to maintain the belief cooperatively, next in Section 6.5, the goal selection method is explained. Thereafter, in Section 6.6 the simulations and in Section 6.7 the real-life experiments are detailed, and we finish with a discussion and conclusions.

6.2 Overview of the Approach

In this chapter, we present a method for multiple mobile robots to search and track a person autonomously and cooperatively, which at the same time allows the robots to operate individually when there is no communication between them or when only a single robot is available. The method is shown in Figure 6.2, and is partly similar to the architecture explained in Section 5.3, but the main difference is that it works simultaneously with multi-agents at the same time.

The method (Figure 6.2) consists of four phases: the first phase, *perception*, is explained in detail in Section 3.2. The observation filter is used to choose an observation, and possibly correct either the person location or the robot location or both of them. For the multi-robot case, for each observation $o = \langle o_{\text{agent}}, o_{\text{person}}, p_o \rangle$ the probability of correctness of the observation (p_o) was added, to distinguish between the correctness of

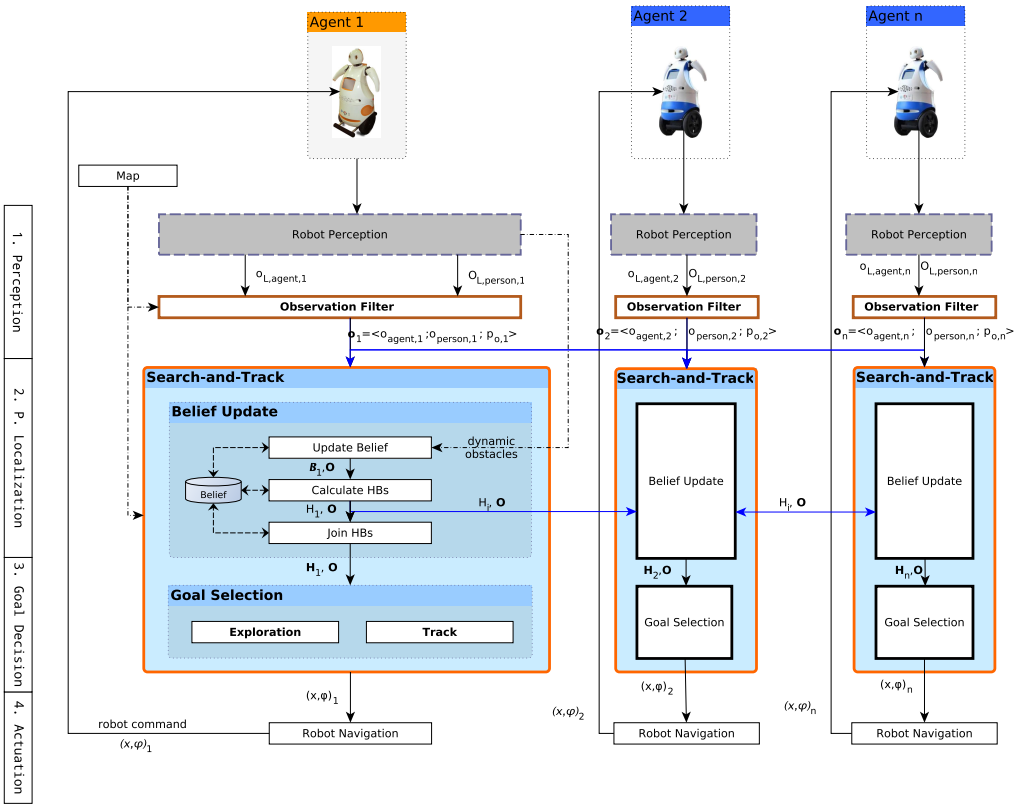


Figure 6.2: The schema shows the complete distributed search-and-track multi-agent approach with the same diagram for each agent and the communicated data, where the diagrams of the first agent are shown in detail. At the left the phases of the search-and-track method are shown. The blocks are algorithms or groups of algorithms, the orange bold lined blocks were created by us. The arrows show how the output of one algorithm is used as input for another and the communication between the agents (blue lines).

the observation based on for example the sensors. For Tibi we put a lower p_o, because its camera had a smaller fov than the camera of Dabo, while in our models, we assume to have full 360° fov.

Together with the observations of the other agents, the belief is updated in the *Person Localization* phase, for which we tried two algorithms: the Multi-agent HB-CR-POMCP Explorer (based on HB-CR-POMCP, see Section 5.6) in which each robot uses the probability map of the CR-POMCP to search and track a person; and the Multi-agent HB-PF Explorer (based on HB-PF, see Section 5.7) which makes use of a

PF.

In the third phase, the belief and observations are used to decide on the locations where the robots should search for the person—which we call goals. If a robot detects the person, then *Tracking* is carried out, otherwise an Exploration method is applied that chooses the goals for each robot from the list of highest belief points of all the robots. The goals are chosen taking into account the probability, the distance to the goal and whether another agent already has a goal close to it.

Finally, the robot's path planner (Section 3.4) plans and executes the path to the chosen goal. Each robot executes the same algorithm and they share the observations and HBs, as shown in the Figure 6.2.

6.3 Experimental Setup Improvements

In Chapter 3, the experimental settings were commented, and in Section 5.4 it was already indicated which other improvements were made to the simulations, such as: using a visibility probability (instead of a boolean visibility indication) and using dynamic obstacles. Here we mainly adapt the experimental settings for a multi-robot environment.

The input of the belief update algorithms is the list of observations $\boldsymbol{O} = \{\boldsymbol{o}_1, \boldsymbol{o}_2, \cdots, \boldsymbol{o}_n\}$ of the different agents, as shown in Figure 6.2. Each observation $\boldsymbol{o}_i = \langle o_{\text{agent},i}, o_{\text{person},i}, p_{o,i} \rangle$ contains the location of the robot ($o_{\text{agent},i}$), the observation of the person ($o_{\text{person},i}$, which can be *empty*) and a correctness probability ($p_{o,i}$), which indicates how much a person detector can be trusted. We estimated the correctness probability based on experimental results. For example, Tibi had a worse camera than Dabo, as explained in Section 3.3, and therefore we put a lower value for $p_{o,\text{Tibi}}$.

Ideally, the observations are sent and received synchronously, however, in the real world this was not the case. In simulation there was no delay of the observations of the other agents, but in the real-life experiments this did happen. Therefore, we set a time limit for the observation to be used in the other robots. If the observation was too old, it was not used. Although asynchronous or delayed observation messages do result in a different belief, in a later step, the agents send each other's highest belief points.

The *observation filter* makes sure that the locations of the person and robot are legal (i.e. within the map and not in an obstacle) by taking the closest most probable location.

6.3.1 Visibility Probability

The probability of visibility is based on Eq. 5.7, and for the multi-agent case, the probability is combined to calculate the probability of seeing location s_2 from any agent location $s_1 \in S$:

$$\bar{P}_{\text{vis}}(S, s_2) = 1 - \prod_{s_1 \in S} (1 - P_{\text{vis}}(s_1, s_2)) \tag{6.1}$$

6.4 Belief Update

In the present section, we explain the *Person Localization* phase, in which, we create the belief to find the most probable location. Two methods are proposed to create the belief: the first is based on the CR-POMCP (Section 5.5) and the second is based on the PF (Section 5.7). Finally, we explain how the belief is processed to select the highest belief points, and how this information is combined with the other agents' results. The next section explains how this belief is used to decide where the agents have to go to.

6.4.1 Multi CR-POMCP

In Section 5.5, we explained the single agent CR-POMCP, and here we extended the model for multi-agent systems. The states for the multi-agent case were not changed, because adding the other agents in the states would exponentially grow the amount of states. For the same reasons the observations of the other agents were not used in the learning phase, otherwise the policy tree would grow too wide. The other agent's observations are used, however, in the belief initialization and update phase, as explained in the rest of this section. Furthermore, our main interest is to keep track of the location of the person.

For the multi-robot case, all the observations have their own probability ($p_{o,i}$) which were added to take into account the accuracy and trustworthiness of the sensors of specific robots. To generate the initial belief, n_{belief} states are generated,

as shown in Algorithm 5.10, with two differences: first, in each iteration a random $o = \langle o_{\text{agent}}, o_{\text{person}}, p_o \rangle \sim \boldsymbol{O}$ is chosen with probability p_o; second, the visibility probability function for multi-agents Eq. 6.1 is used instead of Eq. 5.7.

After having set the belief, the POMCP policy tree is created (see Algorithm 5.7), as explained in subsection 5.2.2, by doing n_{sim} simulations. Then, the best action is chosen from the policy tree, and when it has been executed, the *belief update* is done.

Before the belief update, all the observations $\boldsymbol{O}$ are received from all agents, as can be seen in Figure 6.2. The belief is updated with the observation o, which includes only the information of the own agent, because including other agents' positions would make the policy tree grow very wide, and this would grow exponentially the policy search time.

The belief is first updated by choosing the new *Belief root node* from the policy tree. Since only the own observation is included in the POMDP model, as second update step, the states in the new belief are checked for consistency with all the observations. States that are found to be inconsistent are removed from the belief. Algorithm 6.13 shows the consistency check, which is done for each state in the belief $s \in \mathcal{B}$, using the observations of all agents $\boldsymbol{O}$.

First, it is checked if the observed person locations are not *hidden* (line 4) and if they are close to the belief state s (line 5), within a distance of d_{cons}. Then, Eq. 6.1 is used (line 12) to calculate the probability that the person location of the state should be visible to any of the agents. Finally, to take into account the sensor and actuator noise, and uncertain, a uniformly random function PRAND is used to decide the consistency, which uses the visibility probability function. This results in a Monte-Carlo-like sampling method.

As third step, states are added to the belief until it has n_{belief} states, like the CR-POMCP method. Each new state s_{person} is randomly chosen from $\boldsymbol{O}$ with probability p_o, and if s_{person} is *hidden*, then it is set to a random location where the person is not visible to any of the agents; finally, Guassian noise is added with standard deviation σ_{person}.

The consistency check algorithm sometimes may reject a consistent observation as being consistent, because we make use of the probability visibility and a random value

Algorithm 6.13 The belief consistency check with as input state s and observation list $\boldsymbol{O}$. The function PRAND randomly generates a value between 0 and 1.

```
1: function ConsistencyCheck(s, O)
2:     isVisible = false
3:     for o ∈ O do
4:         if not o_person is hidden then
5:             if ‖s_person − o_person‖ > d_cons then
6:                 return false
7:             else
8:                 isVisible = true
9:             end if
10:        end if
11:    end for
12:    p = P̄_vis({o_agent | o ∈ O}, s_person)
13:    if isVisible then
14:        if PRAND( ) > p then return false
15:        end if
16:    else
17:        if PRAND( ) ≤ p then return false
18:        end if
19:    end if
20:    return true
21: end function
```

PRAND; this allows the method to take into account possible lack of visibility of the sensors.

6.4.2 Multi PF

For the multi-agent PF some changes have to be made to take into account all the observations. First of all, the initialization has to be changed, and then we change the *weight* function of the *update* phase of the PF.

The initialization is the same as the Multi-agent HB-CR-POMCP Explorer, explained in the previous section. The prediction step of the PF algorithm (see subsubsection 5.2.3.2) is a Gaussian movement, as shown in Eq. 5.9. Next, the update step is equal to the single agent case, shown in Algorithm 5.12, except for the weight calculation in line 2, which is changed by:

$$\forall_{s \in S} : s_w = \text{agg}_{o \in O}(w(s, o)) \tag{6.2}$$

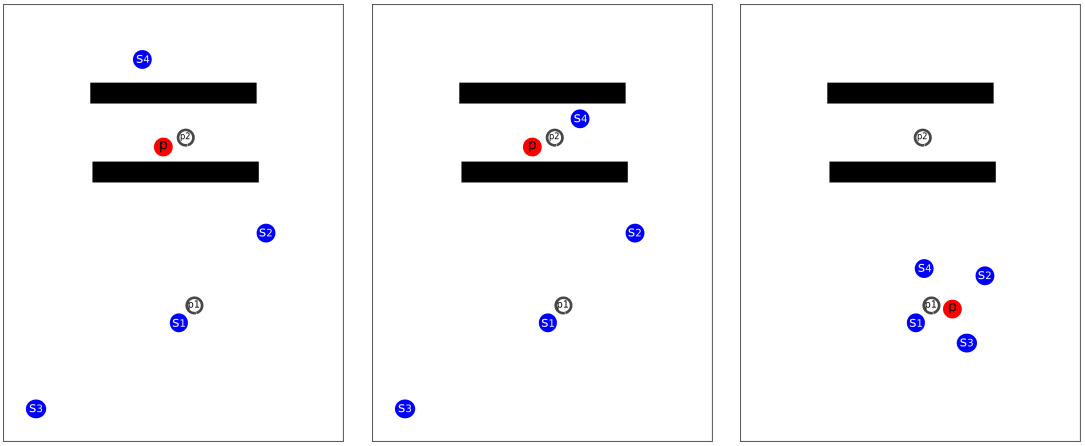
calculating the weight of each particle $s \in S$, using Eq. 5.10, and then aggregating them using either the *minimum, maximum* or *average*. In order to decide on which aggregation function is the best, we can differentiate three situations with the examples shown in Figure 6.3. In these examples there are four agents ($s_1 - s_4$), one person p, and two particles p_1 and p_2:

1. *None of the agents see the person* (Figure 6.3(a)): p_1 can only be seen by s_1, and not by the rest due to distance or occlusion. Particle p_1 should be visible for s_1, and therefore, its weight is the last case of Eq. 5.10: $w_1^1 < w_{\text{inc}}$, since it is inconsistent with the observation; and the weight for the others is: $w_1^{2,3,4} = w_{\text{cons}}$. Here the best weight would be the *minimum* value w_1^1, since the particle is at a location where most probably the agent s_1 would have detected the person if it were at location p_1. The particle p_2 is not visible from any agent, and therefore, is consistent with not having an observation.

2. *Some agents see the person, others do not* (Figure 6.3(b)): For s_1 the particle is visible, but it does not have an observation so the value is inconsistent, therefore we use $w_1^1 < w_{\text{inc}}$ (Eq. 5.10), for s_2 and s_3 the particle is too far, and therefore, $w_1^{2,3} = w_{\text{cons}} = 0.01$; and for s_4 the weight depends on the distance, which in this case is high, resulting in a low weight. Again, the best option is to choose the *minimum* value, because for s_1 the person should have been visible. Particle p_2 is visible to s_4 and therefore has a larger weight $w_2^4 > 0.1$ than the weights calculated for the other agents, which will have a $w_2^{1,2,3} = w_{\text{cons}} = 0.01$. Here we actually would prefer to choose the *maximum* weight, since this favours the observation of s_4.

3. *The person is visible to all robots* (Figure 6.3(c)): since all the agents have an observation, the second case of Eq. 5.10 is used to calculate the weight. Here the *average* would be best since there is no knowledge about which observation is the closest to the real person location.

To combine the values we could use the *maximum*, *minimum*, or *average*; in the first case, *maximum*, we would keep particles even though they should be visible to a seeker but are not, such as can be seen in the first example. The *minimum* would work in most cases, except in example 2, where s_4 sees the person, and therefore has



(a) The person is not visible. (b) The person is visible to one agent (s_4). (c) The person is visible to all agents.

Figure 6.3: Three examples to show the best choice for an aggregation function, with four seeker agents (s_1 - s_4) and one tracked person (p). The white circles p_1 and p_2 are particles.

a much higher weight than the others. For the last example neither the *maximum* nor the *minimum* would be correct. Using the *average* in the third example would be best, and it also would work for the other examples, since it takes into account all values.

6.4.3 Highest Belief

Like explained in Section 5.6 and subsection 5.7.2, a 2D histogram is made by counting the number of particles or belief points per cell, and by dividing them by the total ($n_{\text{particles}}$ or n_{belief}), to get the probability of the person being there. The size of the cells of the histogram should be large enough to be stable, but small enough to be sufficiently precise. In our experiments, we used cells of 3.2 m $\times$ 3.2 m for the large maps, and 1 m $\times$ 1 m for the small map.

Next, for each agent i, the set H_i of n_{hb} highest belief points are selected, and are sent to the other agents (see Figure 6.2). Each $h \in H_i$ contains a position h_{pos}, and a belief h_b. The received highest probability points H_i of all other agents and of the agent itself are joined by summing the beliefs for each highest probability point, and thereby, generating the set of all highest belief points $\boldsymbol{H}$, as can be seen in Algorithm 6.14.

Algorithm 6.14 Join the lists of highest belief points of all n agents $H = \{H_1, H_2, \cdots, H_n\}$, where each highest belief point $h = \langle h_{\text{pos}}, h_b \rangle \in H_i$ contains a position and belief h_b.

```
1:  function JoinHBs(H)
2:      H = ∅
3:      for all H_i ∈ H do
4:          for all h ∈ H_i do
5:              if h_pos ∈ H_pos then          ▷ H_pos is the list of positions in H.
6:                  Get ĥ ∈ H where ĥ_pos = h_pos
7:                  ĥ_b = ĥ_b + h_b
8:              else
9:                  H = H ∪ {h}                ▷ add HB point.
10:             end if
11:         end for
12:     end for
13:     return H
14: end function
```

6.5 Goal Selection

After the belief is updated and the highest belief points are created, received, and joined, then, the *Goal Decision* phase (Figure 6.2) starts in which the robot either tracks the person or explores the most probable locations. If the person is visible and the observations are consistent then the agents follow the person side-by-side [Garrell et al., 2013, tracking in Figure 6.2]. Otherwise the agents explore the joined highest belief locations $\boldsymbol{H}$, as shown in Algorithm 6.15, which is based on the work of [Burgard et al., 2005]. Each agent calculates the goals for all agents, using the joined highest belief points $\boldsymbol{H}$. A score is calculated for each highest belief points $h \in \boldsymbol{H}$ per agent location s:

$$\text{EXPL\_SCORE}(s, h) = w_u U_h + w_d \frac{\text{DIST}(s, h)}{d_{\max}} + w_b \frac{h_b}{b_{\max}} \tag{6.3}$$

where U_h is a utility function for the highest belief point h, s the agent's position, h_b the belief of point h, and DIST calculates the shortest path distance. The second and third term are normalized by the maximum distance $d_{\max}$, and maximum belief $b_{\max}$ with respect to the list of potential target locations $\boldsymbol{H}$. The utility U_h [Burgard et al., 2005] is initialized with 1 (line 1 of Algorithm 6.15), and is updated before searching the goal of the other robot (line 4), where g_i is the already assigned goal to agent i,

Algorithm 6.15 The explorer finds the goals g_i for all agents $i \in \mathcal{A}$ using the score function Eq. 6.3.

1: $\forall_{h \in \boldsymbol{H}} U_h = 1$
2: **for all** $i \in \mathcal{A}$ **do**
3: $\quad g_i = \arg\max_{h \in \boldsymbol{H}} \text{EXPL\_SCORE}(s_{\text{person},i}, h)$
4: $\quad \forall_{h \in \boldsymbol{H}} U_h = U_h - P_{\text{explore}}(\text{DIST}(h, g_i))$
5: **end for**

and:

$$P_{\text{explore}}(d) = \begin{cases} 1.0 - \frac{d}{d_{\text{max\_range}}}, & \text{if } d < d_{\text{max\_range}} \\ 0, & \text{otherwise} \end{cases} \tag{6.4}$$

with $d_{\text{max\_range}}$ being the range within which we want to reduce the chance of other agents' goals being chosen. The terms of Eq. 6.3 are weighted by w_u, w_d, and w_b, and the values we found to work well are: $w_u = 0.4$, $w_d = 0.4$, and $w_b = 0.2$.

The order in which the agents are assigned the goals is important, since the assignment of a goal h to an agent reduces U_h, and therefore, reduces the probability of other agents being assigned this goal. We chose to assign the agent with the highest sum of probabilities ($\sum_{h \in H_p} b_h$) a goal first, and the lowest last. Most importantly, the order should be consistent for all agents such that they calculate the same goals, assuming they have received all highest belief points H_i. With this method it can occur that an agent a_1 is assigned a goal g_1, which is further away than a goal g_2 assigned to a_2, because the latest was assigned firstly. Note that finding the closest goals $g \in G$, such that the sum of the distances with the agents $a \in \mathcal{A}$ is minimum:

$$\min_{a \in A} \sum_{g \in G} \text{DIST}(a, g)$$

has a complexity of $O(\|A\|!)$. Therefore, we approximate it by re-iterating over the calculated goals and assigning the closest in the same order as we calculated the score (i.e. on sum of the HB). It can be seen that for agents from which no positions have been received no goals are calculated.

Finally, to prevent the robot from changing its goal too often, the goal is only changed every t_{update} time, or when the person is visible.

The method explained in this section is only guaranteed to give the same search goals if they receive all the highest belief points of all agents synchronously. If not all

highest belief points are received, the resulting search goal positions may be close to each other, which results in a less efficient search.

6.6 Simulations

Here, the simulations for the multi-agent models are discussed; they were done in the same way as explained in the single robot case in Section 5.8. The movements done by the robots were a step of 1 cell in the direction of the goal using the shortest path.

6.6.1 Multi-agent HB-CR-POMCP Explorer and Multi-agent HB-PF Explorer

In the simulations, the two belief update algorithms were tested: the Multi-agent HB-CR-POMCP Explorer and the Multi-agent HB-PF Explorer. For the latter, two variants were tried with different ways to fuse the observations of the different agents in the update phase: the *average* and the *minimum*. As an upper line we added a best-case algorithm, the See All Follower, which is a follower that always knows the location of the person, independent of the distance or any obstacles being between the agent and the person. Also the influence of the number of agents was tested using from one to five agents, which were searching simultaneously either with or without communication. For the visibility, the probability function Eq. 6.1 was used.

The experiments were done on the *Telecos Square* map (version 3 in Section 3.1). In this version of the Telecos Square map, we added access to another area through two ramps which created a loop. The loop made the experiments more interesting, since the person can now walk in circles, requiring the agents to cooperate.

A crowded environment was simulated by adding a group of 10 or 100 people (dynamic obstacles) to the scene, who reduced the robot's visibility, but they did not block the agents' paths. The movements of the simulated people (including the person to be found) were semi-random; they were given a random goal to which they navigated using a shortest path algorithm, and when the goal was reached, a new random goal was assigned.

More than 40 000 experiments were done, repeating each of the conditions at least 250 times. Like in the previous chapter, for each run of simulations the robot's start

position, and the person's start and path were generated randomly. To make the comparison as fair as possible the same positions were used for all the algorithms and conditions, such that the initial state and the person's path were the same.

6.6.1.1 Simulation Goals

The goals of the simulations were to see how well the presented search-and-track methods worked for multiple agents and under different circumstances. Here, we limited the tests to adding up to 100 dynamic obstacles and using up to five seekers, which either had communication or had not. We split the simulations in two types: first, in *searching* and, second, in *tracking*. For the first we wanted the person to be found as fast as possible, and for tracking, the agent should be close to the person as long as possible and see him/her. In all cases the See All Follower should work best, since it always sees the person.

The *searching* simulations were started with the person being hidden to all the agents and without moving. The simulations ended when either a robot reached the person at a distance of 1 cell (0.8 m in the used map), or 2 000 steps were reached. The simulations were measured using the time it took for at least one agent to see and to be next to the person. The *tracking* simulations were done with the person being visible to one or more of the agents and continued for 500 time steps. Other measurements were the distance between the agent and the person, hereby taking the lowest distance over all of the agents. Furthermore, for all simulations the belief error ε_b (Eq. 5.11) was measured.

6.6.1.2 Algorithm Parameter Values

The values of the parameters used in the simulations and real experiments, which were explained in Section 6.4 and Section 6.5, are shown in Table 6.1. The parameters $p_{o,\text{Tibi}}$ and $p_{o,\text{Dabo}}$ indicate the trustworthiness of the sensors, and since the vision of Tibi was less than the 360° vision of Dabo, we gave it a lower probability. The other parameters for the HB-CR-POMCP are shown in Table 5.3. All the parameters were tuned first in simulation, and later while doing tests with the real robots.

Table 6.1: The parameter values used during the real experiments and simulations.

Parameter	Value	Description
	Common Parameters	
σ_{person}	0.2 m	std. dev. of Gaussian noise person's movement
$p_{\text{v,max}}$	0.85	maximum probability visibility (Eq. 5.7)
α_{vis}	0.17	reduction factor (Eq. 5.7)
$d_{\text{v,max}}$	3.0 m	maximum distance full visibility (Eq. 5.7)
$p_{o,\text{Tibi}}$	0.3	trustworthiness of Tibi's observations
$p_{o,\text{Dabo}}$	0.7	trustworthiness of Dabo's observations
	Multi-agent HB-CR-POMCP Explorer	
n_{sim}	2500	number of simulations
n_{belief}	2000	number of belief points
$p_{\text{false\_pos}}$	0.001	false positive probability
$p_{\text{false\_neg}}$	0.3	false negative probability
$p_{\text{see\_occ}}$	0.01	probability of seeing through an obstacle
d_{max}	1	maximum POMCP tree search depth
d_{cons}	0.7 m	consistency check distance (Algorithm 6.13)
	Multi-agent HB-PF Explorer	
$n_{\text{particles}}$	2000	number of particles
σ_w	1.0	spread of particle weight (Eq. 5.10)
w_{cons}	0.001	weight (Eq. 5.10) when observation consistent
w_{inc}	0.0001	weight (Eq. 5.10) when obs. inconsistent
	Highest Belief	
cell size	3.2 m × 3.2 m	2D histogram cell size
n_{hb}	10	number of highest belief points
t_{update}	3 s / 3 steps	wait time to re-calculate goal
	Goal Selection	
w_u	0.4	weight for utility explorer score (Eq. 6.3)
w_d	0.4	weight for distance explorer score (Eq. 6.3)
w_b	0.2	weight for belief explorer score (Eq. 6.3)
$d_{\text{max\_range}}$	30 m	maximum range of influence score (Eq. 6.4)

6.6.1.3 Results

The results of the *search* simulations are shown in Figure 6.4, visualizing the average time (discrete steps) it took to find the person. The time is measured until one of the agents found the person and was next to him/her. The influence of communication is shown in the rows and the effect of the number of dynamic obstacles is shown in the columns. The See All Follower was only run for one agent. Since none of the data were normal we have used the Wilcoxon ranksum test, 2-sided to compare the different conditions.

For all cases, the See All Follower was significantly faster ($p < 0.001$) than any other algorithm, since it was always able to see everything. It can be seen that it took more than four times longer when using one agent with the PF that had visibility limitations. When using only one agent, the agents using PF were significantly faster

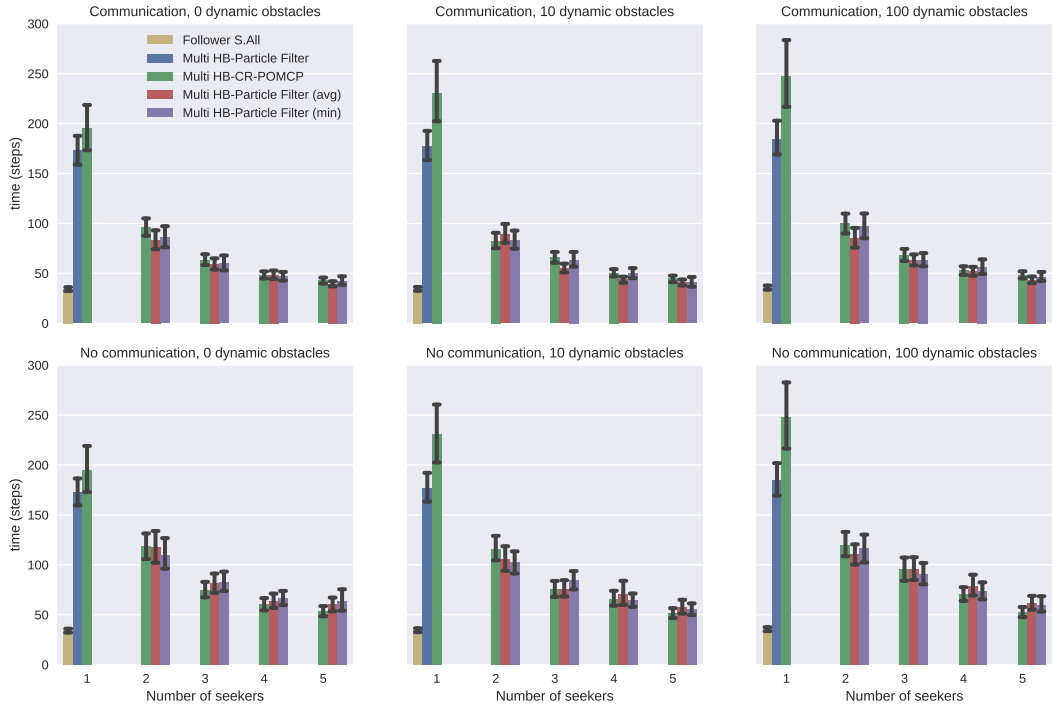


Figure 6.4: The graphs show the average (and 95% confidence interval bars) of the time it took for one or more agents to find and get close to the person. In the first row there is communication between the agents, in the second there is not. In the columns the number of dynamic obstacles change. As a reference, the See All Follower took 34.7 ± 0.6 steps (mean $\pm$ standard error).

than using the CR-POMCP ($p < 0.001$). For the multi-agent simulations the use of communication was also significantly better ($p < 0.05$), except for some cases with the Multi-agent HB-CR-POMCP Explorer. In most of the cases, the Multi-agent HB-PF Explorer was the fastest method, and in particular the version that used the *average* weight combination.

For the track phase, we would like the robot to stay close to and have the person visible as long as possible. Figure 6.5 shows the average time it took to find the person again after losing him/her. The *See All Follower* still was best, but between the tested methods there was no clear winner, nor did communication give an advantage for one or another method, which most probably was because the robots were close to the person already (see Figure 6.5). The increasing number of robots reduced the recovery time

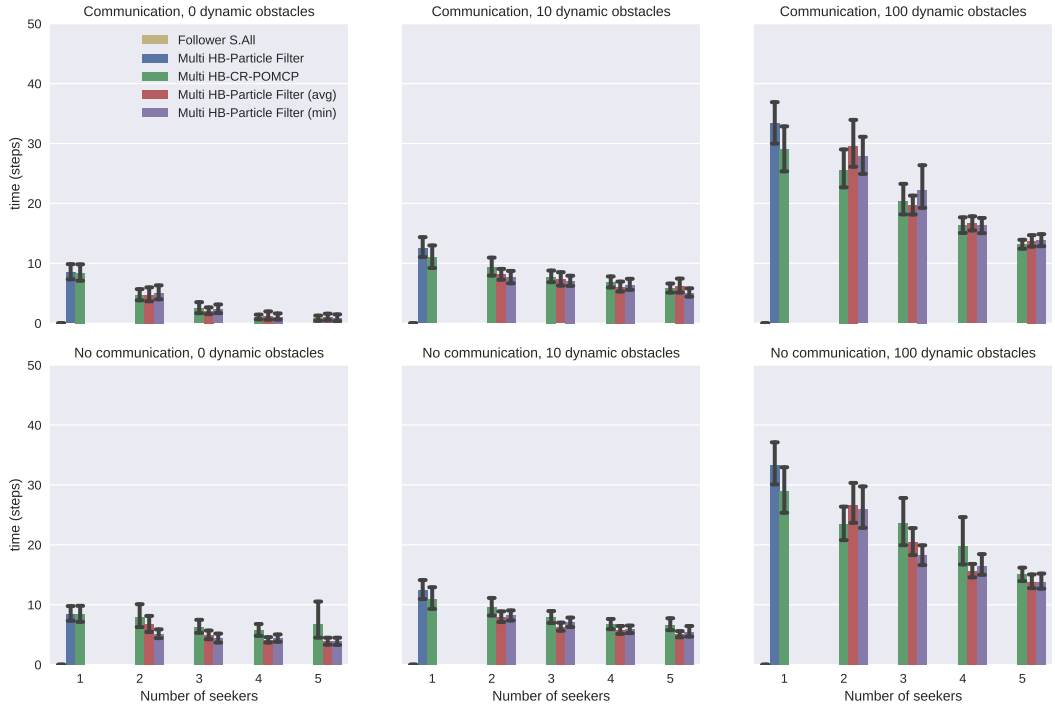


Figure 6.5: The graphs show the average (and 95% confidence interval bars) time to discover the person it is following after having lost it due to (dynamic) obstacles for example.

significantly, however, we did not simulate robots blocking each other's path, which in the real world would reduce the efficiency of having multiple robots in a small area.

The average distance between the person and the closest agent when following is shown in Figure 6.6. The PF method resulted in lower distances, and also using communication resulted in lower tracking distances.

The belief error (Eq. 5.11) was calculated for the algorithms, which use a probability map of the location of the person. For the search simulations, the overall average and standard deviation of the belief error were 25.4 ± 8.9 m when there was communication and 27.8 ± 7.5 m without. Figures 6.7 and 6.8 show the average belief error for the search and track phase respectively. The lowest belief error for the search simulations with communication was with the HB-PF Searcher & Tracker method, using the *minimum* weight combination. There was no clear difference in the other cases.

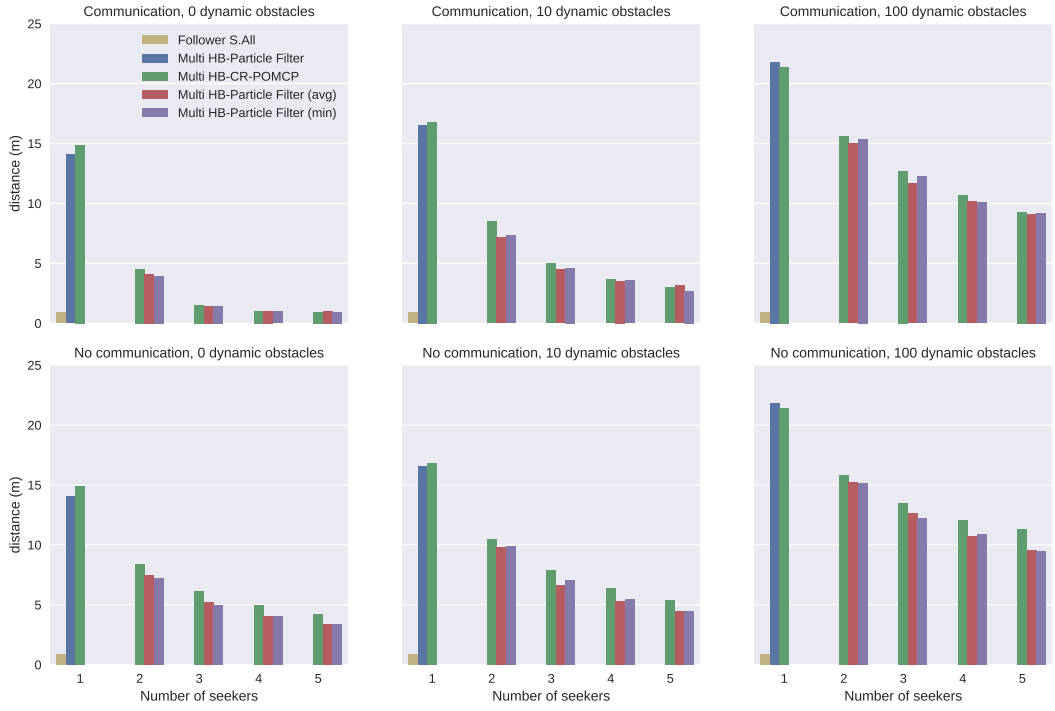


Figure 6.6: The graphs show the average distance between the person and the closest agent when following. The rows show communication or not, and in the columns the number of dynamic obstacles change. The See All Follower had the person always in sight and therefore was at a distance of about 0.88 m, i.e. the following distance.

The influence of having more dynamic obstacles is not clear (i.e. no significant difference for most cases) in the search time (Figure 6.4) because they only block the agents' vision and not the path, i.e. the robot can go through the dynamic obstacles. From Figure 6.5 it can be seen that 10 dynamic obstacles almost did not influence the recovery time, but 100 did. Because of the surface (1400 m^2) having 10 people walking around randomly had a low probability of influencing the vision of the robot, whereas 100 had a much higher probability, spreading the area more. Finally, the influence of dynamic obstacles can also be seen in the average distance to the person (Figure 6.6) and the belief error (Figure 6.8).

To summarize, we found that, as expected, the base line See All Follower was faster in searching, and it tracked the person during the longest time. For *searching* we found the HB-PF Searcher & Tracker to be faster than the HB-CR-POMCP Searcher

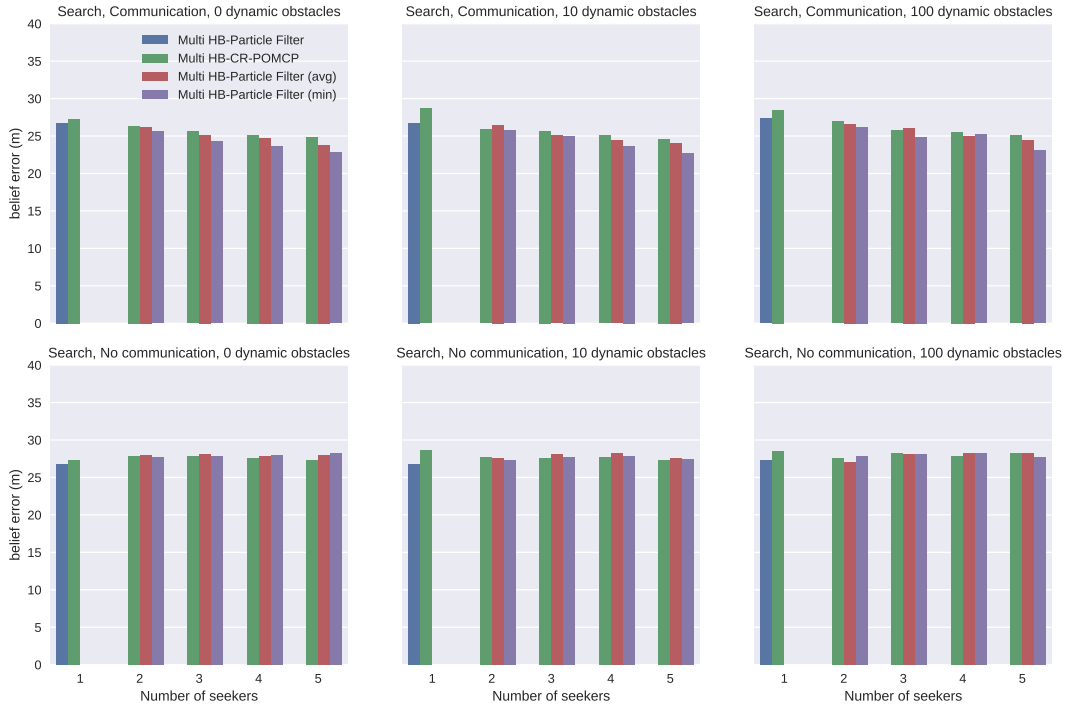


Figure 6.7: The average belief error (using Eq. 5.11) when searching. The rows show communication or not, and the columns the number of dynamic obstacles change. The See All Follower does not use a belief and is therefore not mentioned.

& Tracker in most cases, and in general, there was an improvement when using communication. *Tracking* showed no statistical difference between the methods (except for the See All Follower) in recovery time; it only showed that having more seeker agents resulted in better performance. And for the distance to the person while tracking, the HB-PF Searcher & Tracker showed slightly better results. As weight combination method for the HB-PF Searcher & Tracker when searching, the *average* was found to be slightly faster, but the *minimum* resulted in a slightly lower belief error.

6.7 Real-life experiments

Now we discuss the results of the real experiments.

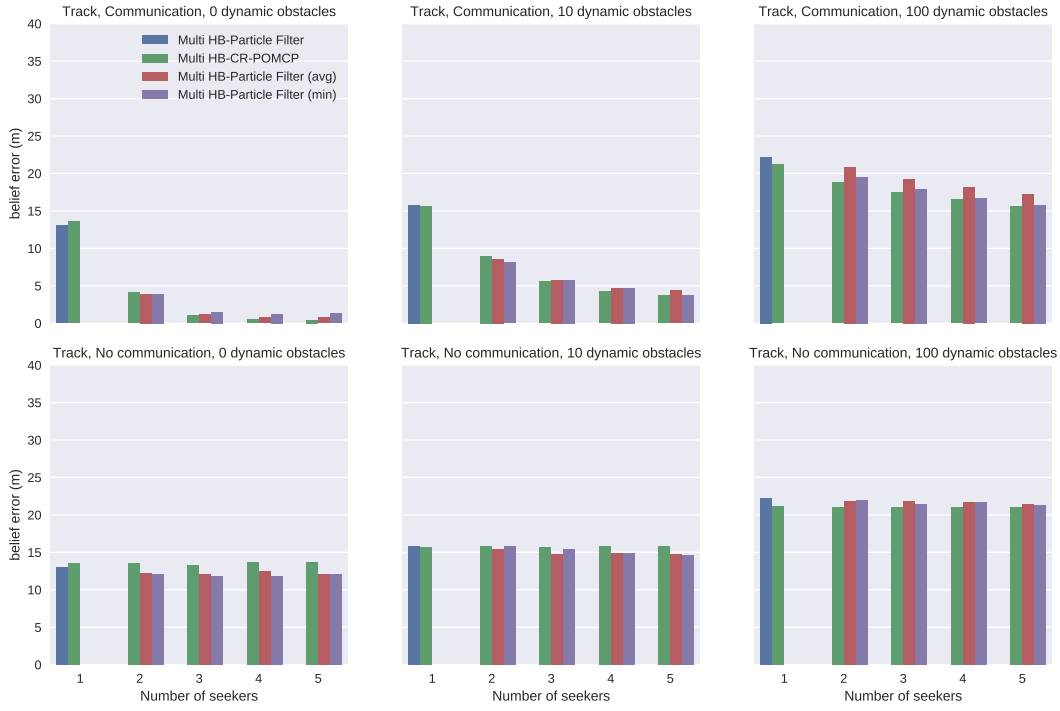


Figure 6.8: The average belief error (using Eq. 5.11) when following. The rows show communication or not, and the columns the number of dynamic obstacles. The See All Follower does not use a belief and is therefore not mentioned.

6.7.1 Multi-agent HB-CR-POMCP Explorer and Multi-agent HB-PF Explorer

These experiments were done during several weeks in the Telecos Square (version 3, like in the simulations; Figure 6.9). Two robots were used, Tibi and Dabo, with the Multi-agent HB-PF Explorer algorithm, since this was found to work best. We only tried the version that used the *minimum* scores when using the observations in the PF update phase.

Like the simulations in subsection 6.6.1, for the *search* behaviour, we measured the time to encounter the person (by the first robot), and for the *track* behaviour, we measured the recovery time and average distance to the person. Since we did not have a ground truth available, we had to use the information obtained through the sensors of the robots and the videos, which show the behaviour of the robots. The first had as

consequence that the distance to the person was only measured when the person was visible.

6.7.1.1 Analysis

Different types of experiments were done: *exploration* without a person, *searching and tracking*, and *tracking* only; they took several weeks of testing and experimenting, from which we obtained a total of about 3 hours of experimental data, and whereby the robots drove each a total distance of about 3 km. A few persons were used during the experiments in which they hid behind one of the obstacles, or just stood out in the open. The robots tried to follow the person at a distance of 1 m, and they always tried to maintain a minimum distance of 1 m to the other robot. The parameters used during the experiments are shown in Table 6.1.

Table 6.2 gives an overview of the different statistics of all the experiments. The distances shown were measured by the robot, i.e. the robot's distances were obtained using the localization positions; the distance that the person walked was also measured by the robot and therefore, is not complete, since the person was not visible the whole time.

The *distance per robot* indicates the total distance covered on average by the robots during the experiments, the *measured dist. person* indicates the distance which was covered by the person, while the robot measured it. The *visibility* indicates the time the person was visible to a robot, the *time connected* indicates the time the robots were exchanging data. The *average distance to the person* is the distance between the robot and the person, measured when the person was visible. The *number of dynamic obstacles* are the average number of people which were visible simultaneously. The *average time found* is the time it took, on average, for a robot to find the person. Finally, the *average recovery time* is the time it took to recover finding the person after having lost it.

In the next subsections we will try to compare the results with the simulations using the *time found* for the *search* experiments, and the *recovery time* and *average distance to the person*. However, in the real experiments the robot sometimes stopped or slowed down (due to obstacles, noisy signals or the low speed), therefore, the comparisons with the simulations should be done with the distance. Since the speed in the simulations was

Table 6.2: Statistics summary of the data recorded during the experiments. The averages (avg) are shown as *average*±*standard deviation*. *Measurements that include the person location were only measured when the person was visible to a robot.

	Exploration	*Search & Track*	*Tracking*	*Total*
Distance per robot [km]	1.2	1.2	0.7	3.2
Measured dist. person* [km]	-	0.4	0.5	0.9
Total time [h]	1.1	1.2	0.9	3.2
Avg. visibility [%]	0	16.3	36.4	15.3
Avg. time connected [%]	95.0	79.5	85.8	86.6
Avg. distance to person* [m]	-	8.4 ± 6.4	8.4 ± 5.6	8.3 ± 5.9
Avg. number dynamic obst.*	2.0 ± 1.5	0.6 ± 1.3	3.9 ± 2.8	1.9 ± 2.2
Avg. time found [s]	-	106.8 ± 138.7	23.5 ± 42.5	72.9 ± 117.6
Avg. distance found [m]	-	69.3 ± 74.0	6.2 ± 13.6	27.3 ± 53.3
Avg. time recovered [s]	-	19.6 ± 39.0	12.0 ± 28.3	15.3 ± 33.6
Avg. distance recovered [m]	-	8.5 ± 12.3	3.3 ± 9.3	3.6 ± 9.5

continuous (0.8 m per discrete time step), we can use it to calculate the distance covered, and thereby comparing it to the *distance found* and *recovery distance*. Nevertheless, we should take into account that we can not do a statistical comparison of the results, since this would require many more experiments.

Figure 6.9 shows two recordings taken during the experiments: the snapshots, the maps with the robot locations, and the belief maps of both robots. The belief map shows that, when the person was detected, the localization was relatively precise (right), but when it was not detected for some time, the location probability is more spread (left). Further information and videos of the experiments can be found on:
http://alex.goldhoorn.net/thesis/st-multi/

Next we will explain the three different kind of experiments done followed by a short discussion.

6.7.1.2 Exploration Only

In these experiments, we wanted to have a look at the search behaviour, and, therefore, no person was present, which can be seen in Table 6.2, because there is no person

distance. An exploration/search phase is shown in the left of Figure 6.9, where none of the robots see the person, and both have a different belief. The experiments showed that the robots clearly explored the whole environment several times, because the probability that the person is in a not visible location, grew again after having passed through it and not having found the person.

6.7.1.3 Search-and-track

In these experiments, a person was present and the robots started not seeing him/her. The robots communicated the observations, and therefore, could update their belief based on them. They also explored in different directions, looking for the person. As soon as one robot saw the person, the other robot also went there. There were also situations where the person was lost, because he went faster than the robot or because one robot temporarily failed; however, the belief of the working robot still helped to recover the person.

The distance covered by the robots until a person was found, was on average 69.3 ± 74.0 m, which is close to the distance covered in simulation, 67.5 ± 66.9 m (see Figure 6.4 for the time with 0–10 dynamic obstacle, which was converted to distance). For the *tracking* part, the *recover distance* was 8.5 ± 12.3 m, which is also close to the simulation's 5.06 ± 7.5 m (converted to distances, see Figure 6.5). The *average distance to the person* shows a low value (8.4 m on average), because only measurements were taken when the person was detected by the robot.

6.7.1.4 Tracking

In the tracking experiments, the robots started with the person visible, and then followed him/her, but due to speed or (dynamic) obstacles they lost the person out of sight temporarily. Nonetheless, the person was found relatively quickly again, because he/she was tracked using the belief.

For some of the *tracking* experiments, the robots had to detect the person first, which took on average 23.5 s, but only 6.2 m was covered because the person was close. The *recover distance* was 3.3 ± 9.3 m, which is also close to the values in simulation (5.06 ± 7.5 m). The *average distance to the person* was a bit higher, because the robot

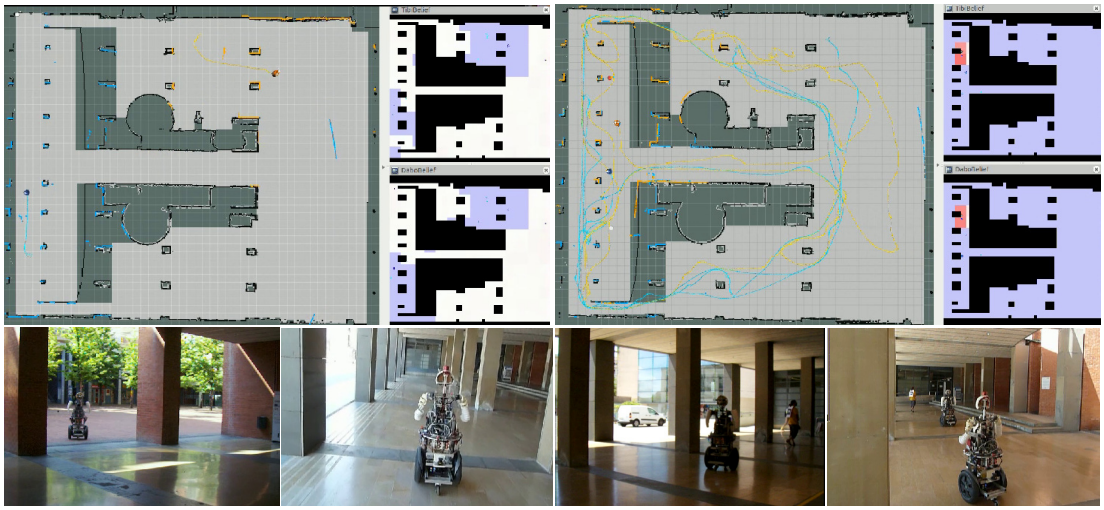


Figure 6.9: Two different scenes during the experiments where the robots search for and find the person. The large maps show the robots (blue and orange) and the trajectories they executed; the red circle indicates that the person was detected at that location, and a white circle means that the person was detected there. The smaller maps represent the beliefs of Tibi (up) and Dabo (down): black squares are obstacles, the blue circles are the robots, and the white to red squares means a low to high probability of the person being there.

was relatively slow, and because having two robots tracking the person, requires them to be at a minimum safe distance.

In the last experiment, the robots searched for the person, which was behind or close to a group of people who occluded him/her, see Figure 6.1. Since there were two robots they had a higher probability of seeing the person, but, when they did not see the person, the belief grew in all directions with a higher probability on areas where the robot probably would not see anything. Here, the dynamic obstacles (small light blue circles in the belief map) were taken into account, and the particles propagated behind static obstacles and dynamic obstacles. Due to the low resolution of the belief map there was also a belief at the location of the other people and the robot. Note that, the low resolution of the map was chosen such that we could group enough particles to create a higher certainty.

6.8 Discussion

The experiments showed that the robots explored the whole environment, thereby taking into account the location of each other. And when tracking, it was also demonstrated that maintaining the belief continuously is important when the person gets out of sight.

Furthermore, the robustness of the multi-agent method was shown in experiments, where one robot suddenly stopped (because of a hardware or software problem). Then, the other robot recovered the person's position, since it had been receiving the person's location until the other robot stopped and it did not receive any information from the other robot. Therefore, using its own belief and observation, it only planned the next goal for itself.

6.8.1 Comparison of Methods

Whereas we use a probabilistic approach to keep track of the probable locations of the person, Hollinger et al. [2010] kept track of a list of *contaminated* nodes, thereby, assuring the person to be found. Their irregular grid maps were converted to graphs, where in each node they assumed full vision, whereas we use a probabilistic vision probability. Next, they did not handle on-line changes in the environment, where we do take into account dynamic obstacles. And finally, they only searched for the person, but we also tracked him/her. In [Hollinger et al., 2015], they focused on data fusion of the beliefs. In our method, we do not send the complete belief, but we send the observations of the agents, and after having locally updated the belief, the most probable locations are sent to the other agents.

Ahmad and Lima [2013] used a method similar to our PF approach to track an object, but we use a fixed observation confidence, whereas they based it on the observation and localization confidence of the sender agent. We however, also share the most probable locations and we do an explicit search of the person by exploring the most probable locations.

The PF method of [Mottaghi and Vaughan, 2007] resembles greatly to our PF method for the belief update method. But, instead of putting a zero weight when a particle is *inconsistent* with the observation, we put a low weight to take into account

that the observation could be a false negative detection. When the tracked person is out of sight, they leave the previous weight, whereas, we put a fixed weight, based on the consistency. While they use a potential field method to search for the target, we make use an exploration method to go through the most probable locations of the target.

Advantage of a centralized PF is that they take into account all observations and no approximations are required, however, DPFs allow for more efficient models, which are more scalable [Hlinka et al., 2013]. Nonetheless, in our real-life problem we are not planning to use a very large amount of agents.

6.8.2 Real-World Issues

Finally, we will discuss some issues with the methods while doing the experiments. First, the robots took the same path several times when they did exploration while this—according to a human point of view—might not be the most efficient way, since taking different paths would allow them to explore more. Our exploration algorithms, however, do not take into account the path, only the goals are optimized, such that the robots choose the closest most probable goal far from other robot's goals. Taking the path into account would require to rewrite the navigation algorithm, moreover, this might be computationally complex when the number of agents is high. A potential solution is given by [Charrow et al., 2013], who tried to optimise for maximal information and therefore, indirectly take the paths into account.

Second, the belief maps of Tibi and Dabo were not always equal, even though they received the same observations—if the communication worked—because there was a random factor in the propagation of the particles, which caused a different spread of the belief. When the robots were without communication, they could only use their own observations and therefore, their beliefs would change. When they recovered the communication they did not send historical information, and although this might be a useful feature, it could be a large amount of information if the amount of seekers is high. This could be tackled by a method like proposed in [Hollinger et al., 2015], where they fused the beliefs by taking a weighted sum of the neighbours' beliefs.

The robots sometimes were not able to drive up or down the ramp due to the narrow passage and the inclined position, which made the horizontal lasers detect the

floor as an object. In some cases, this caused the planner to avoid the ramp and take the detour. In order to cope with ramps, 3D maps and navigation should be used.

6.9 Conclusions

In this chapter, we have presented an unified method for searching and tracking a person using a group of mobile robots in a large continuous urban environment with dynamic obstacles. The observations of the location of the person are obtained from a leg detection algorithm that uses laser sensors and a visual marker detection algorithm that uses a video camera to recognise the person. However, our method does not require a specific sensor type, but it requires a localization of the person or an empty observation—if not visible—as input; moreover, the observations of all the other agents are used. At first, the belief of the person's location is maintained using either the Multi-agent HB-CR-POMCP Explorer or the Multi-agent HB-PF Explorer, then this belief is segmented in a histogram matrix to obtain the locations with the highest probability of the person being there. Thereafter, in the goal decision phase, the agents are either sent directly to the location of the person if he/she was visible, otherwise an exploration is done of the most probable locations taking into account the distance, probability, and the other agent's goals.

Simulations were done in a large urban environment, part of a campus, with up to 100 dynamic obstacles moving around. For *searching*, in most cases, the Multi-agent HB-PF Explorer was fastest in finding the person, and in particular using the *average* weight, when using the observations of all agents. Also communication showed significant improvement for searching. For *tracking* we did not find any significant difference between the methods, neither when using communication. Furthermore, when looking at the tracking distance, the PF methods enabled to get closer, and also having multiple robots communicating reduced the average tracking distance. Finally, the belief of the PF method was found to be closer to the real position.

The real experiments showed consistent results with the simulations and demonstrated it to be a pragmatic method to search and track a person in the real world with two robots. The search behaviour showed an exploration over the field, whereby both robots were coordinating, and the communication between them also showed a

more robust system, for example when one robot failed the other continued tracking the person quickly. The method was also shown to be a robust tracker when several people (dynamic obstacles) obstructed the vision temporarily of the robot, they were able to find the person quickly again.

Chapter 7

Conclusions and Future Work

In this thesis we have presented a search-and-track behaviour for mobile robots to search and to track a person in an urban environment with one method.

We started with a Reinforcement Learning solution to learn the best strategy for the seeker to find the hider, however, we quickly arrived to the computational complexity limitations of the Mixed Observable Markov Decision Process (MOMDP) solvers. A hierarchical model was introduced to reduce the state space greatly, but this did not reduce the policy computation time sufficiently. Next, a Monte-Carlo solution was tried, the POMCP, which we adapted to our problem, to work for continuous state space and in real-time, resulting in the Continuous Real-time POMCP (CR-POMCP). We were now able to work on large sized maps, in continuous space and in real-time, but we saw that the movements of the real robot were not optimal, therefore, we decided to use only the belief. The belief is a probability map of the person's location, which is then used to find the most probable location of the person. And as a next step, to decide where to go to, the belief was used to get the highest probable location, instead of using the policy's action. Since the belief was updated using a Monte-Carlo-based method of the POMCP, we also proposed the use of a PF to represent the belief. A basic PF cannot be used directly, because a PF always needs an observation; therefore, we made a version which does not need an observation. The particles of the PF were used as the belief. Finally, the method was extended for multiple agents using an explorer to distribute different potential locations of the person to the agents.

Summarizing, we have started with a RL method to learn the robot's strategy to

hide-and-seek, but we ended with a distributed cooperative PF-based method with exploration for one or more agents to search-and-track a person. See Table 7.1 for an overview of the different methods and their abilities. All methods were tested extensively in simulations and in real-life experiments, and the final methods—in specific the Multi-agent HB-PF Explorer—are able to **search** and **track** in **large environments**, **continuous state space**, with **dynamic obstacles** and with **multiple agents**.

Table 7.1: This tables shows the different methods proposed in the thesis with the characteristics and references. The second column indicates the method on which it is based; the *Continuous*, *On-line*, *Dyn. obstacles* and *Multi-agents* columns indicate if the methods can handle these characteristics and if they have been tested with them; the *Largest Map* column refers to the largest map size tested; and the last column indicates the section in which the method is explained.

Method	**Based on**	**Continuous**	**On-line**	**Dyn. obstacles**	**Multi-agents**	**Largest Map** (m × m)	**Explained in**
Off-line MOMDP	MOMDP					12 × 12	4.6
Hierarchical MOMDP	MOMDP		✓			12 × 12	4.7
CR-POMCP	POMCP	✓	✓			75 × 69	5.5
HB-CR-POMCP	POMCP	✓	✓			75 × 69	5.6
CR-POMCP Searcher & Tracker	POMCP	✓	✓			75 × 69	5.5,5.6.1
HB-CR-POMCP Searcher & Tracker	POMCP	✓	✓	✓		75 × 69	5.6
HB-PF Searcher & Tracker	PF	✓	✓	✓		75 × 69	5.7
Multi-agent HB-CR-POMCP Explorer	POMCP	✓	✓	✓	✓	75 × 69	6.4.1,6.5
Multi-agent HB-PF Explorer	PF	✓	✓	✓	✓	75 × 69	6.4.2,6.5

7.1 Future Work

The presented methods select a goal for the agent based on the belief, and for the multi-agent version also based on the distance to the HB point and whether another agent already has a search goal close to it. However, it does not take into account the path to the goal. Currently the shortest path is taken to the location with the HB,

but in some cases it might be better to take a slightly larger route to explore more extensively, while going to the HB location. Especially when there are multiple agents, we should avoid them to take the same path. Maximising the entropy can be part of a solution for this problem, as done in [Charrow et al., 2013], or using a potential field method that has an attractive force for particles [Mottaghi and Vaughan, 2007].

For the multi-agent methods, the robots continuously send observations to all other agents, and since we do not have a large amount of robots, this is not a problem. Nevertheless, this could be made more efficient when using for example some Distributed Particle Filter (DPF) methods, as commented in [Hlinka et al., 2013], where information is only sent to neighbours. As a second improvement for the multi-agent methods, the recovery of information could be done after an agent has been disconnected from the network, such that previous observations can be taken into account by all agents, like done in [Hollinger et al., 2015].

For a more pragmatic system, a person recognition system should be used that does not require artificial markers. Several person recognition systems do exist, e.g. the ones used by [Granata and Bidaud, 2012, Linder et al., 2016, Martinson, 2014], but they should be tested outside to see how well they work under different lighting conditions and up to what distance.

Finally, the searching and tracking in an *unknown environment* is another important next step, as already mentioned by [Kulich et al., 2016], who do searching and map exploration at the same time

Bibliography

Abed-alguni, B. H., Chalup, S. K., Henskens, F. A., and Paul, D. J. (2015). A multi-agent cooperative reinforcement learning model using a hierarchy of consultants, tutors and workers. *Vietnam Journal of Computer Science*, 2(4):213–226.

Ahmad, A. and Lima, P. (2013). Multi-robot cooperative spherical-object tracking in 3d space based on particle filters. *Robotics and Autonomous Systems*, 61(10):1084 – 1093. Selected Papers from the 5th European Conference on Mobile Robots (ECMR 2011).

Amor-Martinez, A., Ruiz, A., Moreno-Noguer, F., and Sanfeliu, A. (2014). On-board real-time pose estimation for UAVs using deformable visual contour registration. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2595–2601.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., and Qin, Y. (2004). An integrated theory of the mind. *Psychological review*, 111(4):1036–60.

Araya-Lopez, M., Thomas, V., Buffet, O., and Charpillet, F. (2010). A closer look at MOMDPs. In *2010 22nd IEEE International Conference on Tools with Artificial Intelligence*, volume 2, pages 197–204.

Arras, K. O., Mozos, O. M., and Burgard, W. (2007). Using boosted features for the detection of people in 2D range data. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3402–3407.

Arulampalam, M., Maskell, S., Gordon, N., and Clapp, T. (2002). A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188.

Awheda, M. D. and Schwartz, H. M. (2016). A decentralized fuzzy learning algorithm for pursuit-evasion differential games with superior evaders. *Journal of Intelligent & Robotic Systems*, 83(1):35–53.

Bai, H., Hsu, D., and Lee, W. (2014). Integrated perception and planning in the continuous space: A POMDP approach. *The International Journal of Robotics Research*, 33(9):1288–1302.

Bai, H., Hsu, D., Lee, W. S., and Ngo, V. A. (2011). *Monte Carlo Value Iteration for Continuous-State POMDPs*, pages 175–191. Springer Berlin Heidelberg, Berlin, Heidelberg.

Bandyopadhyay, T., Rong, N., Ang Jr., M., Hsu, D., and Lee, W. (2009). Motion planning for people tracking in uncertain and dynamic environments. In *IEEE International Conference on Robotics & Automation, Workshop on People Detection & Tracking*.

Bellman, R. (1957). A markovian decision process. *Indiana University Mathematics Journal*, 6:679–684.

Bhadauria, D., Klein, K., Isler, V., and Suri, S. (2012). Capturing an evader in polygonal environments with obstacles: The full visibility case. *The International Journal of Robotics Research*, 31(10):1176–1189.

Bhattacharya, S. and Hutchinson, S. (2010). *On the Existence of Nash Equilibrium for a Two Player Pursuit-Evasion Game with Visibility Constraints*, pages 251–265. Springer Berlin Heidelberg, Berlin, Heidelberg.

Bilgin, A. T. and Urtis, E. K. (2015). An approach to multi-agent pursuit evasion games using reinforcement learning. In *International Conference on Advanced Robotics (ICAR)*, pages 164–169. IEEE.

Blackman, S. S. (2004). Multiple hypothesis tracking for multiple target tracking. *IEEE Aerospace and Electronic Systems Magazine*, 19(1):5–18.

Borie, R., Tovey, C., and Koenig, S. (2011). Algorithms and complexity results for graph-based pursuit evasion. *Autonomous Robots*, 31(4):317.

Braziunas, D. (2003). POMDP solution methods. Technical report, University of Toronto.

Brscic, D., Kanda, T., Ikeda, T., and Miyashita, T. (2013). Person tracking in large public spaces using 3-d range sensors. *IEEE Transactions on Human-Machine Systems*, 43(6):522 – 534.

Burgard, W., Moors, M., Stachniss, C., and Schneider, F. E. (2005). Coordinated multi-robot exploration. *IEEE Transactions on Robotics*, 21(3):376–386.

Caley, J. A., Lawrance, N. R. J., and Hollinger, G. A. (2016). Deep learning of structured environments for robot search. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3987–3992.

Capitan, J., Merino, L., and Ollero, A. (2016). Cooperative decision-making under uncertainties for multi-target surveillance with multiples UAVs. *Journal of Intelligent & Robotic Systems*, 84(1):371–386.

Cassandra, A., Kaelbling, L., and Kurien, J. (1996). Acting under uncertainty: discrete bayesian models for mobile-robot navigation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 963–972 vol.2.

Charrow, B., Michael, N., and Kumar, V. (2013). Cooperative multi-robot estimation and control for radio source localization. In Desai, P. J., Dudek, G., Khatib, O., and Kumar, V., editors, *Experimental Robotics: The 13th International Symposium on Experimental Robotics*, pages 337–351, Heidelberg. Springer International Publishing.

Cheong, A., Lau, M., Foo, E., Hedley, J., and Bo, J. W. (2016). Development of a robotic waiter system. *IFAC-PapersOnLine*, 49(21):681 – 686.

Choi, W., Pantofaru, C., and Savarese, S. (2011). Detecting and tracking people using an RGB-D camera via multiple detector fusion. In *Proceedings of the IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 1076–1083.

Chu, H. N., Glad, A., Simonin, O., Sempe, F., Drogoul, A., and Charpillet, F. (2007). Swarm approaches for the patrolling problem, information propagation vs. pheromone evaporation. In *IEEE International Conference on Tools with Artificial Intelligence(ICTAI)*, volume 1, pages 442–449.

Chung, T., Hollinger, G., and Isler, V. (2011). Search and pursuit-evasion in mobile robotics. *Autonomous Robots*, 31(4):299–316.

Doroodgar, B., Ficocelli, M., Mobedi, B., and Nejat, G. (2010). The search for survivors: Cooperative human-robot interaction in search and rescue environments using semi-autonomous robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2858–2863.

Ferrein, A. and Steinbauer, G. (2016). 20 years of robocup. *KI - Künstliche Intelligenz*, 30(3):225–232.

Fleuret, F., Berclaz, J., Lengagne, R., and Fua, P. (2008). Multicamera people tracking with a probabilistic occupancy map. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):267–282.

Foderaro, G., Swingler, A., and Ferrari, S. (2016). A model-based approach to optimizing ms. pac-man game strategies in real time. *IEEE Transactions on Computational Intelligence and AI in Games*, PP(99):1–1.

Foka, A. and Trahanias, P. (2007). Real-time hierarchical POMDPs for autonomous robot navigation. *Robotics and Autonomous Systems*, 55(7):561–571.

Fomin, F. V. and Thilikos, D. M. (2008). An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236 – 245. Graph Searching.

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

Garrell, A., Villamizar, M., Moreno-Noguer, F., and Sanfeliu, A. (2013). Proactive behavior of an autonomous mobile robot for human-assisted learning. In *Proceedings of IEEE RO-MAN*, pages 107–113.

Georgaraki, C. (2012). A POMDP approach to the hide and seek game. Master's thesis, Universitat Politècnica de Catalunya, Barcelona, Spain.

Gerkey, B. P. and Konolige, K. (2008). Planning and control in unstructured terrain. In *In Workshop on Path Planning on Costmaps, Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Gerkey, B. P., Thrun, S., and Gordon, G. (2005). *Parallel Stochastic Hill- Climbing with Small Teams*, pages 65–77. Springer Netherlands, Dordrecht.

Glas, D. F., Miyashita, T., Ishiguro, H., and Hagita, N. (2009). Laser-based tracking of human position and orientation using parametric shape modeling. *Advanced Robotics*, 23(4):405–428.

Glas, D. F., Morales, Y., Kanda, T., Ishiguro, H., and Hagita, N. (2015). Simultaneous people tracking and robot localization in dynamic social spaces. *Autonomous Robots*, 39(1):43–63.

Goldhoorn, A., Garrell, A., Alquézar, R., and Sanfeliu, A. (2014). Continuous real time pomcp to find-and-follow people by a humanoid service robot. In *Proceedings of the IEEE-RAS International Conference on Humanoid Robots*, pages 741–747.

Goldhoorn, A., Garrell, A., Alquézar, R., and Sanfeliu, A. (2016). Un nuevo método cooperativo para encontrar personas en un entorno urbano con robots móviles. In *XXXVII Jornadas de Automática*, pages 206–213, Madrid, Spain.

Goldhoorn, A., Garrell, A., Alquézar, R., and Sanfeliu, A. (2017a). Searching and tracking people with cooperative mobile robots. *Autonomous Robots*.

Goldhoorn, A., Garrell, A., Alquézar, R., and Sanfeliu, A. (2017b). Searching and tracking people in urban environments with static and dynamic obstacles. *Robotics and Autonomous Systems*, 98(Supplement C):147–157.

Goldhoorn, A., Sanfeliu, A., and Alquézar, R. (2013a). Analysis of methods for playing human robot hide-and-seek in a simple real world urban environment. In *ROBOT (2)*, volume 253 of *Advances in Intelligent Systems and Computing*, pages 505–520. Springer.

Goldhoorn, A., Sanfeliu, A., and Alquézar, R. (2013b). Comparison of MOMDP and heuristic methods to play hide-and-seek. In Gibert, K., Botti, V. J., and Bolaño, R. R., editors, *CCIA*, volume 256 of *Frontiers in Artificial Intelligence and Applications*, pages 31–40. IOS Press.

Goodrich, M. A., Morse, B. S., Gerhardt, D., Cooper, J. L., Quigley, M., Adams, J. A., and Humphrey, C. (2008). Supporting wilderness search and rescue using a camera-equipped mini uav. *Journal of Field Robotics*, 25(1-2):89–110.

Granata, C. and Bidaud, P. (2012). A framework for the design of person following behaviours for social mobile robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4652–4659.

Grisetti, G., Stachniss, C., and Burgard, W. (2007). Improved techniques for grid mapping with rao-blackwellized particle filters. *Journal IEEE Transactions on Robotics*, 23(1):34–46.

Gu, D. (2007). Distributed particle filter for target tracking. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3856–3861. IEEE.

Hauskrecht, M. (2000). Value-function approximations for partially observable markov decision processes. *Journal of Artificial Intelligence Research*, 13:33–94.

Hlinka, O., Hlawatsch, F., and Djuric, P. M. (2013). Distributed particle filtering in agent networks: A survey, classification, and comparison. *IEEE Signal Processing Magazine*, 30:61–81.

Hollinger, G., Yerramalli, S., Singh, S., Mitra, U., and Sukhatme, G. (2015). Distributed data fusion for multirobot search. *IEEE Transactions on Robotics*, 31(1):55–66.

Hollinger, G. A., Singh, S., and Kehagias, A. (2010). Improving the efficiency of clearing with multi-agent teams. *International Journal of Robotics Research*, 29(8):1088–1105.

Iocchi, L., Nardi, D., Piaggio, M., and Sgorbissa, A. (2003). Distributed coordination in heterogeneous multi-robot systems. *Autonomous Robots*, 15(2):155–168.

Jiang, B. and Ravindran, B. (2011). Completely distributed particle filters for target tracking in sensor networks. In *Proceedings of the 25th IEEE International Symposium on Parallel and Distributed Processing*, pages 334–344. IEEE.

Johansson, E. and Balkenius, C. (2005). It's a child's game: Investigating cognitive development with playing robots. In *Proceedings of the 4th International Conference on Development and Learning*, pages 164–164.

Kanda, T., Sato, R., Saiwaki, N., and Ishiguro, H. (2007). A two-month field trial in an elementary school for long-term human–robot interaction. *IEEE Transactions on Robotics*, 23(5):962–971.

Katsilieris, F., Lindhé, M., Dimarogonas, D. V., Ögren, P., and Johansson, K. H. (2013). Demonstration of multi-robot search and secure. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) workshop on search and pursuit/evasion.*

Kennedy, W. G., Bugajska, M. D., Marge, M., Adams, W., Fransen, B. R., Perzanowski, D., Schultz, A. C., and Trafton, J. G. (2007). Spatial Representation and Reasoning for Human-Robot Collaboration. *Artificial Intelligence*, pages 1554–1559.

Khan, Z., Balch, T., and Dellaert, F. (2005). Mcmc-based particle filtering for tracking a variable number of interacting targets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(11):1805–1918.

Kocsis, L. and Szepesvári, C. (2006). Bandit based monte-carlo planning. In *Proceedings of the European Conference on Machine Learning*, ECML'06, pages 282–293, Berlin, Heidelberg. Springer-Verlag.

Kolling, A. and Carpin, S. (2010). Multi-robot pursuit-evasion without maps. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3045–3051.

Kolling, A., Kleiner, A., Lewis, M., and Sycara, K. (2011). Computing and executing strategies for moving target search. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 4246–4253.

Koopman, B. (1946). *Search and screening, Operations Evaluation Group Report No. 56.* Rosslyn, Virginia.

Kratzke, T., Stone, L., and Frost, J. (2010). Search and rescue optimal planning system. In *Proceedings of 13th Conference on Information Fusion (FUSION)*, pages 1–8.

Kulich, M., Miranda-Bront, J. J., and Přeučil, L. (2016). A meta-heuristic based goal-selection strategy for mobile robot search in an unknown environment. *Computers & Operations Research*, pages –.

Kurniawati, H., Hsu, D., and Lee, W. (2008). SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proceedings of Robotics: Science and Systems IV*, Zurich, Switzerland.

Lau, H., Huang, S., and Dissanayake, G. (2006). Probabilistic search for a moving target in an indoor environment. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3393–3398.

LaValle, S. M., Lin, D., Guibas, L. J., Latombe, J. C., and Motwani, R. (1997). Finding an unpredictable target in a workspace with obstacles. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 737–742 vol.1.

Lian, F.-L., Chen, C.-L., and Chou, C.-C. (2015). Tracking and following algorithms for mobile robots for service activities in dynamic environments. *International Journal of Automation and Smart Technology*, 5(1):49–60.

Lim, Z., Hsu, D., and Lee, W. (2011). Monte Carlo Value Iteration with Macro-Actions. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pages 1287–1295.

Linder, T., Breuers, S., Leibe, B., and Arras, K. O. (2016). On multi-modal people tracking from mobile platforms in very crowded and dynamic environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 5512–5519.

Liu, F. and Zeng, G. (2006). Multi-agent cooperative learning research based on reinforcement learning. In *Proceedings of the 10th International Conference on Computer Supported Cooperative Work in Design*, pages 1–6. IEEE.

Luber, M., Sinello, L., and Arras, K. (2011). People tracking in RGB-D data with on-line boosted target models. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3844–3849.

Marconi, L., Melchiorri, C., Beetz, M., Pangercic, D., Siegwart, R., Leutenegger, S., Carloni, R., Stramigioli, S., Bruyninckx, H., Doherty, P., Kleiner, A., Lippiello, V., Finzi, A., Siciliano, B., Sala, A., and Tomatis, N. (2012). The SHERPA project: Smart collaboration between humans and ground-aerial robots for improving rescuing activities in alpine environments. In *Proceedings of the IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 1–4.

Martinson, E. (2014). Detecting occluded people for robotic guidance. In *The 23rd IEEE International Symposium on Robot and Human Interactive Communication*, pages 744–749.

Micire, M. (2008). Evolution and field performance of a rescue robot. *Journal of Field Robotics*, 25(1-2):17–30.

Mitsunaga, N., Smith, C., Kanda, T., Ishiguro, H., and Hagita, N. (2008). Adapting robot behavior for human–robot interaction. *IEEE Transactions on Robotics*, 24(4):911–916.

Montemerlo, M., Thrun, S., and Whittaker, W. (2002). Conditional particle filters for simultaneous mobile robot localization and people-tracking. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 695–701, Washington, DC. IEEE.

Mottaghi, R. and Vaughan, R. (2007). An integrated particle filter and potential field method applied to cooperative multi-robot target tracking. *Autonomous Robots*, 23(1):19–35.

Murphy, R. (2003). Urban search and rescue as a domain for studying human-robot interaction. *IEEE Transaction on Systems, Man and Cybernetics*.

Nijssen, P. and Winands, M. (2012). Monte carlo tree search for the hide-and-seek game scotland yard. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(4):282–294.

Ong, S. C. W., Png, S. W., Hsu, D., and Lee, W. S. (2010). Planning under Uncertainty for Robotic Tasks with Mixed Observability. *The International Journal of Robotics Research*, 29(8):1053–1068.

Otte, M., Kuhlman, M., and Sofge, D. (2016). Competitive two team target search game with communication symmetry and asymmetry. In *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, San Francisco, USA.

Oyama, T., Yoshida, E., Kobayashi, Y., and Kuno, Y. (2013). Tracking visitors with sensor poles for robot's museum guide tour. In *Proceedings of the 6th International Conference on Human System Interactions (HSI)*, pages 645–650. IEEE.

Papadimitriou, C. and Tsisiklis, J. (1987). The complexity of markov decision processes. *Mathematics of Operations Research*, 12(3):441–450.

Pineau, J., Gordon, G., and Thrun, S. (2003). Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 477–484.

Porta, J., Vlassis, N., Spaan, M., and Poupart, P. (2006). Point-based value iteration for continuous POMDPs. *The Journal of Machine Learning Research*, 7:2329–2367.

Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*.

Read, J., Achutegui, K., and Míguez, J. (2014). A distributed particle filter for nonlinear tracking in wireless sensor networks. *Signal Processing*, 98:121–134.

Robin, C. and Lacroix, S. (2016). Multi-robot target detection and tracking: taxonomy and survey. *Autonomous Robots*, 40(4):729–760.

Ross, S., Pineau, J., Paquet, S., and Chaib-Draa, B. (2008). Online Planning Algorithms for POMDPs. *Journal of Artificial Intelligence Research*, 32(2):663–704.

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., and Spaan, M. T. J. (2010). Decentralized Sensor Fusion for Ubiquitous Networking Robotics in Urban Areas. *Sensors*, 10(3):2274–2314.

Santos, J. and Lima, P. (2010). *Multi-robot Cooperative Object Localization*, pages 332–343. Springer Berlin Heidelberg, Berlin, Heidelberg.

Satake, S., Kanda, T., Glas, D., Imai, M., Ishiguro, H., and Hagita, N. (2013). A robot that approaches pedestrians. *IEEE Transactions on Robotics*, 29(2):508–524.

Sheh, R., Schwertfeger, S., and Visser, A. (2016). 16 years of robocup rescue. *KI - Künstliche Intelligenz*, 30(3):267–277.

Sheng, X., Hu, Y.-H., and Ramanathan, P. (2005). Distributed particle filter with GMM approximation for multiple targets localization and tracking in wireless sensor network. In *Proceedings of the 4th international symposium on Information processing in sensor networks (IPSN)*, pages 181–188, Piscataway, NJ, USA. IEEE Press.

Shu, G., Dehghan, A., Oreifej, O., Hand, E., and Shah, M. (2012). Part-based multiple-person tracking with partial occlusion handling. In *CVPR*, pages 1815–1821. IEEE Computer Society.

Silver, D. and Veness, J. (2010). Monte-Carlo planning in large POMDPs. *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pages 1–9.

Spaan, M. T. J. and Vlassis, N. (2004). A point-based POMDP algorithm for robot planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2399–2404, New Orleans, Louisiana.

Stein, P., Santos, V., Spalanzani, A., and Laugier, C. (2013). Navigating in populated environments by following a leader. In *Proceedings of IEEE RO-MAN*, pages 527–532.

Stockman, G. and Shapiro, L. G. (2001). *Computer Vision*. Prentice Hall, Upper Saddle River, NJ, USA, 1st edition.

Stone, L. D. (1975). *Theory of optimal search*. Academic Press, New York.

Stone, L. D. (1977). Search theory: A mathematical theory for finding lost objects. *Mathematics Magazine*, 50(5):pp. 248–256.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press.

Tipaldi, G. D. and Arras, K. O. (2011). I want my coffee hot! learning to find people under spatio-temporal constraints. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1217–1222.

Trafton, J. G., Schultz, A. C., Perznowski, D., Bugajska, M. D., Adams, W., Cassimatis, N. L., and Brock, D. P. (2006). Children and robots learning to play hide and seek. *Proceeding of the 1st ACM SIGCHI/SIGART Conference on Human-Robot Interaction*, pages 242–249.

Trulls, E., Corominas Murtra, A., Pérez-Ibarz, J., Ferrer, G., Vasquez, D., Mirats-Tur, J. M., and Sanfeliu, A. (2011). Autonomous navigation for mobile service robots in urban pedestrian environments. *Journal of Field Robotics*, 28(3):329–354.

Uyttendaele, P. (2009). Hide and seek games. Master's thesis, Maastricht University, The Netherlands.

Van Den Berg, J., Patil, S., and Alterovitz, R. (2012). Efficient approximate value iteration for continuous gaussian POMDPs. In *Proceedings of AAAI Conference on Artificial Intelligence*, AAAI'12, pages 1832–1838. AAAI Press.

Vidal, R., Shakernia, O., Kim, H. J., Shim, D. H., and Sastry, S. (2002). Probabilistic pursuit-evasion games: theory, implementation, and experimental evaluation. *IEEE Transactions on Robotics and Automation*, 18(5):662–669.

Vieira, M. A. M., Govindan, R., and Sukhatme, G. S. (2009). Scalable and practical pursuit-evasion. In *2009 Second International Conference on Robot Communication and Coordination*, pages 1–6.

Villamizar, M., Garrell, A., Sanfeliu, A., and Moreno-Noguer, F. (2012). Online human-assisted learning using random ferns. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 2821–2824.

Volkhardt, M. and Gross, H. M. (2013). Finding people in apartments with a mobile robot. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 4348–4353.

Vázquez, M. A. and Míguez, J. (2017). A robust scheme for distributed particle filtering in wireless sensors networks. *Signal Processing*, 131:190 – 201.

Wasson, G., Ferrer, G., and Martin, W. (1997). Hide-and-seek: effective use of memory in perception/action systems. In *Proceedings of the First International Conference on Autonomous Agents*, pages 492–493.

Weigel, T., Gutmann, J.-S., Dietl, M., Kleiner, A., and Nebel, B. (2002). Cs freiburg: coordinating robots for successful soccer playing. *IEEE Transactions on Robotics and Automation*, 18(5):685–699.

White, C. C. (1991). A survey of solution techniques for the partially observed markov decision process. *Annals of Operations Research*, 32(1):215–230.

Wong, E.-M., Bourgault, F., and Furukawa, T. (2005). Multi-vehicle bayesian search for multiple lost targets. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3169–3174.

Xu, Z., Fitch, R., and Sukkarieh, S. (2013). Decentralised coordination of mobile robots for target tracking with learnt utility models. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2014–2020. IEEE.

Zinn, M., Khatib, O., Roth, B., and Salisbury, J. K. (2004). Playing it safe [human-friendly robots]. *IEEE Robotics & Automation Magazine*, 11(2):12–21.