Collision Avoidance in Uncertain Environments for Autonomous Vehicles using POMDPs

Kollisionsvermeidung in unsicheren Umgebungen für Autonome Fahrzeuge mit POMDPs

Bachelor-Thesis von Albert Schotschneider aus Würzburg

März 2018

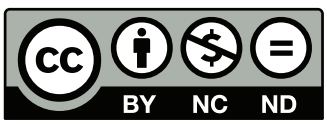
TECHNISCHE
UNIVERSITÄT
DARMSTADT

Collision Avoidance in Uncertain Environments for Autonomous Vehicles using POMDPs
Kollisionsvermeidung in unsicheren Umgebungen für Autonome Fahrzeuge mit POMDPs

Vorgelegte Bachelor-Thesis von Albert Schotschneider aus Würzburg

1. Gutachten: Prof. Jan Peters, Ph.D.
2. Gutachten: Dr. Joni Pajarinen
3. Gutachten: Dorothea Koert, M.Sc.

Tag der Einreichung:



Erklärung zur Abschlussarbeit gemäß §23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Albert Schotschneider, die vorliegende Bachelor-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Datum / Date: Unterschrift / Signature:

_____ _____

Abstract

Collision avoidance is a challenging task for autonomous vehicles, in particular under partial observability. As a specific case, it is desirable to predict and avoid collisions with pedestrians. In this context, the autonomous vehicle should be able to predict changes in the behavior of pedestrians and plan its actions accordingly. A crucial information to avoid colliding with pedestrians is the intention of the pedestrians. Usually, this intention is not known a priori to the autonomous vehicle and is only accessible through partial observations. In this kind of setting a partially observable Markov decision process (POMDP) can be applied, as it takes partial observations and uncertainty into account during planning. This thesis proposes an extension to an existing POMDP model for collision avoidance with pedestrians, by additionally modeling the type of a pedestrian in terms of their trustworthiness. Therefore, we model the intention of the pedestrian as his destination and his next movement towards it, as well as his variance along the path. We use the Determinized Sparse Partially Observable Tree (DESPOT) algorithm to solve this POMDP. The proposed model is evaluated in simulation and we present first results for the combined modeling of the belief over goals and trustworthiness of a pedestrian. The experiments demonstrate that the car slows down ahead of time and avoids colliding with the pedestrian.

Zusammenfassung

Kollisionsvermeidung ist eine Herausforderung für autonome Fahrzeuge, insbesondere bei Teilbeobachtbarkeit. In einem speziellen Fall ist es wünschenswert, Kollisionen mit Fußgängern vorherzusagen und zu vermeiden. In diesem Zusammenhang sollte das autonome Fahrzeug in der Lage sein, Änderungen im Verhalten von Fußgängern vorherzusagen und seine Handlungen entsprechend zu planen. Eine wichtige Information, um eine Kollision mit Fußgängern zu vermeiden, ist die Intention der Fußgänger. In der Regel ist diese Intention dem autonomen Fahrzeug a priori nicht bekannt und nur durch Teilbeobachtungen zugänglich. In diesem Umfeld kann ein partiell beobachtbarer Markov-Entscheidungsprozess (POMDP) angewendet werden, da Teilbeobachtungen und Unsicherheiten bei der Planung berücksichtigt werden. Diese Arbeit schlägt eine Erweiterung eines bestehenden POMDP-Modells zur Kollisionsvermeidung mit Fußgängern vor, indem zusätzlich der Typ eines Fußgängers in Bezug auf seine Vertrauenswürdigkeit modelliert wird. Daher modellieren wir die Intention des Fußgängers als sein Ziel und seine nächste Bewegung, ebenso wie seine Abweichung auf dem Weg. Wir verwenden den DESPOT-Algorithmus (Determinized Sparse Partically Observable Tree), um diesen POMDP zu lösen. Das vorgeschlagene Modell wird in einer Simulation evaluiert und wir präsentieren erste Ergebnisse für die kombinierte Modellierung der Überzeugung über Ziele und Vertrauenswürdigkeit eines Fußgängers. Die Experimente zeigen, dass das Auto im Voraus langsamer wird und eine Kollision mit dem Fußgänger vermeidet.

Acknowledgments

During my work, I came along with great people who helped me accomplish this thesis and motivated me. Therefore, I would like to take the opportunity and express my thanks to them.
First, I would like to thank Prof. Jan Peters, head of the Intelligent Autonomous Systems (IAS) group, for giving me the opportunity to work on such an interesting topic, and also introduced me to my supervisors Dr. Joni Pajarinen and Dorothea Koert, M.Sc. I also greatly thank them for being always available and patiently helped me solving many problems. Their guidance helped me moving forward.
Furthermore, I would like to thank my parents and friends, who pushed me forward and motivated me throughout this time.

Contents

Figures, Tables and Algorithms

List of Figures

List of Tables

List of Algorithms

Abbreviations, Symbols and Operators

List of Abbreviations

Notation	Description
DESPOT	Determinized Sparse Partially Observable Trees
MC-RTBSS	Monte Carlo Real-Time Belief Space Search
MCVI	Monte Carlo Value Iteration
MDP	Markov Decision Process
MiGS	Milestone Guided Sampling
MOMDP	Mixed Observable Markov Decision Process
PGI	Policy Graph Improvement
POMDP	Partially Observable Markov Decision Process
ROS	Robot Operating System
SARSOP	Successive Apporximation of the Reachable Space under Optimal Policies

List of Symbols

Notation	Description
$\mathcal{A}$	The action space of the agent in an MDP.
$\mathcal{B}$	The belief space of the POMDP.
γ	The discount factor of an MDP.
d	The granularity value for dicretizing coordinate, velocity and orientation values.
g	A predefined goal g (vectorized notation $\boldsymbol{g}$) from the set of goals G.
G	The set of predefined goals.

H	The finite time horizon.
$\mathcal{O}$	The observation space of the agent of a POMDP.
R_π	The expected discounted sum of reward under a policy π.
$\mathcal{S}$	The state space of an MDP.
θ_c	Orientation of the car in radians.
t	A specific type t from the type space $\mathcal{T}$.
$\mathcal{T}$	The set of types of the pedestrian.
v_c	The lateral velocity of the car.
v_p	The velocity of the pedestrian.
x_c	The x position of the car in the environment.
x_p	The x position of the pedestrian in the environment.
y_c	The y position of the car in the environment.
y_p	The y position of the pedestrian in the environment.

List of Operators

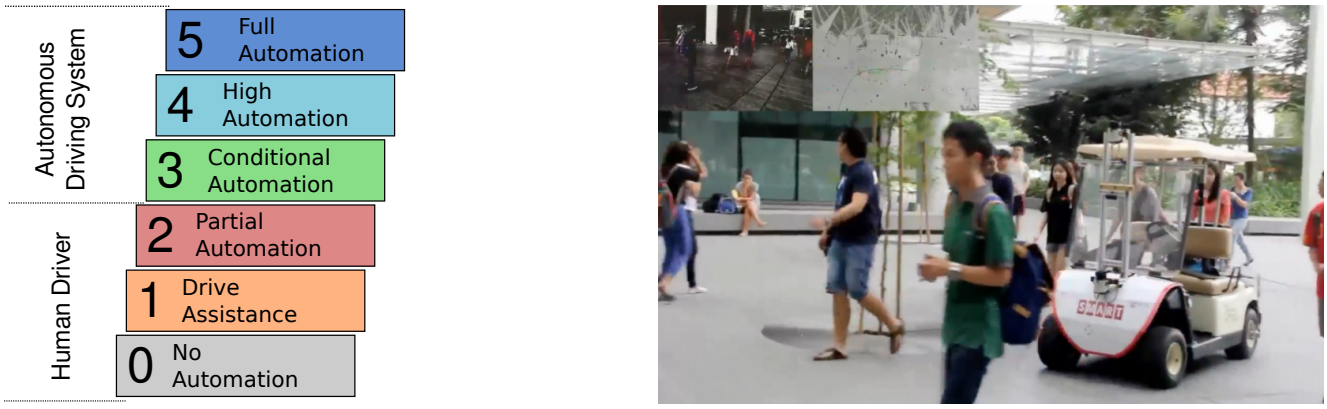
Notation	Description	Operator
b_t	The belief of a state s at time t.	$b_t(s)$
b_0	The initial belief of a state s.	$b_0(s)$
O	The observation probability function of a POMDP given action a, state s' and observation o.	$O(s', a, o)$
π	The policy function for a given state s and action a.	$\pi(s, a)$
π^*	The optimal policy function for a given state s and action a.	$\pi^*(s, a)$
r	The reward function for a given state s and action a.	$r(s, a)$

T	The state transition probability function given states s and s' and action a.	$T(s, a, s')$
V^{π}	The value function for a given state s under policy π.	$V^{\pi}(s)$
V^{*}	The optimal value function for a given state s.	$V^{*}(s)$
V^{π}	The value function for a given belief state b under policy π.	$V^{\pi}(b)$
V^{*}	The optimal value function for a given belief state b.	$V^{*}(b)$

1 Introduction

1.1 Motivation

Intelligent technical systems are becoming more important in our everyday lives. Robots are present in a variety of fields, ranging from industrial robots manufacturing and assembling important parts, to service or home robots, helping us doing different tasks and facilitating our work. However, autonomous agents might also appear in different shapes. In particular, in autonomous vehicles, intelligent agents can potentially increase driving more safely and also autonomously. There are a lot of different assistance systems in autonomous vehicles, e.g., parking assistants which can detect a parking spot and autonomously approach the spot and park the car, or adaptive cruise control for adjusting the speed of the vehicle to the car in front of it. The next level of autonomy of the vehicle would be driving fully autonomously and independently from the human driver. This level of autonomy can be split further. In general, there are 6 levels of autonomous driving, ranging from level 0 to level 5. Level 0 means no autonomous driving at all, whereas level 5 is full autonomous driving, which requires no human interaction with the car to drive [Smith, 2013]. Figure 1.1a shows these levels. In level 0, the human driver takes full control over the steering and throttling of the velocity of the car. In level 1, the vehicle can control the steering or the acceleration and deceleration actions exclusively by incorporating the information from the environment. Taking control over both steering and throttling of the velocity is possible in level 2, where multiple assistance systems work in parallel. In level 3, the system additionally monitors the environment. The human driver does not have to actively perform the driving task but has to be ready to take over control of the car when the assistance systems need to fallback. In high automation driving, the car can handle the full driving task by itself, even when the driver does not respond to a request of the assistant systems. In level 5, the car can manage the driving task under all conditions and environments, which also could be handled by a human driver.



(a) (b)

Figure 1.1: Figure (a) shows the 6 levels of autonomous driving in ascending order from no automation to full automation [Smith, 2013]. These levels allow to distinguish cars in their level of autonomy depending of properties, such as lane detection or holding a safe distance to the vehicle in front. Figure (b) shows the golf cart used as an autonomous vehicle in the work of [Bai et al., 2015]. It drives completely driver-less in a plaza full of people, avoiding collisions. In particular under the presence of partial observations collision avoidance of pedestrians becomes a challenging problem. Bai et al. suggest the use of Partially Observable Markov Decision Processes (POMDPs) to handle this problem.

The question now arises, at which level of autonomous driving we are right now. There are many examples for each level, but a specific answer can't be given. For the full automation level, there only exist concepts of cars driving at that level of automation, like the concept from Volkswagen called *Volkswagen SEDRIC*[1] (abbreviation for

[1] http://www.discover-sedric.com/de/

SElf-DRIving Car) for the car of the future. Level 4 automation is already been made possible by Waymo[2] with their autonomous vehicle. The first consumer-ready car at conditional automation is the Audi A8. This technology is called *Audi AI traffic jam pilot*[3] and can take full control over the driving task. Besides driving autonomously, the car has to adapt to environmental changes, e.g., other cars accidentally driving into the path of the agent vehicle. The safety of the human driver and other traffic participants have to be ensured by important assistance systems.

One particular and crucial assistance system for the safety of the human driver and also other traffic participants is avoiding collisions with static and dynamic obstacles. Due to uncertain environments and partial observations, the autonomous vehicle has to plan ahead of time and make decisions under uncertainty. Therefore, collision avoidance has received much attention in the past years due to its importance, not only in the field of autonomous vehicle driving, but also in various fields of applications, such as [Bai et al., 2012; Bandyopadhyay et al., 2013; Bai et al., 2015]. Collision avoiding with static objects can be done relatively straight forward as these objects don't interfere with their environment by actions. Additionally, avoiding collisions with dynamic objects is crucial as the underlying dynamic process of the objects' intention is not known to the vehicle at all. The agent can compute the movement behavior of dynamic objects based on observations or known movement processes, e.g., for other vehicles. But relying on this information solely can lead to unwanted behavior. The car could detect a possible collision too late. With additional information about the belief of the dynamic object, collisions can be predicted earlier and actions can be taken respectively. However, the vehicle has to make decisions under uncertainty over the objects' intention, which can be expressed in several ways. An intention could be the next position of the traffic participant, or the headed goal it is approaching. In this work, the autonomous vehicle will maintain the intention of a pedestrian as its headed goal from a set of predefined goals. In order to model this problem, a POMDP is used as it can model uncertainty through observations and maintaining a belief of the system state.

1.2 Structure of this Thesis

This thesis is structured in the following chapters:

Chapter 2 provides fundamental information about Markov Decision Processes and Partially Observable Markov Decision Processes. In addition, solving methods for these mathematical tools will be discussed.

Chapter 3 introduces a POMDP model that can incorporate both, the current goal of pedestrians and their type of trustworthiness in account while planning actions for an autonomous vehicle.

Chapter 4 presents details of the implementation of the POMDP model and provides an overview over the DESPOT framework, as well as the complete setup of the simulation environment.

Chapter 5 presents several experimental setups under different preconditions.

Chapter 6 resumes the results and opens a discussion for future work in the field of collision avoidance for autonomous vehicles.

[2] https://waymo.com/

[3] https://www.audi-technology-portal.de/en/electrics-electronics/driver-assistant-systems/audi-a8-audi-ai-traffic-jam-pilot

2 Fundamentals and Related Work

In this chapter, the underlying concept of Markov Decision Processes (MDPs) [Bellman, 1957a] and Partially Observable Markov Decision Processes (POMDPs) [Kaelbling et al., 1998] will be formally introduced. These mathematical tools are used to model and solve stochastic and dynamic decision making, both in fully and partially observable environments. Furthermore, methods for solving problems with uncertainty in their states will be shown. Lastly, the solving algorithm for the POMDP model used in this work will be explained in detail.

2.1 Markov Decision Process (MDP)

A Markov Decision Process is defined by a tuple $(\mathcal{S}, \mathcal{A}, T, r, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are finite sets of states and actions respectively, $T(s,a,s') = \Pr(s' \mid s,a)$ a state transition probability function for a given state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$, $r(s,a)$ a reward function for taking an action a in state s, and γ a discount factor between 0 and 1 exclusively. Generally, the agent takes action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$, leaving him in a subsequent state s', which is computed by the state transition probability function T. Concurrently, the agent receives a reward based on the state s and the action a taken in that state. In order to use this model of decision making, the environment has to follow the *Markov Property*. Formally, it is defined by

$$\Pr(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1,\dots}) = \Pr(s_{t+1} \mid s_t, a_t).$$

Intuitively, the Markov Property states that the next state s_{t+1} does only depend on the current state s_t [Markov, 1954]. The actual generation of an action a is done by a policy π, which maps from states to actions. This mapping can be defined by a deterministic or stochastic function, where a deterministic policy will generate a single action given a state, and a stochastic policy will generate a number of actions the agent can take by a given probability distribution.

The goal of an MDP is to find a policy π that maximizes the expected return over time. We distinct between finite horizon problems, where the agent stops after H time steps, and infinite horizon problems, where the agent continuously takes actions and receives rewards for ever. We can define the *expected discounted sum of rewards* for a finite horizon H as

$$R_\pi = \mathbb{E}_\pi\left[\sum_{t=0}^{H-1} \gamma^t r_t(s_t, a_t) + r_H(s_H)\right]$$

with $r_H(s_H)$ being the final reward at time step H. The *expected discounted sum of rewards* for infinite horizon problems then is defined as

$$R_\pi = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right].$$

The optimal policy π^* is the policy that maximizes the expected discounted reward over time, thus

$$\pi^* = \arg\max_\pi R_\pi. \tag{2.1}$$

We denote the value of a state by the value function $V^\pi(s)$ under policy π, which is also known as the Bellman Equation [Bellman, 1957b]. This function describes how *good* it is to be in the given state s when following policy π by calculating the expected cumulative reward. Formally, the Bellman Equation is defined as

$$V^\pi(s) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^\pi(s') \tag{2.2}$$

For the optimal policy from equation (2.1) the optimal value function is given by

$$V^*(s) = \max_a \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^*(s') \right] \tag{2.3}$$

which is also known as the Bellman optimality equation [Bellman, 1957b]. To get the optimal value function and solve MDPs in general, we can apply the Value Iteration algorithm [Bellman, 1957a], which converges to the optimal Value Function $V^*(s)$ from equation (2.3) on the facing page by initializing the value for all states to 0 and recursively applying the value function (2.2) until the it changes only by a small amount δ. The pseudo code can be seen in algorithm 1. Another algorithm for solving MDPs is called Policy Iteration [Howard, 1960]. The first step is to evaluate the policy by calculating the value function for the current policy π_k. The next step is to improve the policy by using the just calculated value function and applying

$$\pi_{k+1}(s) = \arg\max_a \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^{\pi}(s') \right]$$

for all states s. Policy Iteration continuously executes these two steps until no more improvement can be obtained. Algorithm 2 shows the pseudo code for policy iteration.

Algorithm 1: Value Iteration

Input: Threshold ϵ
Result: Optimal Value Function V^*

begin
  $V(s) \leftarrow 0 \quad \forall s \in \mathcal{S}$
  repeat
    $\delta \leftarrow 0$
    foreach $s \in \mathcal{S}$ **do**
      $V'(s) \leftarrow V(s)$
      $V(s) \leftarrow \max_a \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V(s') \right]$
      $\delta \leftarrow \max(\delta, |V'(s) - V(s)|)$
    end
  until $\delta < \epsilon$
  return $V^* = V$
end

Algorithm 2: Policy Iteration

Input: Some arbitrary policy π'
Result: Optimal Policy π^*

begin
  repeat
    $\pi \leftarrow \pi'$
    Policy Evaluation
    foreach $s \in \mathcal{S}$ **do**
      $V^{\pi}(s) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^{\pi}(s')$
    end
    Policy Improvement
    foreach $s \in \mathcal{S}$ **do**
      $\pi'(s) = \arg\max_a \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^{\pi}(s') \right]$
    end
  until $\pi = \pi'$
  return $\pi^* = \pi'$
end

As mentioned before, an MDP can be used to solve problems where the state current s_t is always known to the agent, meaning that the agent knows exactly in what state the environment is at time t. Unfortunately, this is not always the case, and an environment may have noisy properties, e.g. the GPS signal for determining the position of an agent. To overcome this problem, an extension to the MDP was introduced by [Kaelbling et al., 1998], namely

the *Partially Observable Markov Decision Process*, to model uncertainty by observations, observation probabilities and beliefs. The next section will give a definition of this model.

2.2 Partially Observable Markov Decision Process (POMDP)

In some scenarios, the agent might not be able to know its state in the environment. Therefore, the agent observes the information from the sensors and builds a belief state about where it *might* be. This uncertainty can be modeled by a Partially Observable Markov Decision Processes. A POMDP extends an MDP by observations and observation probabilities, which models decision making under uncertainty. Formally, a POMDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, T, r, \mathcal{O}, O, \gamma)$. The state space $\mathcal{S}$, action space $\mathcal{A}$, transition probability function T, reward function r and discount factor γ are inherited by the MDP. The observation space $\mathcal{O}$ is a set of observations the agent is able to receive. The transition probability function $T(s, a, s') = \Pr(s' \mid s, a)$ can be seen as the probability of being in state s' after executing action a in current state s. The observation probability function $O(s', a, o) = \Pr(o \mid s', a)$ describes the uncertainty of observing a particular observation o when executing action a.

The general process is close the one of the MDP. The agent takes an action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$, which leaves him in state $s' \in \mathcal{S}$. Simultaneously, the agent receives an observation $o \in \mathcal{O}$. The history at time t for actions and observations can be defined as the set of action-observation pairs

$$h_t = \{(a_0, o_1), \ldots, (a_{t-2}, o_{t-1}), (a_{t-1}, o_t)\}.$$

Since the agent does not know the current state it is in, a probability distribution over all states at any time has to be maintained. This probability distribution is indicated as the *belief state* or *belief* b. The probability of the agent being in state s is then given by $b(s)$. Additionally, the initial belief at time step $t = 0$ is denoted by $b_0(s)$. The set of all belief states is called the *belief space* $\mathcal{B}$. Both the history and the belief are sufficient statistics. Decisions under uncertainty can be made by either using the history or using the belief. As the belief is not always necessary, it is used preferably in many applications. In the following section, the definition for updating this belief is described.

2.3 Belief State Update

The probability of the agent being in state s is the belief $b(s)$ of state s. Based on the type of the state space (discrete states or continuous states), different methods for updating the belief can be applied. In this section, two main approaches for updating the belief are introduced. Discrete State Filter for a discrete state space, and Particle Filter for continuous state spaces.

2.3.1 Discrete State Filter

For a discrete state space $\mathcal{S}$, the new belief b_{t+1} for a new state $s' \in \mathcal{S}$ can be updated by applying Bayes' Rule recursively as follows.

$$\begin{aligned} b_{t+1}(s') = b_{t+1}(s' \mid b, a, o) &= \Pr(s' \mid b, a, o) \\ &= \frac{\Pr(o \mid s', a, b) \Pr(s' \mid b, a)}{\Pr(o \mid b, a)} \\ &= \frac{O(s', a, o) \sum_{s \in \mathcal{S}} \Pr(s' \mid b, a, s) \Pr(s \mid b, a)}{Pr(o \mid b, a)} \\ &= \frac{O(s', a, o) \sum_{s \in \mathcal{S}} T(s, a, s') b_t(s)}{\Pr(o \mid b, a)} \end{aligned} \tag{2.4}$$

Since the probability $\Pr(o \mid a, b)$ does not depend on the next state s', it is acting as a normalization factor [Kaelbling et al., 1998]. The definition of $\Pr(o \mid a, b)$ can be seen in equation (2.7) on the next page. This approach works well for small, discrete state spaces, but if the state space is too large or continuous, a Particle-based filtering method has to be applied. In the next subsection, the intuition of a particle in the context of a POMDP is explained. In addition, the general Particle Filter approach is shown.

2.3.2 Particle Filter

Particle Filters are used to determine a posterior distribution by a non-parametric representation through randomly drawing samples [Doucet et al., 2000]. A sample is a particle with a weight assigned to it. In terms of the POMDP, a particle is a state s_i with weight w_i. The belief b is then represented by a set of samples from the state space. In order to update the belief b, a particle s_i is randomly selected from the sample. With the given action a, the generative model $G(s,a)$ will propagate the particle s_i. This model acts as the system transition function. In order to calculate the weight w_i of the particle s_i', the observation probability function $O(s',a,o)$ is applied, given the currently observed observation o of the agent. This observation probability function will compute the probability of observing o when executing a and resulting in state s'. After all new particles s' are generated, the new belief b_{t+1} is constructed by drawing samples from this newly generated sample set with probability proportional to their weights. The pseudo code for the Particle Filter can be seen in algorithm 3.

Algorithm 3: Belief Update with Particle Filtering

Input: Current Belief b_t, Action a, Observation o
Result: Updated Belief b_{t+1}

```
begin
    b_{t+1} ← ∅
    for i ← 1 to |b_t| do
        s_i ← random particle in b_t
        s_i' ~ G(s_i, a)
        w_i ← O(s_i', a, o)
    end
    for i ← 1 to |b_t| do
        Select random j with probability proportional to w_j
        b_{t+1} ∪ {s_j}
    end
    return b_{t+1}
end
```

Sampling new states from the new particle vector is also referred to as *Importance Sampling*, as the particles are drawn from the generated particle set based on their weights. Particles with higher weights are more likely to be drawn than particles with lower weights. This also implies that a particle can be drawn multiple times, and particles with low weights may be left out at all.

2.4 Value Functions

The expected reward for a policy π is

$$V^{\pi}(b) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(b_t)) \;\middle|\; b_0 = b\right]. \tag{2.5}$$

As in MDPs, the POMDP also has an optimal value function. For a belief state b, the optimal value function is

$$V^*(b) = \max_{a \in \mathcal{A}}\left[\sum_{s \in \mathcal{S}} r(s,a)b(s) + \gamma \sum_{o \in \mathcal{O}} \Pr(o \mid b,a)\, V^*(b_{t+1}(s' \mid b,a,o))\right] \tag{2.6}$$

where $b_{t+1}(s' \mid b,a,o)$ is the updated belief calculated by equation (2.4) on the preceding page, and $\Pr(o \mid b,a)$ is given by

$$\Pr(o \mid b,a) = \sum_{s' \in \mathcal{S}} O(s',a,o) \sum_{s \in \mathcal{S}} T(s,a,s')b(s). \tag{2.7}$$

The dimensionality of the belief space $\mathcal{B}$ is equal to the number of states in the system, $|\mathcal{S}|$. Thus, the belief space grows exponentially with $|\mathcal{S}|$, leading to the "curse of dimensionality".

2.5 Solving POMDPs

The goal in solving a POMDP is to get the optimal policy π^* for which the POMDP executes optimal actions based on the current belief b. The question now is how to compute this optimal policy. In general, solutions for POMDPs can be separated into two sections: Exact solutions and approximated solutions. Exact solution methods will calculate the optimal policy, where approximated solution methods will calculate an approximation to the optimal policy. In the following subsections, two different approaches for solving a POMDP will be discussed. First, an exact solution method, namely Value Iteration will be covered. Then, one approximated solution method called DESPOT [Ye et al., 2017] will be introduced.

2.5.1 Value Iteration

Like the value iteration algorithm in an MDP, value iteration in a POMDP tries to find an optimal policy $\pi^*(b)$ for a belief state b, such that the value function for all belief states returns the maximum value.

$$\pi^*(b) \;=\; \arg\max_a \left[\sum_{s\in\mathcal{S}} r(s,a)b(s) + \gamma \sum_{o\in\mathcal{O}} \Pr(o \mid b,a)\; V^*(b_{t+1}(s' \mid b,a,o)) \right] \tag{2.8}$$

The full optimal policy π^* is the policy which maximizes equation (2.8) for all belief states $b \in \mathcal{B}$. Analogously to MDP value iteration, we initialize every value for every belief state to 0. Iteratively , the value function for the next time step $t+1$ is calculated using equation (2.6) on the previous page until the greatest error $\sup_{b\in\mathcal{B}} |V_t^*(b) - V_{t+1}^*(b)|$ is less than some pre-defined threshold ϵ. algorithm 4 shows the pseudo code for the policy value iteration.

Algorithm 4: POMDP Value Iteration

Input: Threshold ϵ
Result: Optimal Value Function V^*

```
begin
    t = 0
    V*_0(b) ← 0   ∀b ∈ B
    while sup_{b∈B} |V*_t(b) − V*_{t+1}(b)| > ε do
        foreach b ∈ B do
            V*_{t+1}(b) ← max_a [ Σ_{s∈S} r(s,a)b(s) + γ Σ_{o∈O} Pr(o | a, b) V*_t(b_{t+1}(s' | b,a,o)) ]
        end
    end
    return V* = V*_{t+1}
end
```

Smallwood and Sondik showed that the optimal value function can be represented by a set of vectors Γ [Smallwood and Sondik, 1973]. These vectors are called *alpha vectors* and define hyperplanes in the belief space $\mathcal{B}$. It can be shown, that the value function is convex and piecewise linear [Smallwood and Sondik, 1973]. We can rewrite the optimal value function in terms of alpha vectors by letting $\boldsymbol{\alpha}_a$ associate the alpha vector to action $a \in \mathcal{A}$, therefore $\boldsymbol{\alpha}_a = r(\cdot, a)$. Likewise, the belief $b(s)$ can also be represented as a vector $\boldsymbol{b}$. The value function for a belief state b is then

$$V^*(b) = \max_{\boldsymbol{\alpha}_a\in\Gamma} \boldsymbol{\alpha}_a^\top \boldsymbol{b}$$

The major problem with alpha vectors is that the set Γ grows exponentially in the number of observations at every iteration. The time complexity of generating an alpha vector lays in $O(|\mathcal{O}||\mathcal{S}|^2)$. In addition, the space complexity is in $O(|\mathcal{A}||\Gamma|^{|\mathcal{O}|})$, making it computationally intractable to compute exact solutions for POMDPs.

2.5.2 Determinized Sparse Partially Observable Trees (DESPOT)

The execution of all policies under all possible scenarios of a POMDP can be represented by a standard belief tree, where each node represents a belief. Two nodes are connected by an action-observation pairs. An example for

a simple standard belief tree can be seen in figure 2.1. It contains all action-observation histories. A path from the root node to a leaf node is one action-observation history. In general, at each belief node, we have $|\mathcal{A}|$ action branches, which further split into $|\mathcal{O}|$ observation branches.

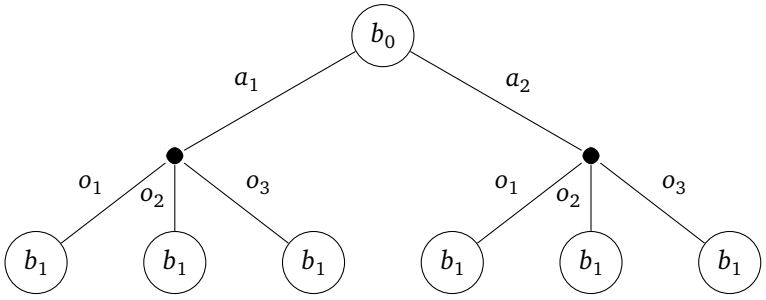


Figure 2.1: A standard belief tree for one time step with height $D = 2$. It has $2 = |\mathcal{A}|$ action branches and $3 = |\mathcal{O}|$ observation branches. Each node represents a belief. Each edge between two belief nodes represents an action-observation pair.

In order to get an optimal action a^* from this belief tree, the agent performs lookahead search on the tree. Starting from the root node, the agent computes the policy π that maximizes the value $V^\pi(b_0)$ for the initial belief b_0 from equation (2.5) on page 7. The optimal action a^* can then be obtained from this policy $\pi(b_0)$. As the height of the tree grows proportional to the number of time steps, the optimal policy can be approximated by truncating the tree at a height D. The lookahead search can be performed on this truncated tree by using the optimal value function 2.6 on every belief node b of the tree. Leaf-nodes will be assigned a value based on a *default policy*, which can be a random policy or a heuristic, estimating the value at that belief state. The optimal value function will then be applied on the belief tree in post-order to compute recursively the maximum value for each belief node and return the best action a^* for the belief node at the root of the tree.

However, constructing the whole standard belief tree to a height D can be computationally expensive, especially when the observation spaces is large. A *Determinized Sparse Partially Observable Tree* (DESPOT) [] overcomes this problem by computing a sparse belief tree, where only some paths of the standard belief tree are computed. An example for a DESPOT can be seen in figure 2.2. More specifically, a DESPOT contains all action branches, but only one observation branch per action. These paths are called *scenarios*, which are sequences of abstract simulation trajectories, starting at an initial state $s_0 \in \mathcal{S}$. Formally, a scenario is an infinite random sequence $\boldsymbol{\phi} = (s_0, \phi_1, \phi_2, \dots)$ of real numbers ϕ_i, which are drawn independently and uniformly from the interval $[0, 1]$. The starting state s_0 is sampled according to the initial belief b.

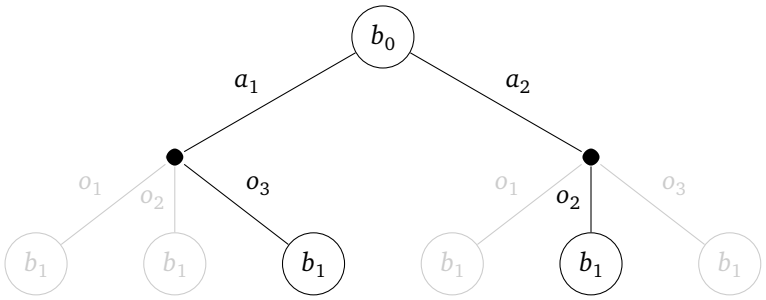


Figure 2.2: The standard belief tree in gray, overlayed by the DESPOT in black. The DESPOT contains all action branches from the standard belief tree, but only one observation branch per action.

2.5.2.1 Building the DESPOT

The DESPOT can be build by applying a *deterministic simulative model*, which simulates an execution of action a in state s, to all possible action sequences under K sampled scenarios. This deterministic simulative model is a function $g : \mathcal{S} \times \mathcal{A} \times \mathbb{R} \to \mathcal{S} \times \mathcal{O}$, which takes a state s, an action a and a random number ϕ from the infinite sequence and returns the next state s' and corresponding observation o' according to the probability distribution $\Pr(s', o' \mid s, a) = T(s, a, s')O(s', a, o')$. With this model, we can simulate an action sequence $(a_1, a_2, \dots)$ under a scenario

$(s_0, \phi_1, \phi_2, \dots)$. The result will be a simulation trajectory with a path of action-observation pairs $(a_1, o_1, a_2, o_2, \dots)$, starting at the root of the belief tree. An example for a trajectory can be seen in figure 2.3. The simulation trajectory $(a_1, o_3, a_{2,2})$ colored in blue reaches from the root of the DESPOT to the leaf node b_2. Each belief node b additionally has a set of scenarios Φ_b, which contains all scenarios that have been encountered in this node. Thus, the root node b_0 contains the scenario $(s_0, \phi_1, \phi_2, \dots) = \boldsymbol{\phi} \in \Phi_{b_0}$. Analogously, for every other node belief b_t is $(s_t, \phi_{t+1}, \phi_{t+2}, \dots) = \boldsymbol{\phi} \in \Phi_{b_t}$. The whole DESPOT D can then be constructed by repeating this process for every action sequence under all sampled scenarios.

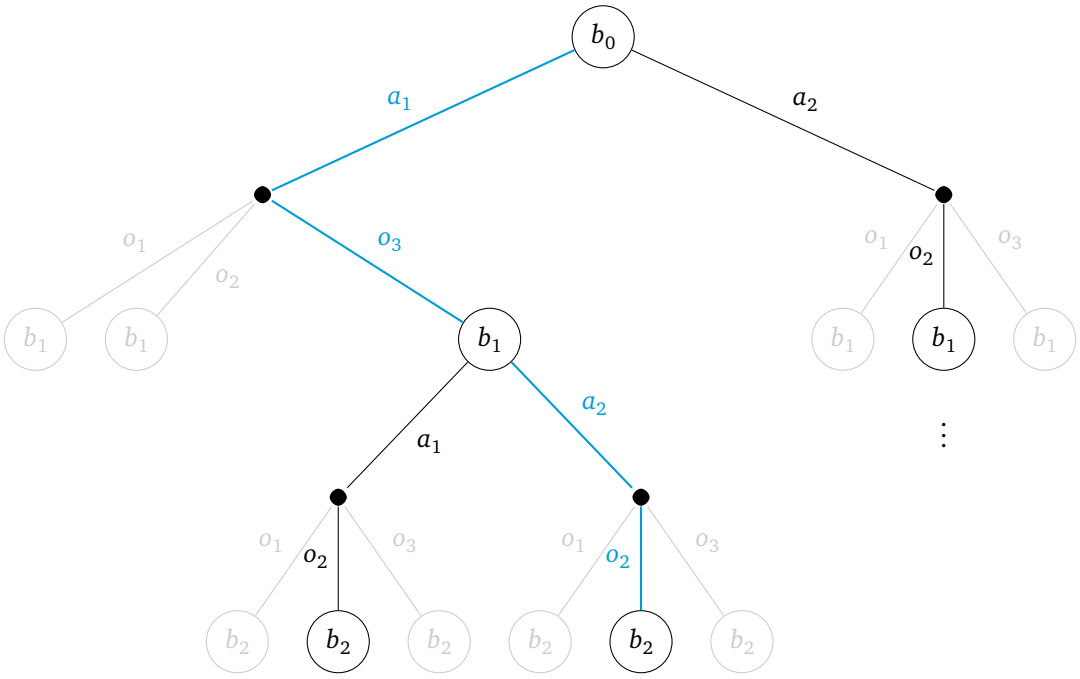


Figure 2.3: Another example for a DESPOT with a possible simulation trajectory colored in blue from the action sequence (a_1, a_2). The DESPOT itself is colored in black, while the standard belief tree is colored in gray.

In a DESPOT, we can define the *empirical value* $\hat{V}_\pi(b)$ for a belief b of policy π, which corresponds to the average discounted reward, as

$$\hat{V}_\pi(b) = \sum_{\boldsymbol{\phi} \in \Phi_b} \frac{V_{\pi,\boldsymbol{\phi}}}{|\Phi_b|},$$

where $V_{\pi,\boldsymbol{\phi}}$ denotes the discounted reward obtained by simulating the policy π under sequence $\boldsymbol{\phi}$, and $|\Phi_b|$ is the number of scenarios encountered in the belief node b. A policy π in the DESPOT can be represented as a *policy tree* derived from D. Instead of having all action branches like in the DESPOT, a policy tree has only one action branch at each layer. An agent starts at the root belief node b_0, executes action a_0 according to π given the action branch and observes an observation. It then takes the observation branch to the next node. If the encountered observation is not present, which can be the case as the policy tree only contains observations resulting from the sampled scenarios, the agent will follow a default policy instead. If the agent reaches a leaf node, it also follows the default policy.

2.5.2.2 Action Selection using Dynamic Programming

In order to compute an optimal action from the DESPOT D, we first construct the DESPOT with K randomly sampled scenarios using the deterministic simulative model. According to [Ye et al., 2017], the objective function which needs to be maximized is

$$\hat{V}_\pi(b_0) - \lambda|\pi|, \tag{2.9}$$

with $\lambda \geq 0$, where b_0 is the current belief and $|\pi|$ the size of the policy π. The policy, that maximizes (2.9), is then computed by

$$\max_{\pi \in \Pi_D} \left\{ \hat{V}_\pi(b_0) - \lambda|\pi| \right\}.$$

The set Π_D contains all policies of the DESPOT D. The *regularized weighted discounted utility* (RWDU) for each belief node b of the policy tree π is

$$v_\pi(b) = \frac{|\Phi_b|}{K} \gamma^{\Delta(b)} \hat{V}_{\pi_b}(b) - \lambda|\pi_b|,$$

where $|\Phi_b|$ is the number of scenarios passing through belief node b, γ is the discount factor of the POMDP, $\Delta(b)$ is the depth of belief node b in the policy tree π, π_b is the subtree rooted at node b, and $|\pi_b|$ is the size of policy π_b. $|\Phi_b|/K$ denotes the empirical estimate of the probability for reaching belief node b. For the initial belief b_0, the set of scenarios passing through the belief node b_0 contains all sampled scenarios, hence $|\Phi_{b_0}| = K$. The depth $\Delta(b_0)$ is 0. Therefore, we get

$$v_\pi(b_0) = \hat{V}_\pi(b_0) - \lambda|\pi|.$$

We can now Assuming that D has finite depth, for every belief node b in D we define $v^*(b)$ as the maximum RWDU of b over all policies in Π_D. In order to compute v^*, we start at a leaf node b of D and simulate the default policy π_0 under the sampled scenarios. As the default policy has size 0, we get

$$v^*(b) = \frac{|\Phi_b|}{K} \gamma^{\Delta(b)} \hat{V}_{\pi_0}(b).$$

At each internal belief node b, the maximum RWDU is then computed recursively through

$$v^*(b) = \max \left\{ \frac{|\Phi_b|}{K} \gamma^{\Delta(b)} \hat{V}_{\pi_0}(b),\ \max_{a \in \mathcal{A}} \left\{ \rho(b,a) + \sum_{o \in \mathcal{O}_{b,a}} v^*(\tau(b,a,o)) \right\} \right\}, \tag{2.10}$$

where $\tau(b,a,o)$ is the child node of belief node b following action branch a and observation branch o at b, $\mathcal{O}_{b,a}$ is the set of observations following action branch a from belief node b, and $\rho(b,a)$ is the reward for belief b and action a given by

$$\rho(b,a) = \frac{1}{K} \sum_{\phi \in \Phi_b} \gamma^{\Delta(b)} r(s_\phi, a) - \lambda$$

with s_ϕ being the start state of scenario ϕ. The outer maximization in equation (2.10) chooses between executing the default policy or expanding the belief node b. The inner maximization recursively computes the utility for the belief node b for all actions $a in \mathcal{A}$. The optimal action is then given by the maximizer at the root node b_0 of D.

2.6 Related Work

In the field of robotics, autonomous agents experience an increase of interest. Autonomous robots are able to interact with their environment and learn from experience by rewards and costs applied to the robot's actions. The goal of the robot then is to maximize its expected reward from the current time step onward by choosing optimal

actions. These actions are determined by a certain strategy, also known as a *policy*, which has to be adapted and optimized at each time step. The policy returns an action based on the state in the environment the robot currently is. When it comes to determine an optimal action for the agent, decision theory [Raiffa, 1968] lays the foundation for the mathematical concepts of decision making. This concept of trial-and-error learning is known as "Reinforcement Learning" [Sutton and Barto, 1998], as the agent tries to get better by incorporating the experience of the past and the expected rewards of the future. One mathematical concept of solving reinforcement learning problems is the use of a Markov Decision Process (MDP). It provides the foundation of modeling the problem statement into mathematical formulas and equations. To achieve the goal of maximizing the robot's accumulated reward in an MDP, various solving methods have been proposed over the past decades. The solving approaches follow the general idea of *Dynamic Programming* [Bertsekas et al., 1995], which key idea is to break down a complex problem into simpler problems, which can be solved first. The most known algorithm is *Value Iteration* algorithm [Bellman, 1957a], which has been introduced by Richard Bellman in 1957. Another similar solving method is *Policy Iteration* [Howard, 1960] introduced by Ronald A. Howard in 1960. These models expect to have a fully observable environment, meaning that the agent knows everything about the environment at any time.

Unfortunately, the full environment of the robot is not always known and information about the state is only partially observable. When dealing with uncertainty and uncertain environments, a Partially Observable Markov Decision Process (POMDP) is often used, as it is able to model uncertainty through observations and beliefs. In addition, different variants of POMDPs has been developed in order to model individual problem statements [Pineau and Thrun, 2001; Fard and Yahya, 2012]. Based on these mathematical frameworks, a variety of problems can be solved by using different solving methods, such as Policy Graph Improvement (PGI) [Pajarinen and Peltonen, 2011], POMCP [Silver and Veness, 2010] or SARSOP [Kurniawati et al., 2008]. A survey of point-based solving methods for POMDPs can be seen in [Shani et al., 2013].

2.6.1 Motion Planning Under Uncertainty

In motion planning, autonomous robots have to be aware of uncertainty from e.g. noisy sensors of the robot, leading to partially available information of its state. Therefore, computing the best action causes high computational complexity. In order to handle the task of motion planning under uncertainty, various solutions have been published.

Ong et al. [2010] proposed an approach where two different solving methods are compared. The first method uses a *Mixed Observable Markov Decision Process* (MOMDP) [Fard and Yahya, 2012] together with a point-based algorithm to compute the policy. The second method uses a POMDP in combination with the point-based algorithm SARSOP [Kurniawati et al., 2008], which stands for *Successive Approximations of the Reachable Space under Optimal Policies*. In general, SARSOP samples a set of points from the belief space to use it as a representation for the whole space. Experiments were made on different problems, such as the *Tag* problem, firstly mentioned in [Pineau et al., 2003a], where the agent has to follow an object in a grid world that moves away from it. The agent only knows the position of the object, when both are at the same position. Results show that the MOMDP can solve the problem faster than the POMDP with the original SARSOP algorithm, even when the amount of states is increased.

Kurniawati et al. [2011] tested different agents on planning while navigating in 2-dimensional, as well as in 3-dimensional space. In the 3-dimensional case, an unmanned aerial vehicle (UAV) navigates in an indoor environment without the use of GPS. This loss of localization causes uncertainty in the robots position. In addition, the environment contains obstacles and danger zones, which must be avoided. Predefined landmarks help the robot to localize itself. In order to solve this task, Kurniawati et al. proposed a point-based POMDP solver named *Milestone Guided Sampling* (MiGS), which samples a set of points from the state space and uses them as a simplified representation of the state space. This representation is then used to guide sampling in the belief space of the POMDP. Results show that over time, the success rate of the agent to reach its goal increases.

2.6.2 Decision Making Under Uncertainty

As for decision making under uncertainty, Pineau et al. [2003b] described and developed a mobile robotic assistant to assist elderly people with cognitive and physical restrictions. The robot is made of three software modules, one of those being a high-level robot controller for planning under uncertainty. In particular, decision making is performed for responding to an individual by speech recognition, and for robot control. Therefore, the number of states and the number of actions for the agent to take is very large. In order to solve this task, a hierarchical version of

the POMDP was used [Pineau and Thrun, 2001], in which the action space is partitioned for solving the POMDP more efficiently. For solving this problem, the MAX-Q algorithm from [Dietterich, 1998] was adapted to work on POMDPs.

In autonomous driving, traffic participants behave in a stochastic manner, leading to uncertainty. Brechtel et al. [2014] tackled the problem of decision making under uncertainty in the context of driving an autonomous vehicle. Traffic participants may be not visible to the agent because the view can be occluded by obstacles. The goal of the car is to cross an intersection safely without crashing into other traffic participants. The resulting POMDP here is a continuous state space POMDP, which is been solved by the Monte Carlo Value Iteration algorithm, including incremental learning [Brechtel et al., 2013]. Experiments were run multiple times in simulation. The results show, that the Monte Carlo Value Iteration algorithm handled different scenarios well, even when traffic participants were occluded by obstacles.

2.6.3 Collision Avoidance for Unmanned Aircraft

One field for solving the problem of collision avoidance is unmanned aircrafts. Avoiding collisions with unmanned aircrafts has been experienced increased attention since 2009 [Wolf, 2009; Temizer et al., 2010; Bai et al., 2012]. These works differ in solving the given problem statement but using the same mathematical tool.

In the work of [Temizer et al., 2010], both MDPs and POMDPs were used to model the behavior of the aircraft. The MDP was used when perfect sensors were assumed. As there is no model for uncertainty in an MDP, the uncertainty in intruder aircraft behavior is modeled as a random process. This will lead to uncertainty in state transitions. For noisy sensors and uncertainty in observations, the POMDP was applied. The MDP was solved using the *value iteration* algorithm, which is a method for returning an exact solution for the policy. The results were promising due to the assumption of perfect sensing, but in real applications, perfect sensing is difficult to accomplish. The POMDP was solved using the point-based algorithm SARSOP. The experiments were made with different sensors. In conclusion, the POMDP solution performed well on these sensors. Due to intractable computations, the policy-search was done offline.

Another work in the field of aircraft collision avoidance is done by Travis Benjamin Wolf in 2009 [Wolf, 2009]. In order to solve the collision avoidance problem, they proposed an online solving method named *Monte Carlo Real-Time Belief Space Search* (MC-RTBSS), which combines a sample-based belief state representation with a branch and bound pruning method to search for the optimal policy in the belief space.

In 2012, Bai et al. [2012] proposed a solution for avoiding collisions for unmanned aircraft by using Continuous-State POMDPs and solving them with the *Monte Carlo Value Iteration* (MCVI) algorithm [Bai et al., 2010]. MCVI samples the agent's state space and the corresponding belief space in order to avoid *a priori* discretization of the state space. As the results show, MCVI found well-performing policies for this task.

2.6.4 Collision Avoidance for Pedestrians

Another interesting field where collision avoidance is a crucial task derives from traffic. More precisely, the problem of avoiding collision with pedestrians, which is also the core application of this thesis. Pedestrians are one of the most vulnerable traffic participants when it comes to civil traffic.

Bandyopadhyay et al. [2013] proposed a method for solving this task by formulating the pedestrian avoidance problem as an *Intention-Aware Motion Planning* problem. They assume that there is a set of all possible pedestrian intentions in the environment, which enables a MOMDP to model this problem. A MOMDP is a factored variant of the POMDP, where the state space contains tuples of fully and partially observable variables. This is advantageous as there is only the belief of the partially observable variable to be maintained. In conjunction with the SARSOP solving algorithm, the experiments show that this task can be solved online.

The work of Bai et al. [2015] — which has been re-implemented and adapted to different scenarios in this thesis — tackles the very same problem and uses a POMDP together with the DESPOT solving algorithm as described in subsection 2.5.2 on page 8. The POMDP has been run online on a golf cart. Bai et al. used pre-defined goals to model the pedestrian's intention and update the belief with the help of a belief tracker, which - as the name mentions - tracks the goal of the pedestrian.

In this work, the collision avoidance approach from [Bai et al., 2015] is been re-implemented. In particular, the DESPOT solving algorithm was used as it has been shown to work in the context of collision avoidance satisfactorily. Although the DESPOT algorithm shows promising results, it is not the only algorithm that can be used for this

problem statement. Other solvers such as POMCP [Silver and Veness, 2010], SARSOP [Kurniawati et al., 2008] or PGI [Pajarinen and Peltonen, 2011] could be used instead.

3 Belief Representation and Classifying Pedestrian Types

This chapter introduces the concept of the belief for collision avoidance. Representations for the belief of the POMDP model, as well as equations for updating this belief are defined.

3.1 Problem Statement

In the environment of the agent, we want to avoid collisions with pedestrians. In particular, a pedestrian is a dynamic obstacle with a position, instantaneous velocity and a target at any time step. This target will be denoted by *the goal* of the pedestrian and is predefined. A goal g is a location in the environment and therefore has coordinates $\boldsymbol{g} = (x_g, y_g)^\mathsf{T}$. The agent in the POMDP needs to predict the pedestrian's movement and target goal, as the intention of the pedestrian is to approach this goal. Hence, the fully observable variables are the position and instantaneous velocity of the pedestrian. The goal however is partially observable and therefore is part of the belief b. As mentioned in section 2.3 on page 6, we have to maintain and update this belief at every time step. In addition, subsection 3.2.2 on the facing page will introduce the belief over predefined types for the pedestrian as an extension to the model by [Bai et al., 2015]. The following sections will define the belief over goals and types, as well as their update equations for maintaining the belief at every time step.

3.2 Modeling the Belief for different Pedestrian Goals and Types

This section provides definitions for representing the belief with different pedestrian goals and types. First, we will look at the belief over pedestrian goals and derive the update equation respectively. The belief over types is then defined. Afterwards, we combine the belief over goals and the belief over types in order to get the belief over both goals and types of the pedestrian.

3.2.1 Belief over Pedestrian Goals

Since the partial observable variable is the goal $g \in \mathbb{R}^2$ of the pedestrian, the belief will be represented as a probability distribution over all predefined goals. As we are referring to the model for collision avoidance from [Bai et al., 2015], we briefly show the definition of the belief. The belief over all goals g is represented as a vector of distributions d_i for $i = 1, \dots N$, where N is the number of predefined goals. Every distribution d_i corresponds to the belief for goal g_i.

$$b_t(g) = \begin{pmatrix} d_1 \\ \vdots \\ d_N \end{pmatrix}, \quad b_t(g_i) = d_i \quad \text{for } i = 1, \dots, N = |G|$$

In order to maintain the belief, it needs to be updated at every time step. The next subsection will derive the update equation needed for the model to incorporate new observations. Updating the belief is crucial as it contains information on what the pedestrian's intention might be and what his next step could be. In order to maintain this information, the belief needs to be updated with the new observation. As discussed in section 2.3 on page 6,

updating the belief is done by applying Bayes' Rule, as shown in equation (2.4) on page 6. The updated belief $b_{t+1}(g)$ for a goal g is then given by applying Bayes' rule. This update will lead to the following equation.

$$\begin{aligned} b_{t+1}(g) &= \Pr(g \mid o_t, b) \\ &= \frac{\Pr(o_t \mid g, b)\Pr(g \mid b)}{\Pr(o_t \mid b)} \\ &= \frac{\Pr(o_t \mid g, b)\Pr(g \mid b)}{\sum_g \Pr(o_t \mid g)\Pr(g \mid b)} \\ &= \frac{\Pr(o_t \mid g, b) b_t(g)}{\sum_g \Pr(o_t \mid g) b_t(g)} \end{aligned} \tag{3.1}$$

Here, $\Pr(o_t \mid g, b)$ is the pedestrian observation model and returns the probability of the pedestrian being at the observed position from the current observation o_t, given the calculated position $\mathbf{x}'_p = (x'_p, y'_p)$. This position will be computed by using the current belief b_t. The pedestrian movement function will calculate this new position by incorporating the last observed position $\mathbf{x}_p = (x_p, y_p)$, the velocity v_p, the current goal g of the pedestrian after time Δt. The definition of this movement function h is

$$\begin{aligned} h(\mathbf{x}_p, v_p, g) &= \mathbf{x}_p + \hat{\boldsymbol{u}}\, v_p \Delta t \\ &= \mathbf{x}_p + \frac{\boldsymbol{g} - \mathbf{x}_p}{||\,\boldsymbol{g} - \mathbf{x}_p\,||_2} v_p \Delta t, \end{aligned} \tag{3.2}$$

where $\hat{\boldsymbol{u}}$ is the normalized unit vector of the difference of the goal g and the position $\mathbf{x}_p$ of the pedestrian. The change in position between two time steps t_1 and t_2 is computed by $v\Delta t$. The normalized difference of the goal position and the current position will give the direction in which the pedestrian will move. Thus, the probability from the observation model $\Pr(o_t \mid g, b)$ can now be calculated using the multivariate Gaussian probability distribution function $f(\boldsymbol{x}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ with dimensionality $n = 2$, where $\boldsymbol{x} = (o_t^x, o_t^y)^\mathsf{T}$ is the current observed pedestrian position from observation o_t, and $\boldsymbol{\mu} = (x'_p, y'_p)^\mathsf{T}$ the calculated position according to equation (3.2). This probability distribution function is defined as

$$f(\boldsymbol{x}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^n \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^\mathsf{T} \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu})\right). \tag{3.3}$$

The denominator of the Belief-Update in equation (3.1) is the normalization factor and sums up all probabilities of the pedestrian being at the current observed position over all goals. The result $b_{t+1}(g)$ will contain the updated probability of goal g. Algorithm 5 on the next page shows the pseudo code for updating the belief at time step t given the current belief b_t, last observed pedestrian position $(x_p, y_p)_{t-1}$ from observation o_{t-1} and the currently observed pedestrian position $(x_p, y_p)_t$ from observation o_t. The result will be the updated belief b_{t+1}.

3.2.2 Belief over Pedestrian Types

This section will describe the concepts introduced in this thesis, which are not covered by the work of [Bai et al., 2015]. In addition to the model of collision avoidance for pedestrians from subsection 4.1.1 on page 21, the state $s \in \mathcal{S}$ will include a variable named *type*, which can take the values *trustworthy* or *untrustworthy*. This variable indicates, whether the pedestrian is a trustworthy person and will not suddenly walk in an arbitrary manner towards his goal, differing from his optimal path. A pedestrian is trustworthy, if he approaches his goal directly without any noise in his movement. On the other hand, a pedestrian is untrustworthy, if it approaches it goal non-optimally with noise in its movement. In general, there can be more than just two types. The only restriction to a type is that it can be modeled by the pedestrian model. In that case, the set $\mathcal{T}$ holds all possible types t of the pedestrian. The belief now is defined as

$$b_t(t) = \begin{pmatrix} d_1 \\ \vdots \\ d_N \end{pmatrix}, \quad b_t(t_i) = d_i \quad \text{for } i = 1, \ldots, N = |\mathcal{T}|.$$

Algorithm 5: Belief Update with Goals for Collision Avoidance

Input: Current Belief b_t, Current Observed Pedestrian Position (o_t^x, o_t^y)
Data: Normalization factor η; Observation Probability p_g for goal g
Result: Updated Belief b_{t+1} as the new distribution over all goals

begin
  $b_{t+1} \leftarrow \emptyset$
  $\eta \leftarrow 0$
  /* Calculate normalization factor and each observation probability */
  foreach $g \in G$ **do**
    $\mathbf{x}'_p = h(\mathbf{x}_p, \nu_p, g)$
    $p_g = f((o_t^x, o_t^y)^\mathsf{T}, (x'_p, y'_p)^\mathsf{T}, \mathbf{\Sigma})$ from $\mathcal{N}\big((o_t^x, o_t^y)^\mathsf{T} \mid (x'_p, y'_p)^\mathsf{T}, \mathbf{\Sigma}\big)$
    $\eta = \eta + p_g$
  end
  /* Calculate new goal distribution b_{t+1} */
  foreach $g \in G$ **do**
    $b_{t+1}(g) \leftarrow \dfrac{p_g b_t(g)}{\eta}$
  end
end
return b_{t+1}

Notice that the subscript t will always denote the time step of the belief, while the parameter t will indicate the type of the pedestrian.

With including types into the pedestrian model, the dynamics of this model changes slightly. In particular, the variance in the Gaussian is changed according to the specified type. If the type for the pedestrian is t = trustworthy, the variance will be assigned a small constant. If the pedestrian type is t = untrustworthy, the variance will be a slightly larger value. In order to update the belief over the types only, we apply Bayes' Rule equation (2.4) on page 6 analogously to the belief over goals.

$$
\begin{aligned}
b_{t+1}(t) &= \Pr(t \mid o_t, b) \\
&= \frac{\Pr(o_t \mid g, t, b)\Pr(t \mid b)}{\Pr(o_t \mid b)} \\
&= \frac{\Pr(o_t \mid g, t, b)\Pr(t \mid b)}{\sum_t \Pr(o_t \mid t)\Pr(t \mid b)} \\
&= \frac{\Pr(o_t \mid g, t, b) b_t(t)}{\sum_t \Pr(o_t \mid t) b_t(t)}
\end{aligned}
$$

Classifying pedestrian types requires the goal to be known, as the term $\Pr(o_t \mid g, t, b)$ is the probability for the current pedestrian observation being the calculated pedestrian position, given goal g and type t. Because the goal is also a partial observable variable, we need to incorporate the goal into the belief. In the next section, the definition of the belief over goals and types is introduced. Also, updating this belief is shown.

3.2.3 Belief over Goals and Types

As we introduced goals and types as part of the state space, we can now combine these information in order to form a belief over goals and types. The belief b now is a joint probability distribution over goals and types and can be visualized as a table, where the rows indicate the goals, and the columns indicate the types. Table 3.1 on the facing page shows the probabilities for each combination of goals and types, as well as the marginal probabilities.

		Type			
		t_1	$\cdots$	t_m	
Goal	g_1	$\Pr(g_1, t_1)$	$\cdots$	$\Pr(g_1, t_m)$	$\Pr(g_1)$
	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
	g_n	$\Pr(g_n, t_1)$	$\cdots$	$\Pr(g_n, t_m)$	$\Pr(g_n)$
		$\Pr(t_1)$	$\cdots$	$\Pr(t_m)$	

Table 3.1: The probability table for the belief over goals and types. The inner table contains each combination of goal and type. The marginal probabilities represent the corresponding beliefs.

In order to obtain the marginal probability for some goal g, we sum over the probabilities for that goal and each type $t \in \mathcal{T}$ as follows.

$$\Pr(g) = \sum_{t \in \mathcal{T}} \Pr(g, t) = b(g)$$

Analogously, the marginal probability for some type t can be computed by summing over all probabilities for that type and each goal $g \in G$ as follows.

$$\Pr(t) = \sum_{g \in G} \Pr(g, t) = b(t)$$

With this formalization of the type and goal of a pedestrian, the belief update equation can now be derived. The belief over a goal $g \in G$ and a type $t \in \mathcal{T}$ can be updated by incorporating the current observation o_t and current belief b. With these information, the belief update function can be derived as follows.

$$\begin{aligned} b_{t+1}(g, t) &= \Pr(g, t \mid o_t, b) \\ &= \frac{\Pr(o_t \mid g, t, b) \Pr(g, t \mid b)}{\Pr(o_t \mid b)} \\ &= \frac{\Pr(o_t \mid g, t, b) \Pr(g, t \mid b)}{\sum_g \sum_t \Pr(o_t \mid g, t, b) \Pr(g, t \mid b)} \end{aligned}$$

Furthermore, $\Pr(g, t \mid b)$ is the current belief $b_t(g, t)$ of goal g and type t. The term $\Pr(o_t \mid g, t, b)$ is the probability for the current pedestrian observation being the calculated pedestrian position, given goal g and type t. With this notation, we end up with the final belief update function

$$b_{t+1}(g, t) = \frac{\Pr(o_t \mid g, t, b) b_t(g, t)}{\sum_g \sum_t \Pr(o_t \mid g, t, b) b_t(g, t)} \tag{3.4}$$

The pseudo code for updating the new belief model also changes, which can be seen in algorithm 6 on the next page.

3.3 Incorporating Goals and Types in the Model by Updating the Particle Vector

With the updated belief, the vector of particles $\boldsymbol{p} = (p_1, \ldots, p_K)^\mathsf{T}$ has to be updated as well. The particle vector consists of K particles p_i. A particle is a state s with a weight w assigned to it, which needs to be calculated based on the distribution over goals and types. First, the number of particles k_i for each combination of goal and type is determined by multiplying the probability value $\Pr(g, t) = b(g, t)$ for goal g and type t by the total number of particles. To ensure the number of all particles sum up to K, the last partition of particles is the difference

$$k_N = K - \sum_{g \in G} \sum_{t \in \mathcal{T}} \Pr(g, t) K = K - \sum_{i=1}^{N-1} d_i K$$

Algorithm 6: Belief Update with Types for Collision Avoidance

Input: Current Belief b_t, Current Observed Pedestrian Position $(o_t^x, o_t^y)^\mathsf{T}$

Data: Set of goals G; Set of types $\mathcal{T}$; Normalization factor η; Observation Probability $p_{g,t}$ for goal g and type t; Covariance Matrix $\mathbf{\Sigma}_t$ and $\mathbf{\Sigma}_u$ for type trustworthy and untrustworthy respectively

Result: Updated Belief b_{t+1} as the new joint distribution over all goals and types

begin
  $b_{t+1} \leftarrow \emptyset$
  $\eta \leftarrow 0$
  /* Calculate normalization factor and each observation probability */
  for $g \in G$ **do**
    $\mathbf{x}'_p = h(\mathbf{x}_p, v_p, g)$
    foreach $t \in \mathcal{T}$ **do**
      if $t = trustworthy$ **then**
        $\mathbf{\Sigma} \leftarrow \mathbf{\Sigma}_t$
      end
      if $t = untrustworthy$ **then**
        $\mathbf{\Sigma} \leftarrow \mathbf{\Sigma}_u$
      end
      $p_{g,t} = f((o_t^x, o_t^y)^\mathsf{T}, (x'_p, y'_p)^\mathsf{T}, \mathbf{\Sigma})$ from $\mathcal{N}\left((o_t^x, o_t^y)^\mathsf{T} \,|\, (x'_p, y'_p)^\mathsf{T}, \mathbf{\Sigma}\right)$
      $\eta = \eta + p_{g,t}$
    end
  end
  /* Calculate new goal distribution b_{t+1} */
  foreach $g \in G$ **do**
    foreach $t \in \mathcal{T}$ **do**
      $b_{t+1}(g,t) \leftarrow \dfrac{p_{g,t} b_t(g,t)}{\eta}$
    end
  end
end
return b_{t+1}

For every particle p_i the weight w_i will take the value of the corresponding probability value $\Pr(g,t)$. As the belief over goals is a joint distribution over goals and types, the weights of the particles have to sum up to one. To achieve this, the weights have to be normalized. This is done by dividing each weight by the sum of all weights.

$$w_i = \frac{w_i}{\sum_{j=1}^{K} w_j}$$

The pseudo code for updating the particle vector can be seen in algorithm 7.

Algorithm 7: Particle Vector Update

Input: Current particle vector $\boldsymbol{p}_t$, Updated Belief b_{t+1} from algorithm 5 on page 16
Data: Total number of particles K; Total number of goals N; Weight $w(p)$ of a particle
Result: Updated particle vector $\boldsymbol{p}_{t+1}$

begin
  $\boldsymbol{p}_{t+1} \leftarrow \emptyset$
  $K \leftarrow |\boldsymbol{p}_t|$
  $N \leftarrow |b_{t+1}|$
  /* Compute the number of particles for each combination of goal g and type t */
  $i \leftarrow 1$
  foreach $g \in G$ *and* $t \in \mathcal{T}$ **do**
    $k_i \leftarrow b_{t+1}(g,t)K$
    $i = i + 1$
  end
  /* Number of particles for last goal and type combination */
  $k_N \leftarrow K - \sum_{g \in G} \sum_{t \in \mathcal{T}} b_{t+1}(g,t)K$
  /* Set weight of each particle to the corresponding probability value from belief */
  for $i \leftarrow 1$ **to** N **do**
    Set $g \in G$ and $t \in \mathcal{T}$
    for $j \leftarrow 1$ **to** k_i **do**
      $w(p_j) \leftarrow b_{t+1}(g,t)$
    end
  end
  /* Normalize weights and add the particle to the new particle vector */
  foreach $p \in \boldsymbol{p}_t$ **do**
    $w(p) \leftarrow \frac{w(p)}{\sum_{i=1}^{K} w(p)}$
    $\boldsymbol{p}_{t+1} \cup \{p\}$
  end
  return $\boldsymbol{p}_{t+1}$
end

4 Setup and Implementation

In this chapter, the setup for the simulation is described. A POMDP model for collision avoidance will be defined to serve as a basis for the implementation. In addition, the general implementation of the DESPOT framework will be shown.

4.1 Simulation-based Setup

In order to test the algorithm, a simulator was built using the Robot Operating System (ROS)[1] primarily. The simulator mimics a small environment of the car including a static map, the car itself, and pedestrians as cylindric markers. In order to plan a path for the car to a destination, the `teb_local_planner`[2] is used. A snapshot of the environment can be seen in figure 4.1. The map is a binary valued picture, where each pixel can take either a grayscale value of 0 or 255 for representing a wall or drivable area respectively. This information will serve the path planner as a costmap to determine the optimal global path for the car.

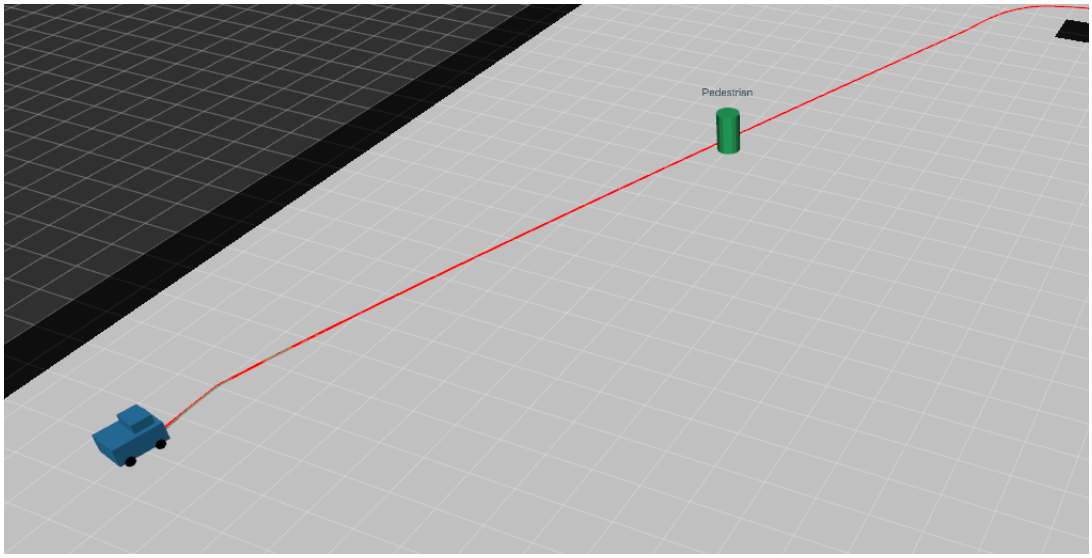


Figure 4.1: Simulation environment of the car, the pedestrian and the path planned for the car. The car colored in blue will stick to the path planned, marked in red. The pedestrian, colored in green, will move towards a goal position.

The car is modeled by its position $\mathbf{x}_c = (x_c y_c)$, orientation θ and its velocity v_c. The change dx and dy in position for the car as a non-holonomic robot can then be computed by

$$dx = v_c \cos(\theta)\Delta t, \tag{4.1}$$

$$dy = v_c \sin(\theta)\Delta t, \tag{4.2}$$

where Δt is the time between two time steps. The pedestrian is modeled by its position $\mathbf{x}_p = (x_p y_p)$, instantaneous velocity v_p and its desired goal $g_i = (x_i, y_i)$ for some predefined goals $g_i \in G$ from a set of goals G. The car does know the locations of all goals, but it does not know the real goal of the pedestrian. This partially observable property is needed in order to predict the pedestrians' behavior.

The pedestrians movement behavior is modeled analogously by calculating the difference in position given the current position $\mathbf{x}_p = (x_p, y_p)$ and velocity v_p. As the pedestrian only moves towards a given goal $\boldsymbol{g}$, the next position after time Δt can be calculated by

[1] http://www.ros.org/

[2] https://github.com/rst-tu-dortmund/teb_local_planner

$$h(\mathbf{x}_p, v_p, g) = \mathbf{x}_p + \frac{g - \mathbf{x}_p}{||g - \mathbf{x}_p||_2} v_p \Delta t \tag{4.3}$$

In the simulation engine, the pedestrian controller computes the change in position of the pedestrian in free space through

$$dx = v_x \cos(\theta_p) - v_y \sin(\theta_p)\Delta t,$$
$$dy = v_y \sin(\theta_p) + v_y \cos(\theta_p)\Delta t.$$

Here, v_x, v_y and θ_p are the pedestrians' x position, y position and orientation respectively.
In the next subsection, the POMDP model for collision avoidance from [Bai et al., 2015] will be briefly defined.

4.1.1 POMDP Model for Collision Avoidance

Based on the model for collision avoidance of [Bai et al., 2015], the description of the POMDP model $(\mathcal{S}, \mathcal{A}, T, r, \mathcal{O}, O, \gamma)$ is as follows:

1. The state in this model is a tuple, which consists of the position (x_c, y_c), velocity v_c and orientation θ of the car, and the position (x_p, y_p), velocity v_p and current goal g_p of the pedestrian. In addition as described in chapter 3 on page 14, the state also contains the type t of the pedestrian. The state space can then be represented as the set

$$\mathcal{S} = \{s \mid (x_c, y_c, v_c, \theta, x_p, y_p, v_p, g_p, t)\}.$$

2. An action will change the velocity of the car by a constant factor. This factor will act as an acceleration. There are three actions in the action space, which is the set

$$\mathcal{A} = \{\text{accelerate, hold, decelerate}\}.$$

3. The transition probability function T returns the probability of the car resulting in a successor state $s' \in \mathcal{S}$ when being in state $s \in \mathcal{S}$ and executing action $a \in \mathcal{A}$. The state transition behavior is modeled as described in equation (4.1) and equation (4.2) on the facing page.
4. The reward model $r(s, a)$ will return a positive or negative reward based on the present conditions of the environment. There are a total of four rewards:
   a) The car will get a negative reward c_{ped} if the pedestrian gets too close to it. This range will be denoted as the radius r_{ped} from the center of the car.
   b) The car will get another negative reward c_{acc} for accelerating and decelerating to prevent oscillated driving.
   c) In order to enhance smooth driving, the car will get a negative reward c_{speed} which is dependent of the current and the maximum velocity. This penalty is calculated as $c_{speed} = (v - v_{max}/v_{max})$.
   d) Finally, the car will get a positive reward c_{goal} for approaching its goal location. This reward will be granted, if the car gets in a certain range r_{goal} to the goal.
5. An observation o includes the position (x_c, y_c) orientation θ_c and velocity v_c of the car, and the position (x_p, y_p) of the pedestrian. The set of observation is then defined as

$$\mathcal{O} = \{o \mid (x_c, y_c, v_c, x_p, y_p, \theta_c)\}.$$

6. The observation probability function $O(s', a, o)$ returns the probability of observing o when resulting in state s' after executing action a. This function will model the noisy sensors of the car and the uncertainty in the pedestrians movement.
7. The discount factor $\gamma \in [0, 1)$ weights the immediate and distant rewards. A low discount factor means that the agent favors immediate rewards more than future rewards. A high discount factor will favor future rewards more than immediate rewards.

4.2 Implementation of the POMDP Solver

For solving the POMDP online, the DESPOT toolkit "Approximate POMDP Planning Online" [3] is used. This POMDP Solver is written in the C++ programming language. In order to define a specific problem to let the DESPOT solve it, a description of the problem needs to be written. The states of the POMDP will be represented by a class, containing all necessary information as described in subsection 4.1.1 on the previous page. The actions will be enumerated and therefore be represented as integer values. Observations are encoded as an unsigned 64 bit integers. This encoding will be described in subsection 4.2.3 on the facing page (*Representation of Observations*). The transition function T is internally implemented as the dynamics model described by equation (4.1) and (4.2). The observation probability function O models noisy sensor observations and therefore will return a biased probability value for observing $o \in \Omega$ given that action $a \in A$ takes the environment from state $s \in S$ to state $s' \in S$. Rewards will be taken into account in every time step based on the environmental state as discussed in subsection 4.1.1 on the previous page, item 4.

The general procedure is as follows: The DESPOT algorithm will start with an initial belief b_0. Based on this belief, an action will be computed and sent directly to the simulation environment. The simulation environment will execute this action and response with an observation, which will be received by the Belief Tracker. This Belief Tracker updates the belief given the observation and the previous prior belief according to Bayes' Rule from equation (2.4) on page 6. A general overview can be seen in figure 4.2.

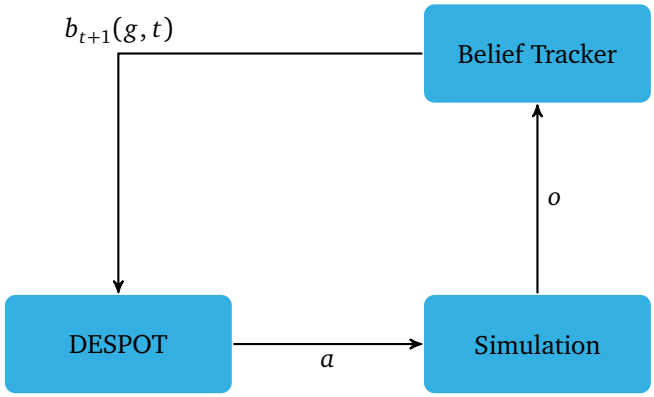


Figure 4.2: The general overview of the DESPOT framework including information exchange between the components. The DESPOT will send an action to the simulation based on the current belief $b_t(g,t)$. The agent executes the action, observes the environment and sends this observation o to the Belief Tracker. The Belief Tracker then calculates the new belief based on the action, observation and previous belief, and transmits the new belief to the DESPOT.

As the overall structure of the DESPOT framework is explained briefly, a more detailed description of the implementation will be given in the following subsections.

4.2.1 Value-based Boundaries for the Environment

For a real car, the lateral velocity is restricted to a maximum value due to physical and mechanical parameters [Meywerk, 2015]. To hold the simulation environment simple but also get appropriate results, the maximum velocity is set to 1.0 m/s. In addition, the size of the simulated map is a square with side 100 m. This leads to positional values bounded to 100 m in x- and y-axis. Because coordinates and velocities are continuous, a discretization has to be performed to keep the state and observation space discrete. In order to discretize a given number n_{raw}, equation (4.4) can be applied, where $d \in \mathbb{Q}_{>0}$ is the discretization value.

$$n_{\text{discretized}} = d \left\lceil \frac{n_{\text{raw}}}{d} \right\rceil \tag{4.4}$$

In this work, the positional values will be discretized to 0.2 m, velocities to 0.1 m/s and orientation values to 0.1 rad. Depending on the granularity of the discretized coordinates and velocities, d can be changed accordingly.

[3] https://github.com/AdaCompNUS/despot

4.2.2 Representation of Sates

As mentioned in subsection 4.1.1 on page 21, a state in the state space consists of the environmental position of the car and the pedestrian, as well as their instantaneous velocities. To handle the dynamics of the system, a state is represented as a class called `CarState`, which inherits from the generic class `State`. The code can be seen in listing 4.3.

```
1 class CarState : public State {
2 public:
3         geometry_msgs::Pose car_position;
4         geometry_msgs::Pose ped_position;
5         geometry_msgs::Point ped_goal;
6         double car_velocity;
7         double ped_velocity;
8 };
```

Figure 4.3: C++ code for representing a state for the POMDP model from subsection 4.1.1 on page 21. `Pose` is a ROS type for modeling an object in terms of its position and orientation in the environment with respect to the world frame. `Point` is also a ROS type, which only models the position of an object with respect to the world frame.

During the runtime of the DESPOT algorithm, a lot of states will be created and destroyed, which is expensive. To ensure performance, memory management has to be performed. The DESPOT framework provides a memory management technique, which creates a chunk of `State` objects rather than a single `State` object one at a time. When `State` objects are no longer needed, they will be put into a list instead of being deleted.

4.2.3 Representation of Observations

An observation is encoded as an unsigned 64 bit integer, internally defined as `OBS_TYPE`. One observation property will hold 8 bits. Because there is no way of representing a float value as an integer besides flooring or ceiling it, a multiplication has to be done for encoding the float value. Flooring or ceiling the value would lead to information loss. When decoding the observation, a division has to be performed. As mentioned in subsection 4.2.1 on the preceding page (*Value-based Boundaries for the Environment*) , position values will be discretized to 0.2 m, velocity to 0.1 m/ sec and orientation values to 0.1 rad. The order of the properties is as follows:

$$\underbrace{0_{63}\,0_{62}\,0_{61}\ldots}_{\text{unused}}\ \underbrace{0_{47}\ldots 0_{40}}_{\theta_c}\ \underbrace{0_{39}\ldots 0_{32}}_{v_c}\ \underbrace{0_{31}\ldots 0_{24}}_{y_p}\ \underbrace{0_{23}\ldots 0_{16}}_{x_p}\ \underbrace{0_{15}\ldots 0_{8}}_{y_c}\ \underbrace{0_{7}\ldots 0_{0}}_{x_c}$$

If there will be more observation variables for different traffic participants or different observation properties in the future, this encoding can be adapted easily and the unused bits 48 to 63 can be used.

4.2.4 Deterministic Simulative Model

The *Deterministic Simulative Model* is responsible for simulating the execution of an action a in state s for the DESPOT to sample scenarios. As mentioned in subsection 2.5.2 on page 8, a deterministic simulative model is a function $g : \mathcal{S} \times \mathcal{A} \times \mathbb{R} \mapsto \mathcal{S} \times \mathcal{O}$ that simulates one step of the environment given a state $s \in \mathcal{S}$ and an action $a \in \mathcal{A}$. In addition, the deterministic simulative model will calculate the reward of the agent when executing action a in state s, and makes a pseudo-observation $o \in \mathcal{O}$. In this work, the random number will not be used as it os not needed. In the DESPOT framework, the Deterministic Simulative Model is named as the `Step` function and returns a boolean value. This value indicates, whether the succeeding state s_{t+1} results in a terminal state.

In particular, the Step function handles the transition for the car and the pedestrian in the environment, depending on the given action. The car state transition is computed by executing action a_t on the car given the car dynamics from equation (4.1) and equation (4.2) on page 20. Executing an action on the car means either accelerate the car by a constant acceleration value, holding its velocity by applying no acceleration, or decelerating

the car by decreasing the velocity. Transitioning the pedestrian works analogously to the car transition model but with the goal given from the state s_t as the moving direction. Equation (4.3) on page 21 will compute the next position of the pedestrian. After processing both transitions, the function will create an observation from state s_{t+1} as described in subsection 4.2.3 on the previous page. Next, it will calculate the reward $r(s_t, a_t)$ resulting from executing action a_t in state s_t. Lastly, the function will return the boolean value `true`, if state s_{t+1} is a terminal state, and otherwise it will return `false`. The general procedure of the step function can be seen in algorithm 8.

Algorithm 8: Deterministic Simulative Model

Input: State $s_t \in \mathcal{S}$, Action $a_t \in \mathcal{A}$
Result: Next state s_{t+1}, Reward r_t, Observation o_{t+1}

begin
  $r_t \leftarrow 0$
  Process car dynamics given action a_t
  Process pedestrian dynamics given goal $\boldsymbol{g}$ within state s_t
  $o_{t+1} \leftarrow$ Observation from state s_{t+1}
  $r_t \leftarrow r(s_t, a_t)$
  return *True if s_{t+1} is a terminal state. False otherwise.*
end

4.2.5 Framework- and Simulation-Workflow

The whole framework starts by initializing the start positions of the car and the pedestrian, as well as the positions of all goals in the environment. After the car receives a goal position, the DESPOT will start to compute the optimal action. First, the initial belief b_0 will be initialized together with the particle vector. Based on this initial belief, DESPOT will compute an action a_0, which will be executed by the car. After executing a_0, the car receives an observation o_1. The belief can then be updated according to equation (3.4) on page 17. In this first time step, the last observation is non-existent. Therefore, the last observation will be the current observation, i.e. $o_0 = o_1$. After updating the belief and the particle vector, the procedure will continue by computing an optimal action a_t from current belief b_t and receiving observation o_{t+1} until the car reaches its goal or crashes with the pedestrian. The car will crash into the pedestrian, if it reaches the pedestrian within a range r_{terminal}. The same range applies for reaching the goal. Figure 4.4 shows the flow char of the whole framework.

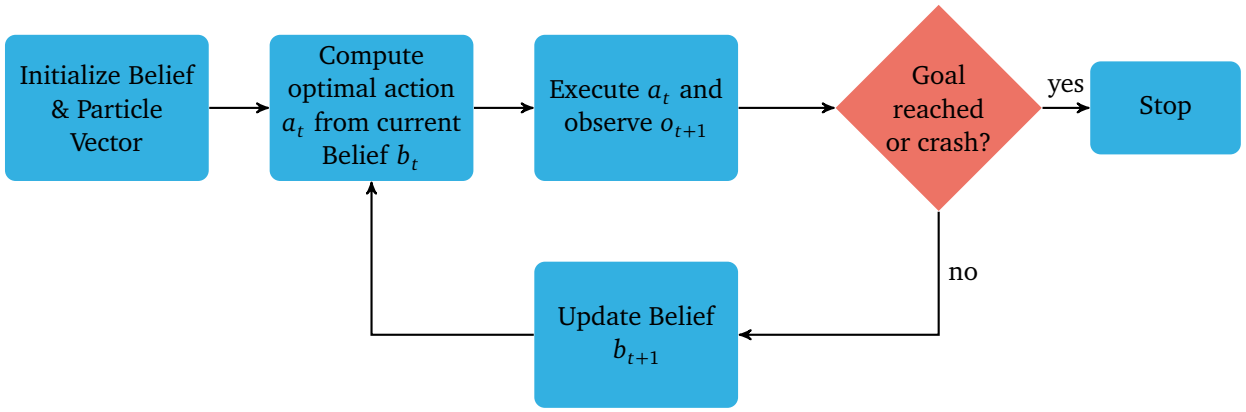


Figure 4.4: Workflow of the DESPOT framework together with the simulation. First, the belief and particle vector will be initialized. Based on this initial belief, DESPOT computes an optimal action and transmits it to the simulation environment. After executing action a, the vehicle checks if it reached the goal or crashed into the pedestrian. If so, the DESPOT stops immediately. Otherwise, it sends the observation o back to the DESPOT, which updates its belief. Based on the new belief, it computes the new optimal actions and proceeds.

5 Experiments

This chapter describes different simulation setups for the vehicle. Various experiments will be run for different positions of the pedestrian, as well as for the car. Each experiment will cover the belief and the velocity of the car while approaching its goal. These results will then be analyzed separately.

5.1 Pedestrian Oscillating between two Goals

In this experimental setup, the pedestrian will oscillate between two different goals with some Gaussian additive noise in his position. This oscillation will trace out a path, which the car will cross at some time. The car will start at position $(x_0, y_0) = (10, 19)$. The goal is placed behind the oscillating path of the pedestrian at position $(x_g, y_g) = (10, 35)$ such that the car has to cross this path. The setup can be seen in figure 5.1. The two spheres represent the goals of the pedestrian, namely goal 1 (left) and goal 2 (right). A red sphere indicates that this is the real goal of the pedestrian. The goal of the pedestrian changes, if he reaches the goal in a radius $r_{\text{reached}} = 1$.

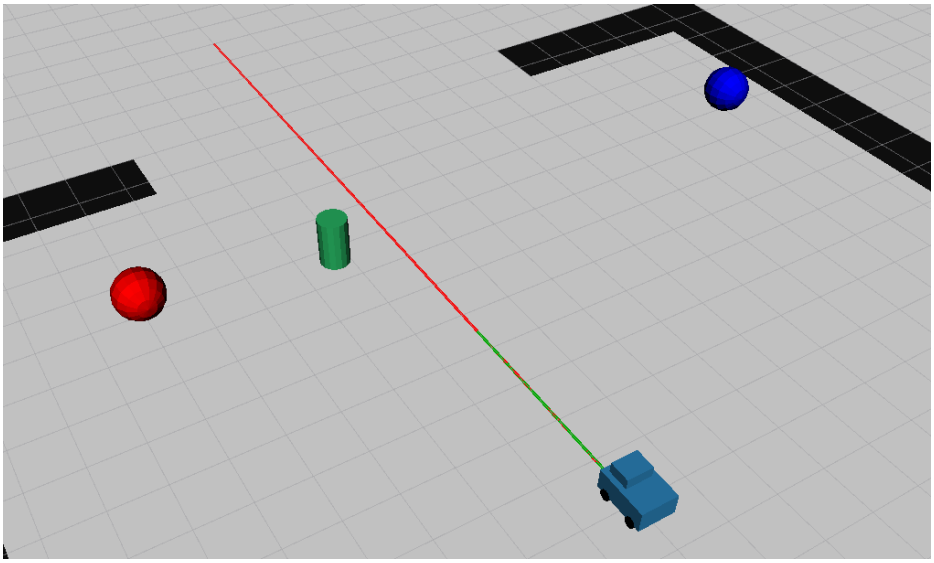


Figure 5.1: Experiment with the pedestrian oscillating between two goals, goal 1 (left) and goal 2 (right). The goal of the car is placed behind the oscillating path of the pedestrian, so that the car has to cross this path. A red sphere indicates the real goal of the pedestrian. The red line represents the global path of the car, which is computed by the path planner.

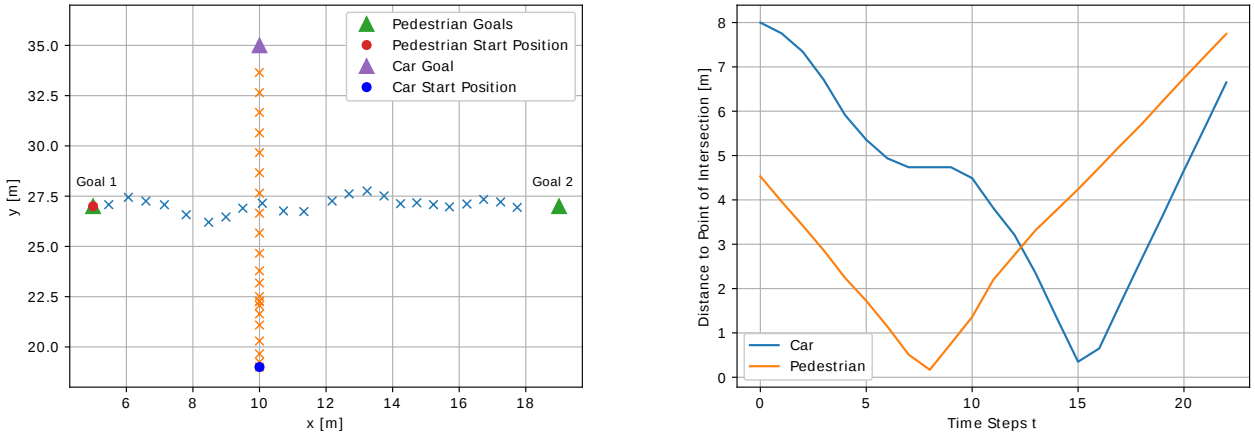
After the car initializes its goal, the pedestrian will start moving between the goals at a speed of 0.5m/s, and the car is starting to move towards its goal. The acceleration of the car is 0.2m/s^2. The probability for the observed pedestrian position from observation o_t being the calculated position is computed from the multivariate Gaussian probability density function by equation equation (3.3) on page 15. This multivariate Gaussian $\mathcal{N}\left(o_t \mid \boldsymbol{\mu} = \mathbf{x}_p, \boldsymbol{\Sigma} = I\sigma^2\right)$ has parameters o_t being the current observation, $\mathbf{x}_p$ being the calculated pedestrian position as shown in equation (3.2) on page 15, and $\sigma^2 = 1.0$ as the variance. For the DESPOT, the planning horizon is set to 40, the number of particles used is $K = 6000$, and the number of scenarios is 400.

5.1.1 Experiment 1: Pedestrian Moving Towards Goal 2

This setup will show the results for the pedestrian starting at goal 1 and moving at constant speed towards goal 2. As mentioned before, the car will start at position $(x_c, y_c)_{t=0} = (10, 19)$. The point of intersection of the pedestrian

and the car will be at position $(x_p, y_p)_{t=0} = (10, 27)$ with little Gaussian noise caused by the pedestrians' movement behavior.

Figure 5.2a shows the path of both the pedestrian and the car at each time step as a top-down view, together with the distance to their point of intersection. The pedestrian starts at goal 1 and approaches goal 2 in the planning cycle. As the goal of the car is initialized, the pedestrian starts to move towards goal 2, while the car starts computing the optimal action a to take. In figure 5.2a, the car starts to slow down, as the position stamps' density increases at $y \approx 22.5$. This can also be seen in figure 5.2b. At time step $t = 5$, the car stands still, as the distance to the intersection point is not changing. At that time, the pedestrian crosses this point. The car will start accelerating shortly after the pedestrian moved along. Finally, the car passes the point of intersection after time step $t = 15$.



(a) Paths with point of intersection.

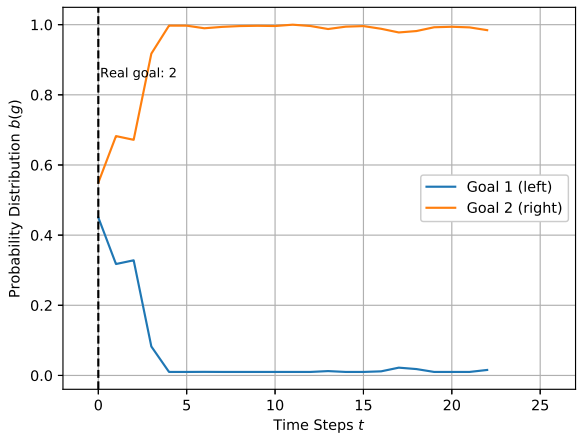
(b) Distances to point of intersection.

Figure 5.2: *Experiment 1:* Path of the car and the pedestrian at each time step, together with the distance to their point of intersection at $(10, 27)$. As the pedestrian approaches goal 2, the car decreases its speed to let the pedestrian pass the point of intersection. In figure (a), the car starts to slow down, indicated by the dense grouping at $y \approx 22.5$. This can also be seen in figure (b). At time step $t = 5$ the car stands still to let the pedestrian pass the intersection point. As the pedestrian crosses this point, the car starts to accelerate, crossing the point of intersection at $t = 15$.

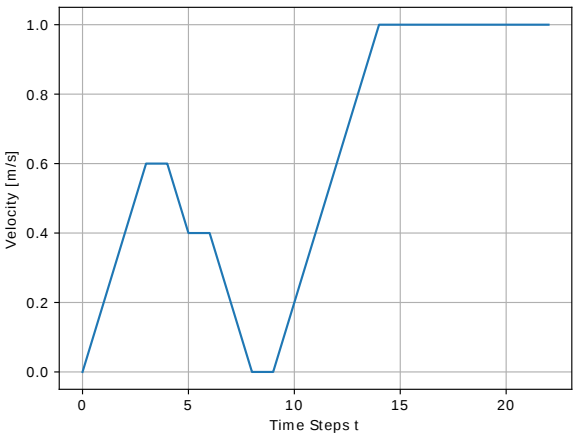
Figure 5.3 on the next page shows the belief $b_t(g)$ of the car over goal 1 and goal 2 at each time step t. The initial belief $b_0(g)$ is set uniformly, thus $b_0(g) = (0.5, 0.5)^\mathsf{T}$. In the beginning, the car starts to increase its velocity. The belief over goal 2 at this duration is about to rise, which is an indicator for the car to decrease its velocity. At time step $t = 3$, the belief for goal 2 is increasing steeply. In the succeeding time step, the car is fully certain about the real goal of the pedestrian, resulting in a deceleration additionally caused by the pedestrian moving towards the intersection point. This happens about 5m before the intersection point, leading to a safe driving behavior.

5.1.2 Experiment 2: Pedestrian Moving Towards Goal 1

In this experiment, the pedestrian will start at goal 2 and approach his way to goal 1, while the cars' starting and goal position will stay as it is. The parameters for the noise in the pedestrian model will now change in order to see what impact they have on computing the belief and the optimal action. Therefore, the variance for the pedestrian model will be $\sigma^2 = 0.5$. With the initialized goal, the experiment ran 34 time steps. The path can be seen in figure 5.4 on the facing page. The Manhatten-Distance from the car to the pedestrian is now higher than in the previous experiment. We can see that there are two small chunks of dense position stamps at $y \approx 20$ and $y \approx 22.5$. This indicates a rapid deceleration after the car started to drive on its path. We can verify this by looking at the velocity curve in figure 5.5b on page 28. At $t = 3$, the car starts to decelerate, then accelerate for 2 time steps, and decelerating again until complete stop at time step $t = 10$. The car then waits for the pedestrian to pass the point of intersection, so that it can start driving towards its goal again at $t = 19$, reaching it after 34 time steps. The reason the car took longer to reach it goal is because it was standing still longer, than in the first experiment.



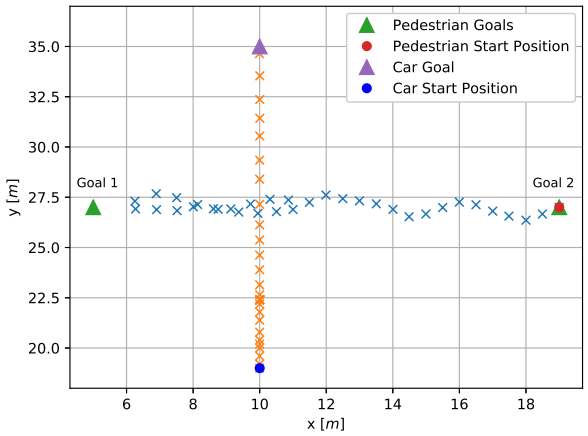
(a) Belief over goals over time.



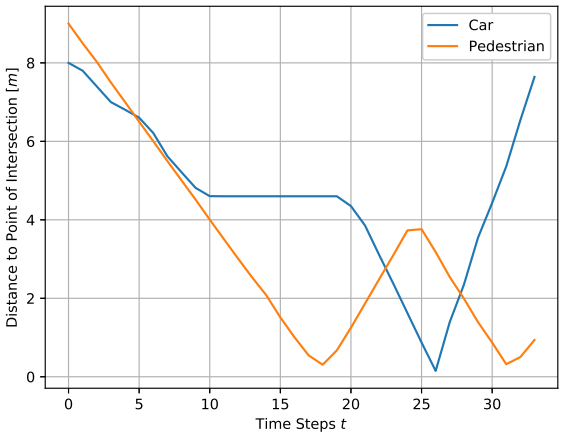
(b) Velocity of the car over time.

Figure 5.3: *Experiment 1:* Belief and velocity over time. The black dashed line indicates the real goal of the pedestrian from this time step onwards. At time step $t = 4$, the car is certain about the goal of the pedestrian, being goal 2 and starts to decelerate in order to avoid a crash at the intersection point. The car reaches its goal after 23 time steps.

A look into figure 5.5a on the following page shows that the belief has more noise in the first 10 time steps. The car is uncertain about the pedestrians' goal at this duration, until the pedestrian gets closer to its goal. This uncertainty is a consequence of the noise in the observation probability and the noise in the pedestrian model. After the pedestrian reaches goal 1, it heads back to goal 2 at time step $t = 25$. The car notices this change and updates the belief, resulting in an increase for the belief over goal 2.



(a) Paths with point of intersection.

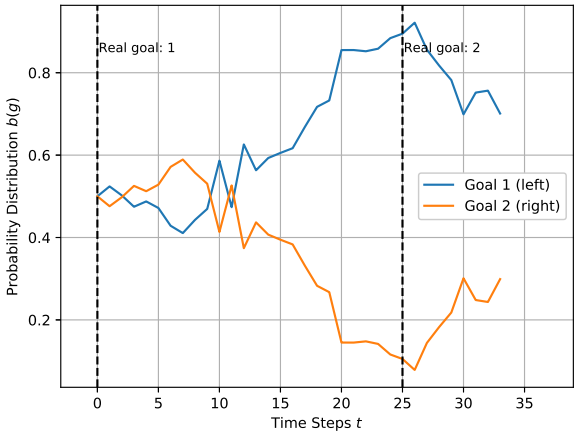


(b) Distances to point of intersection.

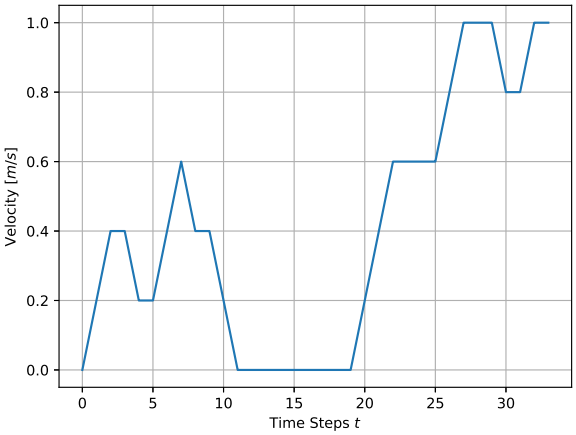
Figure 5.4: *Experiment 2:* The path and distance to the point of intersection for both the pedestrian and the goal. The pedestrian starts at goal 2 and approaches goal 1. The dense collection of pose stamps in figure (a) at $y \approx 20$ and $y \approx 22$ indicates that the car slowly approached its goal.

5.1.3 Experiment 3: Fixed Model Parameters with varying Pedestrian Movement Noise

In this experiment, the visual setup will stay fixed like in experiment 1, meaning the starting position of the car, the starting position of the pedestrian and their goals will stay the same. The car will approach its goal 4 times, while the model parameters of the observation probability model will be fixed, but the noise in the pedestrians



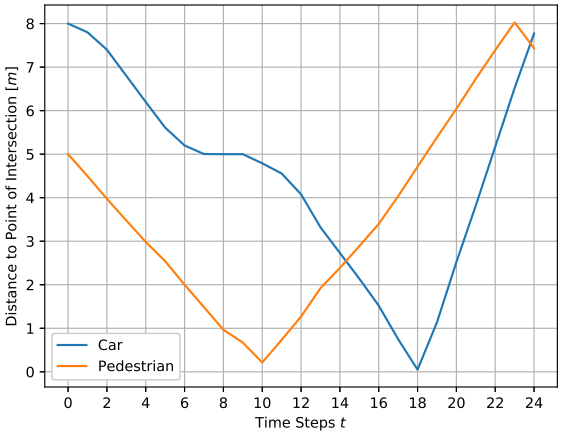
(a) Belief over goals over time.



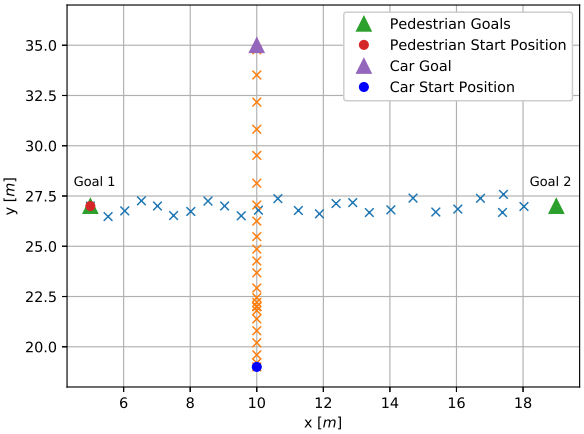
(b) Velocity of the car over time.

Figure 5.5: *Experiment 2:* The belief and velocity over time for the pedestrian approaching goal 1. The belief shows more noise than in the first run of the experiment. Also, the pedestrian changed the goal after reaching goal 1.

real movement will vary. The objective of this experiment is to analyze the effect of the movement noise of the pedestrian on the accelerating and decelerating behavior of the car. Therefore, the target figures will be the comparison of the distances from the pedestrian and the car to the point of intersection, and the path of the car and the pedestrian at every time step. The noise for the pedestrians movement is modeled by letting the pedestrian oscillate in y direction with varying amplitudes. A noisy movement is therefore a path from the pedestrian to its goal, which is not the optimal path. An optimal path will always be a straight line directed to the real goal of the pedestrian. The amplitudes for the pedestrians' noisy movement are 0.5m, 1m, 2m and 4m.



(a) Distance for car and pedestrian to the point of intersection.

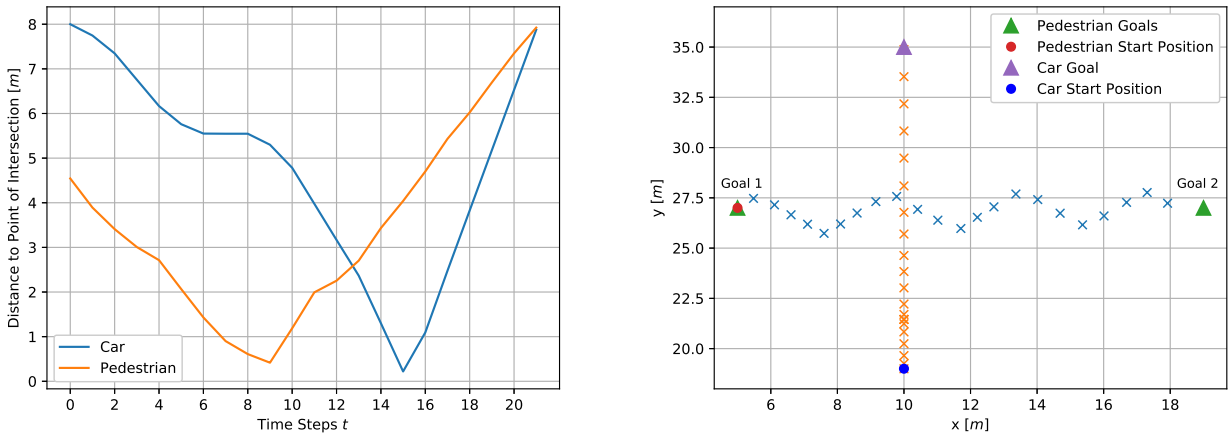


(b) Path of the car and the pedestrian at every time step.

Figure 5.6: *Experiment 3:* The distance graph of the car and the pedestrian, together with their path at every time step for a pedestrian noise of 0.5m.

In the first run of the experiment, the noise of the pedestrians movement is set to 0.5m. Figure 5.6 shows the graph of the distances from the pedestrian and the car to the point of intersection, and the top-down view of the environment with the path of the pedestrian and the car. Here, the point of intersection is at $y \approx 27$. Looking at figure 5.6a, we can see that the pedestrian crosses the point of intersection at time step $t = 10$. The car starts to decrease its velocity at $t = 5$ and stopping fully at $t = 7$. At this time step, the car waits for the pedestrian to cross the cars' path. Shortly after the pedestrian passes the intersection point, the car starts accelerating, passing the

point of intersection at time step $t = 18$. The car successfully predicted the pedestrians movement towards goal 2 and started decelerating early enough for the pedestrian to cross the path of the car. Figure 5.6b shows the noisy movement of the pedestrian. At $y \approx 22$ the car starts to decrease, which is indicated by the dense collection of position stamps at that area.



(a) Distance for car and pedestrian to the point of intersection. **(b)** Path of the car and the pedestrian at every time step.

Figure 5.7: *Experiment 3:* The distance graph of the car and the pedestrian, together with their path at every time step for a pedestrian noise of 1m.

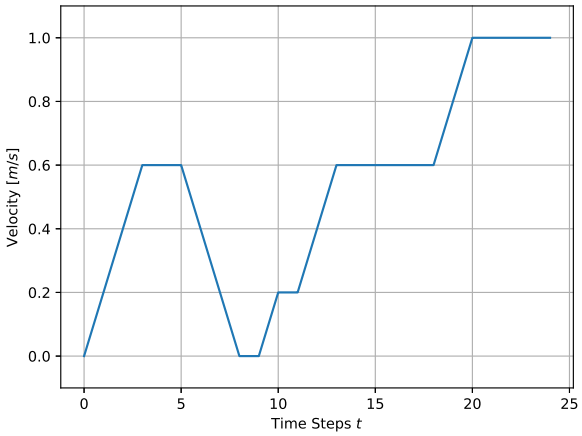
Now, the movement noise of the pedestrian is increased to 1m. This noisy movement can be seen in figure 5.7b. The pedestrian crosses the intersection point at approximately $t = 8$. This approximate value is caused by the car observing the time of intersection to late. As we can see in figure 5.7a, the car starts to decrease its velocity at time step $t = 4$ and fully stopping at $t = 6$. It waits 3 time steps for the pedestrian to cross the point of intersection before it increases its velocity again. In this run, the car started decreasing one time step earlier than in the previous run. The dense collection of position stamps at $y \approx 21$ in figure 5.7b also shows the deceleration at that position and is indeed earlier than in the first run.

In both runs, the car waited 2 time steps with a velocity of $v = 0$ for the pedestrian to cross the point of intersection. In the succeeding step, it started accelerating again until it reached it goal. In contrast to the first run, which lasted 24 time steps, the second run lasted 21 time steps. If we take a look at the cars' intersection time with the point of intersection in the first run, we can see that it reached this point at $t = 18$, whereas in the second run, it reached it 3 time steps earlier, at $t = 15$. This indicates that the car either held its velocity longer in the first run, or accelerated one more time in the second run. Figure 5.8 on the next page shows the velocity profile of the car in these two runs. In figure 5.8a, the car held its velocity 2 time steps longer after crossing the point of intersection.

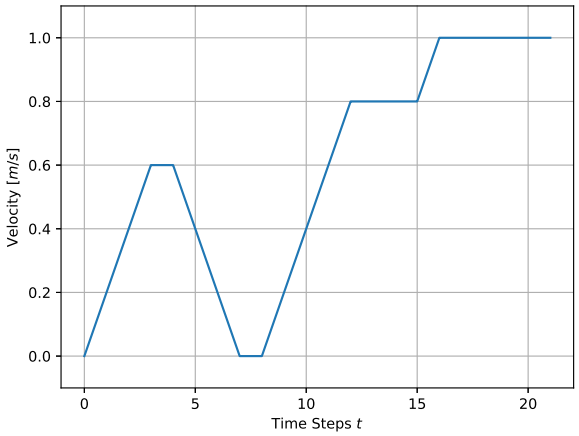
The result of the next run is shown in figure 5.9 on the following page, in which the noise of the pedestrian's movement is increased to 2m. As figure 5.9a shows, the car is not fully stopping at any time. The car is starting to decelerate at time step $t = 2$. From that time onwards, it holds its velocity until $t = 9$. One time step earlier, the pedestrian crossed the point of intersection, which now is at approximately $y \approx 27.5$, shown in figure 5.9b. This run shows that the car is careful at the beginning, not to accelerate to much, as the pedestrian moves non-optimal towards its goal. After the car reaches the intersection point at time step $t = 18$, it ends the run in a total of 25 time steps.

In the last run, the movement noise of the pedestrian is set to 4m, which can be seen in figure 5.10b on page 31. The pedestrian oscillates two times in y direction before it reaches goal 2. A look into the distance graph for both the car and the pedestrian in figure 5.10a on page 31 shows that the car starts with a low velocity and holds it low until the pedestrian is about to cross the point of intersection at time step $t = 6$. The point of intersection is now at $y \approx 30.5$ due to the movement noise of the pedestrian. At time step $t = 5$, the car starts to accelerate. This behavior could be explained through the large distance between the car and the pedestrian. The car is gradually increasing its velocity after the pedestrian passed the intersection point. A total of 16 time steps are enough for the car to reach the goal, which is the fastest run in this experiment.

Table 5.1 on page 31 summarizes the timing of the car in each run and lists the time steps for the first deceleration, stopping time and duration, as well as the time of re-acceleration. We can see, that the starting time

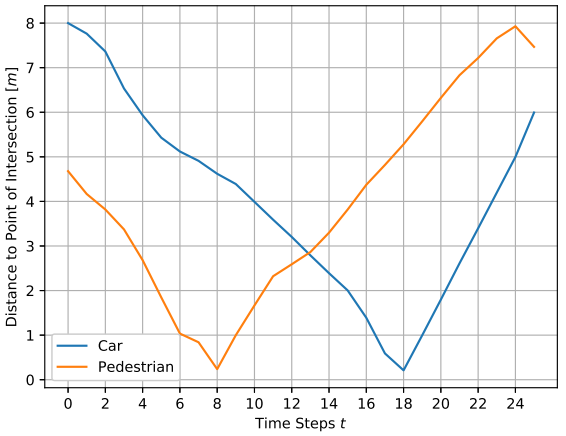


(a) Velocity profile of the car with pedestrian noise of $0.5\mathrm{m}$.

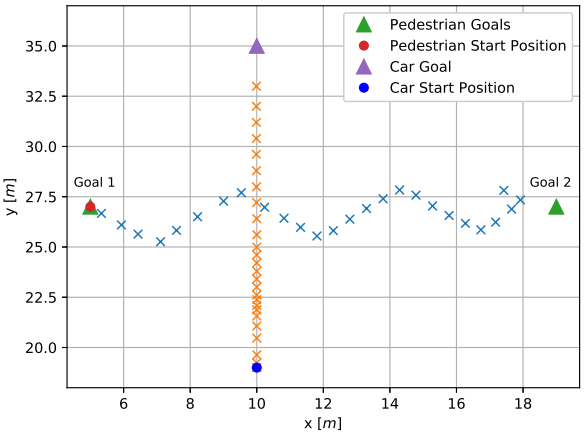


(b) Velocity profile of the car with pedestrian noise of $1\mathrm{m}$.

Figure 5.8: *Experiment 3:* Velocity profile of the car with pedestrian noise of $0.5\mathrm{m}$ and $1\mathrm{m}$. In the first run, the car held its velocity 2 time steps longer than in the second run.

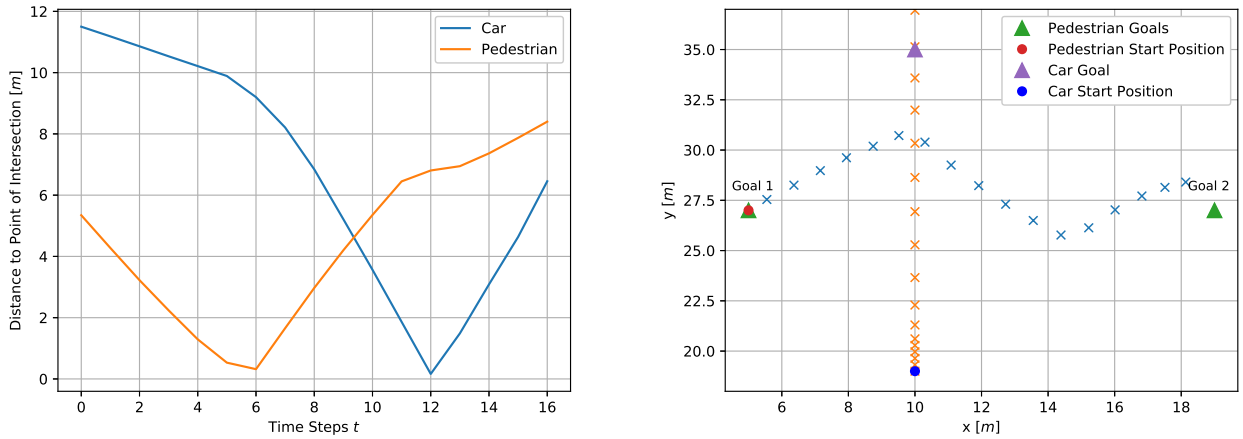


(a) Distance for car and pedestrian to the point of intersection.



(b) Path of the car and the pedestrian at every time step.

Figure 5.9: *Experiment 3:* The distance graph of the car and the pedestrian, together with their path at every time step for a pedestrian noise of $2\mathrm{m}$.



(a) Distance of car and pedestrian to the point of intersection. **(b)** Path of the car and the pedestrian at every time step.

Figure 5.10: *Experiment 3:* The distance graph of the car and the pedestrian, together with their path at every time step for a pedestrian noise of 4m.

of decelerating is decreasing with increasing pedestrian movement noise. This indicates the uncertainty in the pedestrians behavior, as the path of the pedestrian towards its goal is non-optimal.

Run no.	Start of decelerating	Fully stopping	Duration of stop	Start of accelerating
1	$t = 5$	$t = 7$	2	$t = 10$
2	$t = 4$	$t = 6$	2	$t = 9$
3	$t = 2$	–	–	$t = 9$
4	$t = 1$	–	–	$t = 5$

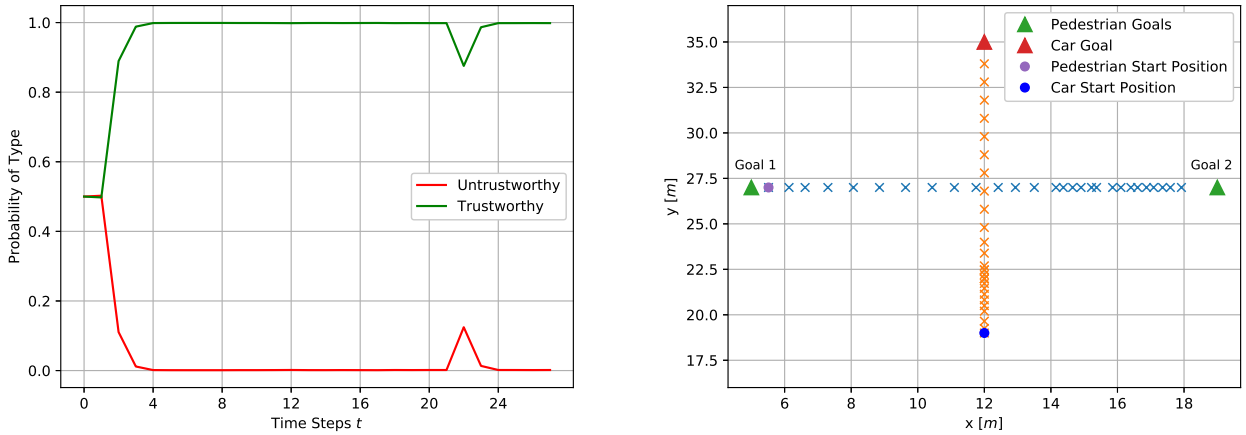
Table 5.1: Summarized timings of the runs with differing pedestrian noise. The second column shows the time step at which the car firstly starts to decelerate, followed by the time step at which the car fully stops. The fourth column shows the duration of the stop. The last column gives the time step at which the car starts to accelerate again.

5.2 Type-based Pedestrian Oscillating between two Goals

Now that we analyzed the behavior of the car during the planning cycle, the extension to the belief with types will be analyzed as well. The type-based classification of the pedestrian will not infer the online policy search of the POMDP solver. In this experiments, the target types of the pedestrian are t_1 = untrustworthy and t_2 = trustworthy. As mentioned in chapter 3 on page 14, these types can be modeled by changing the variance of the observation model from equation (3.3) on page 15. For type trustworthy, the variance will be set to a small value. In this experimental setup, the variance for the pedestrian being trustworthy is set to $\sigma_t^2 = 0.05$. For type untrustworthy, the variance will be set to a slightly larger value. Here, it will be $\sigma_u^2 = 0.5$. The car will start at position $(x_c, y_c)_{t=0} = (12, 19)$. Its goal will be located at $(12, 37)$. The pedestrian will start at the left goal and move towards goal 2.

5.2.1 Experiment 1: Pedestrian Approaching Goal Optimally

In this experiment, the pedestrian will approach its goal directly through an optimal path with no noise. The belief over types $b(t)$ should therefore be trustworthy. Looking at figure 5.11a, we can see that the type is trustworthy over the whole planning cycle.



(a) The belief over types of the pedestrian at every time step.

(b) Path of the car and the pedestrian at every time step.

Figure 5.11: Belief over types of the pedestrian together with the path of the pedestrian and the car at every time step. The pedestrian approaches its goal directly through an optimal path with no noise.

At time step $t = 22$, a little peak can be seen, indicating a little uncertainty of the type "trustworthy". If we look at figure 5.12, the real goal of the pedestrian changes at time step $t = 20$. Shortly after, the car notices this change and updates the belief accordingly. This change affects the belief over the types because the pedestrian model assumes the pedestrian would stand still at goal 2. Instead, the pedestrian changes its heading, walking back towards goal 1. This results in an "untrustworthy" behavior, which is indicated by the little peak in the belief over types in figure 5.11a.

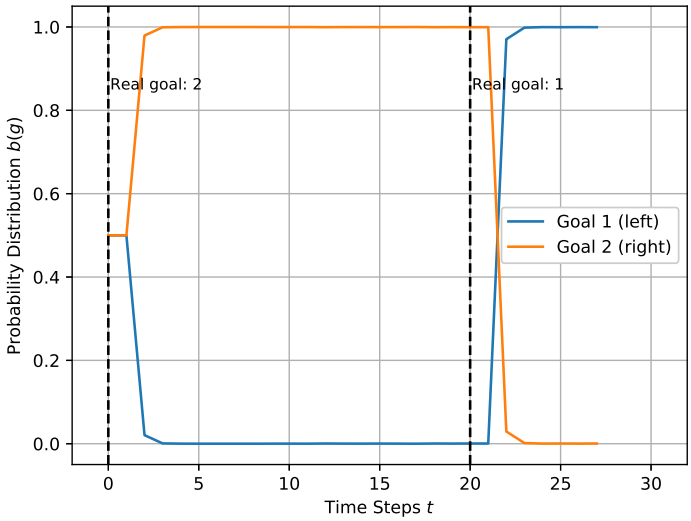
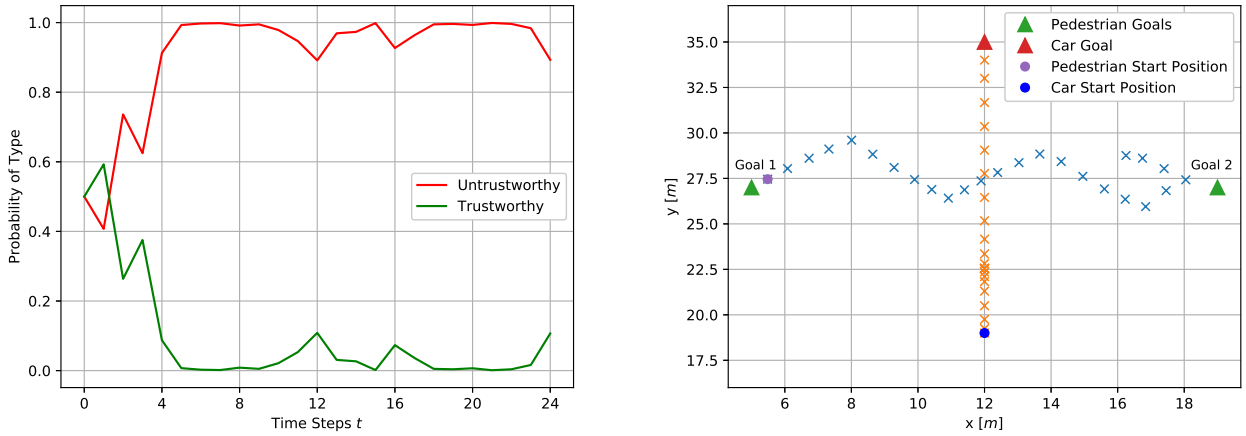


Figure 5.12: Belief over goals of the pedestrian at every time step. The pedestrian changes its goal at time step $t = 20$, which is been noticed by the car 3 time steps afterwards, indicated by the increasing belief of goal 1.

5.2.2 Experiment 2: Pedestrian Approaching Goal Non-Optimally

Now the pedestrian will move towards goal 2 with noise in its movement. Specifically, the pedestrian will oscillate in y-direction approaching its goal non-optimally. This will model the type "untrustworthy". Figure 5.13 shows the belief over types and the path of the pedestrian and the car at every time step.



(a) The belief over types of the pedestrian at every time step. **(b)** Path of the car and the pedestrian at every time step.

Figure 5.13: Belief over types of the pedestrian together with the path of the pedestrian and the car at every time step. The pedestrian approaches its goal non-optimally with some noise in its movement.

Looking at figure 5.13a, we can see that shortly after the pedestrian starts moving towards its goal, the belief over type "untrustworthy" increases. As in the previous experiment, there are little peaks in the distribution. This time at time step $t = 12$, $t = 16$ and $t = 24$. This might be the result of the pedestrian walking directly towards the goal at some points on its path. Nevertheless, the car classified the pedestrian successfully as "untrustworthy" at the whole planning cycle.

5.2.3 Multiple Runs with varying Noise

This subsection will briefly show the results of multiple experiments with the same setup as in the previous experiment except with varying seeds for the sampled noise. This means, that the start position of the car and the pedestrian will stay the same, as well as their goals. First, the pedestrian will move with little noise towards its goal, but varying random seed. Then, the noise will be increased. Each experiment will run 4 times. We will then capture the mean and variance of the belief over types for all runs. Table 5.2 shows the mean and variance over the 10 experiments for each type for little noise. Here, the noise is sampled from a Gaussian with mean $\mu_n = 0$ and variance $\sigma_n^2 = 0.5$. An example for the pedestrian path with little and high noise can be seen in figure figure 5.14 on the following page. μ is the mean of the belief over the type throughout the planning time. The standard deviation σ denotes the spread of the belief over type from the mean.

	Run 1		Run 2		Run 3		Run 4		Average	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Untrustworthy	0.058	0.122	0.096	0.279	0.097	0.161	0.074	0.114	0.081	0.169
Trustworthy	0.941	0.122	0.903	0.279	0.902	0.161	0.925	0.114	0.917	0.169

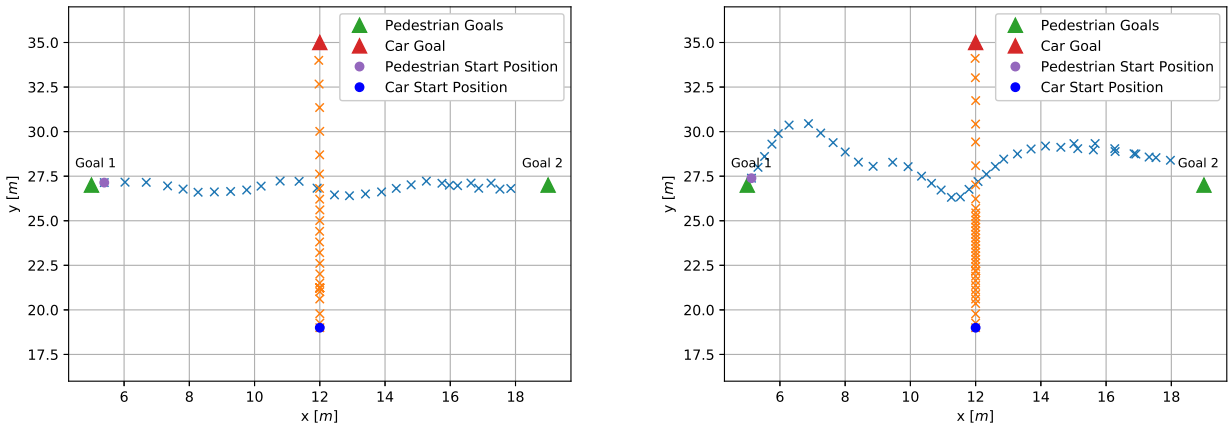
Table 5.2: Mean and standard deviation of the belief over types for the pedestrian moving with little noise. Here, μ is the mean of the belief over the type throughout the planning time. The standard deviation σ denotes the spread of the belief over type from the mean.

We can see, that for little noise, the pedestrian is classified as *trustworthy* with mean $\mu = 0.917$. With a standard deviation of $\sigma = 0.169$, the variability of the belief is low.

For high pedestrian noise $\sigma^2 = 2.0$, we observe from table 5.3 that the belief over types will tend to favor the type *untrustworthy*. With a mean over type *untrustworthy* of $\mu = 0.869$ the pedestrian will be classified as *untrustworthy*.

	Run 1		Run 2		Run 3		Run 4		Average	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Untrustworthy	0.804	0.279	0.877	0.273	0.874	0.259	0.924	0.205	0.869	0.254
Trustworthy	0.195	0.279	0.122	0.273	0.125	0.259	0.075	0.205	0.129	0.254

Table 5.3: Mean and standard deviation of the belief over types for the pedestrian moving with higher noise. With the pedestrian being noisy in his movement, the belief has a larger standard deviation from the mean. Overall, the car classifies the pedestrian as *untrustworthy* as the mean for type *untrsutworthy* is much higher than for type *trustworthy*.



(a) Path of the pedestrian with little noise.

(b) Path of the pedestrian with high noise.

Figure 5.14: Path of the pedestrian with little and high noise in its movement.

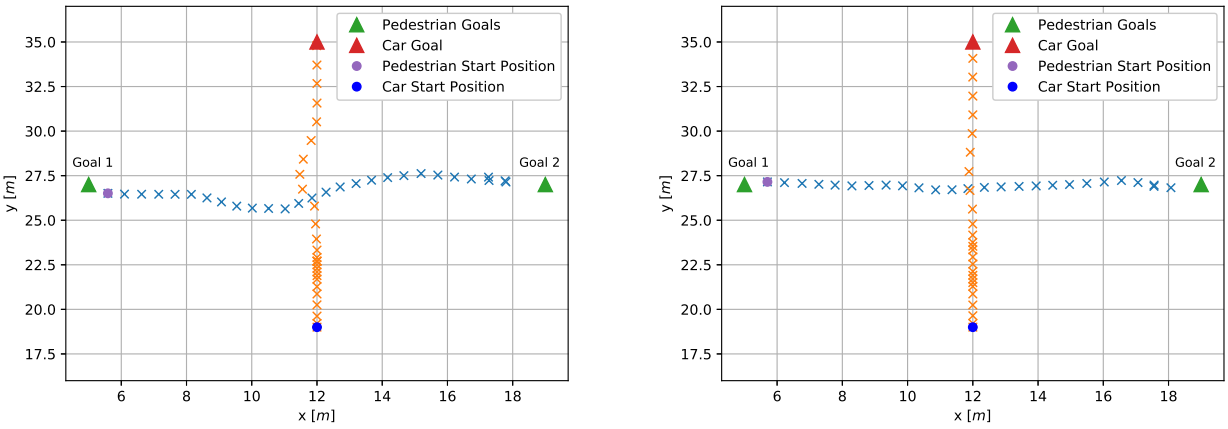
5.3 Type-based DESPOT Planning

In this experiment, the type will be part of the inner DESPOT transition model of the pedestrian as part of the deterministic simulative model described in subsection 4.2.4 on page 23, together with replanning the car's path every 5th time step. In the previous experiments, we only considered the type to be part of the observation model. Now, DESPOT will also compute the optimal policy with noise in the simulated pedestrian movement based on the type. In particular, if the type of the pedestrian in the simulated state is *trustworthy*, the pedestrian will have additive noise in its movement of $\sigma_n^2 = 0.01$. Otherwise, it will have additive Gaussian noise with variance $\sigma_n^2 = 0.5$. We will start with a low noise on the real pedestrian movement and varying random seeds. As in the previous experiment, we will take the mean of the beliefs for each type. For each noise range, the experiments will run 3 times in total. Table 5.4 on the next page summarizes the means and standard deviations for the pedestrian type with low noise.

In addition, figure 5.15 on the facing page shows two example paths from the environment. The left plot shows a bend on the car's path. This bend indicates the replanning of the local path of the car, caused by the timing and the pedestrian's position at that timing. As the pedestrian walks towards goal 2, the path will bend towards the opposite direction. Looking at the plot in figure 5.16 on the next page for the belief over types, we can see that the car is classifying the pedestrian as *trustworthy* throughout the run.

	Run 1		Run 2		Run 3		Average	
	μ	σ	μ	σ	μ	σ	μ	σ
Untrustworthy	0.029	0.097	0.022	0.097	0.023	0.089	0.024	0.094
Trustworthy	0.970	0.097	0.977	0.097	0.976	0.089	0.974	0.094

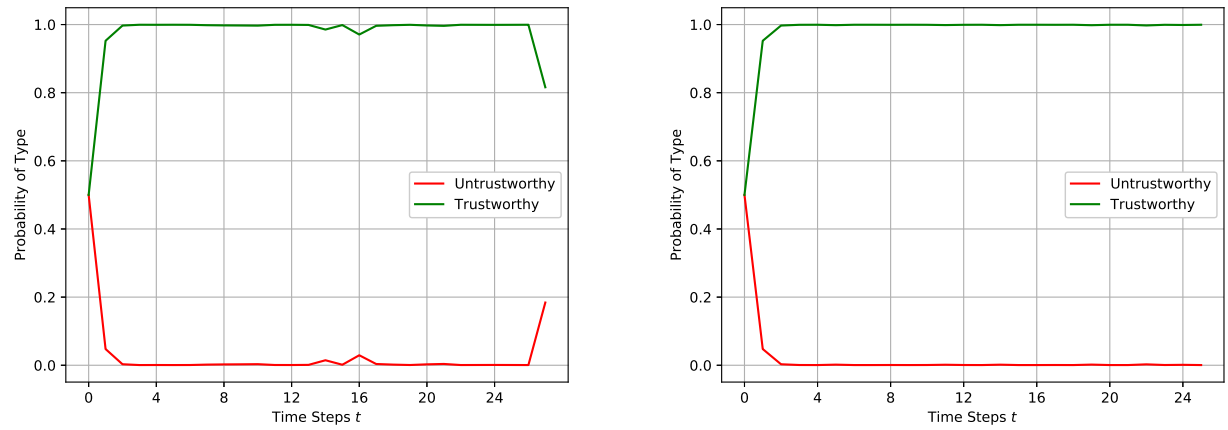
Table 5.4: Mean and standard deviation of the belief over types for the pedestrian moving with low noise for every run.



(a) Path of the car and the pedestrian with little noise.

(b) Path of the car and the pedestrian with little noise.

Figure 5.15: Two example paths of the pedestrian with little noise in its movement. The left path shows the car replanning its path based on the timing and pedestrian position along the path.



(a) Belief over types for the pedestrian with little noise.

(b) Belief over types for the pedestrian with little noise.

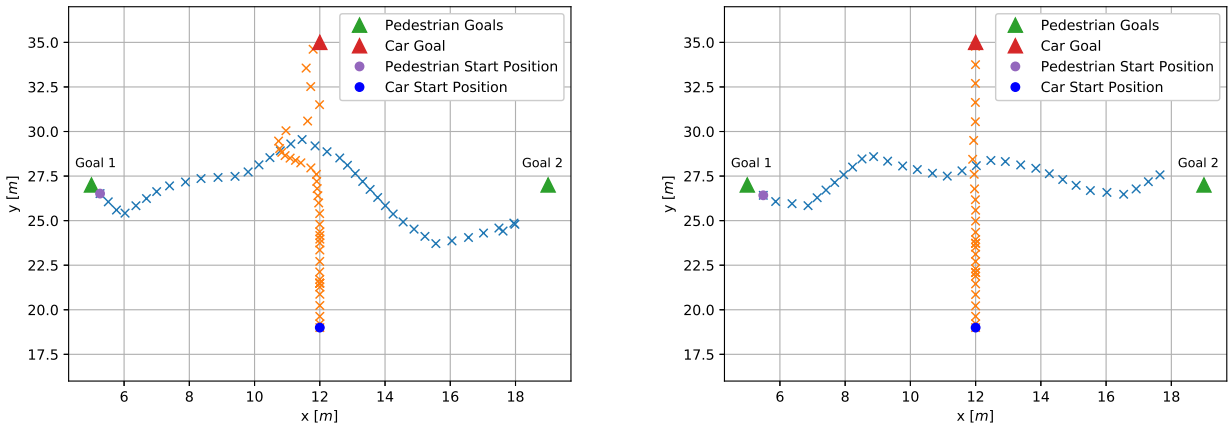
Figure 5.16: Two example plots for the belief over types with little pedestrian noise. The probability for the pedestrian being trustworthy is high throughout the run. In figure 5.16a we can see the belief changing at the end. At this time, the pedestrian changed its goal, while the model expected the pedestrian to stand still.

In contrast, we will now look at the car approaching its goal when the pedestrian is moving within a higher noise range. Table 5.5 shows the average mean and variance over the type belief from all 3 runs. We notice that in run 3, the pedestrian is classified as *trustworthy*, although the noise in its movement was higher than in the last experiments. The corresponding path-plot can be seen in figure 5.17b. The pedestrian moves randomly in the beginning, but optimizes its path halfway through the run. This behavior lead to the increase in trust. Looking at the velocity profile in figure 5.18b on the facing page for this run, the car comes to a full stop, being uncertain about the pedestrians behavior. After the pedestrian crosses the car's path, the car speeds up and approaches its goal.

	Run 1		Run 2		Run 3		Average	
	μ	σ	μ	σ	μ	σ	μ	σ
Untrustworthy	0.883	0.245	0.661	0.366	0.193	0.320	0.579	0.310
Trustworthy	0.116	0.245	0.338	0.366	0.806	0.320	0.42	0.310

Table 5.5: Mean and standard deviation of the belief over types for the pedestrian moving with high noise. In the third run the car classifies the pedestrian as *trustworthy,* although the pedestrian was moving with high noise.

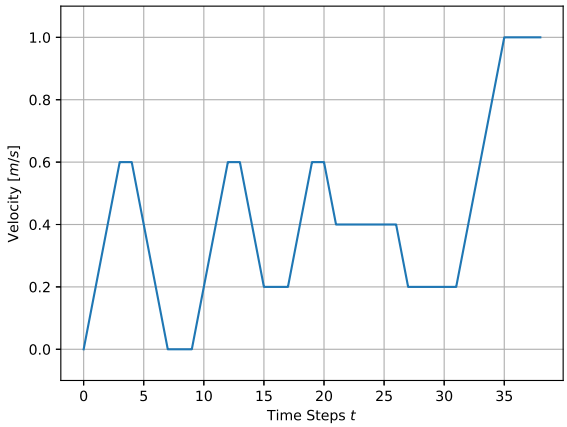
Figure 5.17a shows the path of the second run in this experiment. The pedestrian is classified as *untrustworthy* with mean $\mu = 0.661$. Looking at this path plot, we can see that the pedestrian moves with high noise after crossing the car's path. Also, the car replanned its path while the pedestrian was crossing the path. It bend to the left again so it could avoid the pedestrian. The velocity profile at figure 5.18a on the facing page shows that the car slows down 3 times before approaching its goal without a velocity decrease.



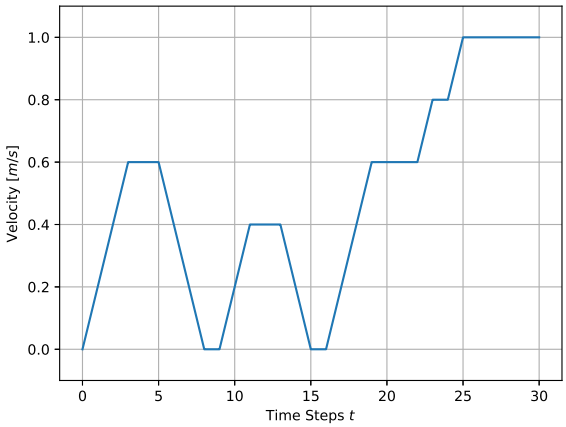
(a) Path of the car and the pedestrian with higher noise.

(b) Path of the car and the pedestrian with higher noise.

Figure 5.17: Two example plots for the path with higher pedestrian noise. The car is replanning its path in the left plot, while in the right plot, it is able to drive towards its goal without replanning. Also, the movement noise of the pedestrian in the left plot is more random than in the right plot.



(a) Velocity of the car for the pedestrian with higher noise.



(b) Velocity of the car for the pedestrian with higher noise.

Figure 5.18: Two example plots for the velocity profile of the car with higher pedestrian noise. The left and right plot match the plot from figure 5.17a on the preceding page and figure 5.17b on the facing page respectively. In the left plot, we can observe that the car slows down multiple times. This decrease of speed indicates the uncertainty in the pedestrians movement and type.

6 Conclusion and Future Work

6.1 Conclusion

In this work, we briefly showed the POMDP model for collision avoidance for autonomous vehicles and defined the extension to model the trustworthiness of pedestrians. First, we defined the concept of MDPs and POMDPs, and how they can be used to solve specific problems. Additionally, we looked at methods for solving both decision processes. One of these methods is the DESPOT algorithm, which has been used to plan actions for the car. We looked at the belief model for this problem, which was represented as the belief over goals of the pedestrian. This belief has to be maintained by the car to know in which state the car is in approximately. Thus, the belief has to be updated. We derived the update equation for the belief over goals in subsection 3.2.1 on page 14 and explained the way it is used in the POMDP model, i.e., by a particle vector. The general setup and implementation of the DESPOT and the simulation environment, as well as the workflow of the DESPOT framework has been described and illustrated. In addition to the belief over goals, we extended the model by types. Pedestrians are classified into types *trustworthy* and *untrustworthy*, modeling the trustworthiness. Generally, the pedestrian can be classified as any pre-defined type, as long as there is an appropriate model for this type. Afterwards, we introduced and derived the update equation for this belief representation.

Having the formal foundations discussed, experiments showed the car and the pedestrian acting in the simulation environment. First, experiments without the type classification were run where the pedestrian oscillated between two goals. We increased the noise in the pedestrians' movement for each run to see the influence of this change on the car's behavior. The next set of experiments included the type classification of the pedestrian. Here, the pedestrian again oscillated between two goals, starting from one goal. To analyze the belief over types, the pedestrian was assigned no noise in its movement indicating a "trustworthy" pedestrian. In the succeeding experiment, noise was added to the pedestrians movement in terms of oscillation in y-direction. This modeled the "untrustworthy" pedestrian. In both cases, the car classified the pedestrian correctly as "trustworthy" and "untrustworthy" respectively. In all experiments, the car successfully avoided a collision with the pedestrian by predicting the next positions of the pedestrian with some belief and decelerated ahead of time.

6.2 Future Work

In this thesis, we only looked at the pedestrian as the target traffic participant in order to avoid collisions. However, the model for the traffic participant can be exchanged by arbitrary models. This model could represent another or a cyclist. These models can also be more complex, depending on the knowledge of the domain or the model. Additionally, the model of the agent itself can be exchanged, for example with the model of an aircraft. This aircraft would then avoid in the air while flying [Wolf, 2009].

There could also be situations, where the traffic participant is not visible to the vehicle at all. This could be the case when a wall or an object is between the vehicle and the traffic participant. For those scenarios, the vehicle has to be aware of possible collisions, even if there are currently no other traffic participants, or they are hidden to the vehicle. This problem was discussed and tackled by Brechtel et al. in their work [Brechtel et al., 2014].

Right now, the observations made by the vehicle are done passively, meaning that the car only observes the observable information about the traffic participant and does not execute actions that are intended to get a reaction from them. Instead of this passive observing behavior, the car could try executing actions and observe the reaction from the traffic participant in order to build a more richer model [Sadigh et al., 2016]. In addition, the agent can learn a more complex pedestrian model based on the pedestrians' movement and behavior [Scovanner and Tappen, 2009]. Not only can a pedestrian model be learned with this method, but also a model for cyclists or other traffic participants.

The concept of avoiding collisions with dynamic obstacles could also be applied to robotic arms e.g., in an industrial environment, where humans collaborate with robots. To avoid collisions with humans, the robot observes the movements of the human and builds the belief over the next possible positions. As a result, the robot acts more

carefully when the human worker is in range of the robot. Therefore, accidents can be avoided ahead of time. In a domestic environment, e.g., a household, a service robot moving around the house and accomplishing domestic work has to be careful about humans intersecting the robot's path. It needs to plan under uncertainty and predict the movement of the human or even replan its path depending on the situation.

Also, instead of using the DESPOT solving algorithm for the POMDP, a different algorithm could be used to benchmark the performance. As shown in section 2.6 on page 11, a variety of point-based solving methods can be used.

Bibliography

Bai, H., Cai, S., Ye, N., Hsu, D., and Lee, W. S. (2015). Intention-aware online POMDP planning for autonomous driving in a crowd. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 454–460.

Bai, H., Hsu, D., Kochenderfer, M. J., and Lee, W. S. (2012). Unmanned aircraft collision avoidance using continuous-state POMDPs. *Robotics: Science and Systems VII*, 1:1–8.

Bai, H., Hsu, D., Lee, W. S., and Ngo, V. A. (2010). Monte Carlo value iteration for continuous-state POMDPs. In *Algorithmic foundations of robotics IX*, pages 175–191. Springer.

Bandyopadhyay, T., Jie, C. Z., Hsu, D., Ang Jr, M. H., Rus, D., and Frazzoli, E. (2013). Intention-aware pedestrian avoidance. In *Experimental Robotics*, pages 963–977. Springer.

Bellman, R. (1957a). A Markovian Decision Process. *Indiana Univ. Math. J.*, 6:679–684.

Bellman, R. (1957b). *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition.

Bertsekas, D. P., Bertsekas, D. P., Bertsekas, D. P., and Bertsekas, D. P. (1995). *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA.

Brechtel, S., Gindele, T., and Dillmann, R. (2013). Solving continuous POMDPs: Value iteration with incremental learning of an efficient space representation. In *International Conference on Machine Learning*, pages 370–378.

Brechtel, S., Gindele, T., and Dillmann, R. (2014). Probabilistic decision-making under uncertainty for autonomous driving using continuous POMDPs. In *Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on*, pages 392–399. IEEE.

Dietterich, T. G. (1998). The MAXQ Method for Hierarchical Reinforcement Learning. In *ICML*, volume 98, pages 118–126. Citeseer.

Doucet, A., Godsill, S., and Andrieu, C. (2000). On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and computing*, 10(3):197–208.

Fard, P. R. and Yahya, K. (2012). A Mixed Observability Markov Decision Process Model for Musical Pitch. *arXiv preprint arXiv:1206.0855*.

Howard, R. (1960). *Dynamic Programming and Markov Processes*. Technology Press of the Massachusetts Institute of Technology.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99 – 134.

Kurniawati, H., Du, Y., Hsu, D., and Lee, W. S. (2011). Motion planning under uncertainty for robotic tasks with long time horizons. *The International Journal of Robotics Research*, 30(3):308–323.

Kurniawati, H., Hsu, D., and Lee, W. S. (2008). SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces. In *Robotics: Science and systems*, volume 2008. Zurich, Switzerland.

Markov, A. (1954). *Theory of Algorithms*. TT 60-51085. Academy of Sciences of the USSR.

Meywerk, M. (2015). *Vehicle Dynamics*. John Wiley & Sons.

Ong, S. C., 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.

Pajarinen, J. and Peltonen, J. (2011). Periodic finite state controllers for efficient POMDP and DECPOMDP planning. In *Advances in Neural Information Processing Systems (NIPS)*, page 26362644.

Pineau, J., Gordon, G., Thrun, S., et al. (2003a). Point-based value iteration: An anytime algorithm for POMDPs. In *IJCAI*, volume 3, pages 1025–1032.

Pineau, J., Montemerlo, M., Pollack, M., Roy, N., and Thrun, S. (2003b). Towards robotic assistants in nursing homes: Challenges and results. *Robotics and autonomous systems*, 42(3-4):271–281.

Pineau, J. and Thrun, S. (2001). Hierarchical POMDP Decomposition for A Conversational Robot.

Raiffa, H. (1968). *Decision Analysis: Introductory Lectures on Choices und Uncertainty*. Addison-Wesley.

Sadigh, D., Sastry, S. S., Seshia, S. A., and Dragan, A. (2016). Information gathering actions over human internal state. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 66–73. IEEE.

Scovanner, P. and Tappen, M. F. (2009). Learning pedestrian dynamics from the real world. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 381–388. IEEE.

Shani, G., Pineau, J., and Kaplow, R. (2013). A survey of pointbased POMDP solvers. *Autonomous Agents and MultiAgent Systems*, 27(1):151.

Silver, D. and Veness, J. (2010). Monte-Carlo planning in large POMDPs. In *Advances in neural information processing systems*, pages 2164–2172.

Smallwood, R. D. and Sondik, E. J. (1973). The optimal control of partially observable Markov processes over a finite horizon. *Operations research*, 21(5):1071–1088.

Smith, B. W. (2013). SAE levels of driving automation. *Center for Internet and Society. Stanford Law School. http://cyberlaw. stanford. edu/blog/2013/12/sae-levels-drivingautomation*.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Temizer, S., Kochenderfer, M. J., Kaelbling, L. P., Lozano-Pérez, T., and Kuchar, J. K. (2010). Collision avoidance for unmanned aircraft using Markov decision processes. In *AIAA Guidance, Navigation, and Control Conference, Toronto, Canada*.

Wolf, T. B. (2009). *Aircraft Collision Avoidance Using Monte Carlo Real-Time Belief Space Search*. PhD thesis, Massachusetts Institute of Technology.

Ye, N., Somani, A., Hsu, D., and Lee, W. S. (2017). DESPOT: Online POMDP planning with regularization. *Journal of Artificial Intelligence Research*, 58:231–266.