Learning Complex Markov Models for Mobile Robotics

Peter Ondrúška
St. Hilda's College



Supervisor:
Professor Ingmar Posner

Mobile Robotics Group
Department of Engineering Science
University of Oxford

July 2016



Peter Ondrúška
St. Hilda's College

Doctor of Philosophy
July 2016

Learning Complex Markov Models for Mobile Robotics

Abstract

Robotics is going to radically transform the shape of the society in the coming decades. Applications such as self-driving cars and the mass adoption of automatic robotic manufacturing have the potential to cause the biggest disruption since the industrial revolution. The improvements in computational power and sensors allow robots to process more and more data, but robots still lack effective software technology to operate in complex real-world environments. This absence leaves even today's most-advanced robots fragile and with the level of intelligence closer to an ant or a mouse rather than a human. The family of *Markov models* provides a sound theoretical footing for robots to handle a wide variety of tasks, but their representation power and inference speed suffers when scaled up to the complexity of real-world domains.

In this thesis we argue that these issues can be ameliorated by learning many of the constituent components of a Markov models directly from the data. We demonstrate that such an approach allows us to successfully tackle a variety of challenging real-world problems such as: object tracking using only raw, imperfect sensor data; learning driver rewards; predicting effective electric vehicle range; planning in partially-observable environments or transferring learned knowledge. The work in this thesis contributed to a total of six original research publications, presented at several reputable robotics and AI conferences.

Statement of Authorship

This thesis is submitted to the Department of Engineering Science, University of Oxford, in fulfilment of the requirements for the degree of Doctor of Philosophy. This thesis is entirely my own work, and except where otherwise stated, describes my own research.

Peter Ondrúška, St. Hilda's College

Funding

The work described in this thesis was funded by the Engineering and Physical Sciences Research Council (EPSRC).

Acknowledgements

First and foremost, thanks are due to my supervisor Ingmar Posner. His excellent guidance, support and freedom to explore interesting things gave rise to the work in this thesis.

Thanks also to Paul for creating Oxford Mobile Robotics Group, the place of excellent research and people. Thanks also to all my co-authors Julie, Corina, Chi, Markus and Dominic who helped me wrestling the robots.

Finally, but most importantly thanks to all my teachers, mentors family and friends for their support and inspiration over the years.

Notation

Symbol	Description
$\mathcal{S}$	Set of possible states
$\mathcal{A}$	Set of possible actions
$\mathcal{T}$	State-transition model
$\mathcal{Z}$	Set of sensor observations
s	MDP state, $s \in \mathcal{S}$
a	MDP action, $a \in \mathcal{A}$
$\mathcal{R}$	MDP reward
π	MDP policy
h	HMM state
z	HMM observation, $z \in \mathcal{Z}$
ς	Statespace trajectory $\{s_1, s_2, ..., s_n\}$
$\Lambda_{s \to g}$	Set of all trajectories between MDP states s and g
$\mathcal{B}$	Belief state
w	Weight vector of length N, $w \in \mathbb{R}^N$
f	Features vector of length N, $f \in \mathbb{R}^N$
$x_{1:N}$	Continuous sequence of variables $\{x_1, x_2, ..., x_N\}$

Contents

1 Introduction **9**

1.1 Motivation 9

1.2 Contributions and Thesis Outline 13

2 Preliminaries **16**

2.1 Markov Models 16

2.1.1 The Markov Chain 18

2.1.2 The Hidden Markov Model 18

2.1.3 The Markov Decision Process 20

2.1.4 The Partially Observable Markov Decision Process 21

2.1.5 MDP policy 23

2.2 Function Approximation 26

2.2.1 Data Fit 28

2.2.2 Common Families of Models 31

2.2.3 Training 33

3 Learning Representations for State Estimation and Tracking **34**

3.1 Related Works 37

3.2 Problem Formulation 40

3.3 The Model 41

3.3.1 Predicting Occupancy 42

3.3.2 Semantic Segmentation 44
3.4 Learning to Track with a Recurrent Neural Network 44
3.4.1 Supervised Training 46
3.4.2 Unsupervised Training 47
3.5 Learning to Understand with Knowledge Transfer 49
3.5.1 Training 49
3.6 Neural Network Architectures 51
3.6.1 Simple Architecture 51
3.6.2 Complex Architecture 52
3.6.3 Multi-task Architecture 57
3.7 Experiments 58
3.7.1 Synthetised Experiments 59
3.7.2 Real-world Experiments 62
3.8 Conclusions 70

4 Learning and Evaluating Policy for Personalised Prediction 71
4.1 Related Works 74
4.2 Electric Vehicle Range Prediction 76
4.2.1 Energy Prediction as MDP 77
4.2.2 Attainability Map 79
4.3 Learning Driver's Policy 81
4.4 Evaluating Driver's Policy 85
4.4.1 Computing the Mean 86
4.4.2 Computing Variance 87
4.4.3 Predicting All Destinations 88
4.5 Reward Models 91
4.5.1 Fixed Reward 91

4.5.2 Linear Model . . . 91
4.6 Energy Cost Models . . . 92
4.6.1 Hand-Engineered Model . . . 92
4.6.2 Linear Model . . . 94
4.7 Experiments . . . 96
4.7.1 Routing Prediction Accuracy . . . 96
4.7.2 Energy Model Accuracy . . . 99
4.8 Implementation . . . 103
4.9 Conclusions . . . 104

5 Planning in Partially Observable Environments for Sensing Optimisation 105
5.1 Perception Scheduling Problem . . . 107
5.2 The Model . . . 109
5.2.1 Perception Plan Energy . . . 110
5.2.2 Perception Plan Feasibility . . . 110
5.3 Solution Method . . . 113
5.3.1 Greedy Algorithm . . . 113
5.3.2 Belief MDP Algorithm . . . 117
5.4 Implementation . . . 119
5.4.1 Odometry Model . . . 120
5.5 Experiments . . . 123
5.5.1 Trials . . . 123
5.5.2 Evaluation . . . 125
5.6 Related Works . . . 128
5.7 Conclusions . . . 129

6 Conclusions 131
6.1 Summary of Contributions . . . 131

6.2 Future Work 132
6.2.1 Tracking in Partially Observable Environments 132
6.2.2 Learning and Evaluating the Policy 133
6.2.3 Planning in Partially Observable Environments 134
6.2.4 A Full Neural Pipeline 134
6.3 Concluding Remark 137

Chapter 1

Introduction

1.1 Motivation

The ability to know and understand *what is around me* and *what to do*, and to have this information available in *real time* is paramount for the effective operation of intelligent systems in real world. For example, an autonomous car must know about other cars, pedestrians and traffic signs on the road. At the same time, the robot must know how these objects behave in order to anticipate the future and plan accordingly in order to identify and achieve its own goals. Finally, the robot must *know what it does not know* to avoid risky behaviour.

To effectively deal with all these diverse requirements, the robotics and artificial intelligence communities have developed a family of *Markov Models* (MM) consisting of four main models *Markov Process* (MP), *Hidden Markov Model* (HMM), *Markov Decision Processes* (MDP) and *Partially-Observable Markov Decision Processes* (POMDP) capturing different problem scenarios. Particularly, in the most general POMDP model, the entire system can be described in terms of world state representation $\mathcal{S}$, capturing robot situation, possible actions $\mathcal{A}$ that the robot can execute, state transition $\mathcal{T}$ function, reward function $\mathcal{R}$ modelling robot goals, possible sensor observations $\mathcal{Z}$ and its observa-

tion model O modelling the limited and imperfect information robots receive about their environment. The main advantage of this abstraction is its simplicity and generality. It provides a principal solution for optimal robot behaviour (π) and it has been successfully applied to wide range of problems [44, 68, 77, 101].

However, the application of the Markov model formulation to complex robotics tasks suffers from a range of issues. Many of these arise from the inability to specify expressive-enough representations to capture necessary the complexity of the task or the lack of efficient inference methods. In a classical scenario, the state representation $\mathcal{S}$ should capture the complete state of a system. For example, for an autonomous car it should capture the exact position of the car on a road together with all other cars both visible and occluded, their velocities, accelerations etc. In a robotic manipulation task, it should capture the position and types of all relevant objects as well as the position of the robot itself. At the same time, the state transition models $\mathcal{T}$ and observation models O should be expressive enough to exactly capture the world dynamics, and be able to predict the outcome of different actions. Finally, the robot's objective $\mathcal{R}$ must be exactly and correctly given for the robot to both achieve the desired goals and avoid undesired behaviour. This system specification might be very hard to define, simply due to the fact that these things are too complex to compute or even unknown. At the same time, many examples of robotic decision making require forward-planning to compute a policy π. In complex partially-observable environments where only part of the robot environment is directly observed by robot sensors this requires a form of belief planning which takes into account the distribution of uncertainty about the actual unobserved world state. This, however, often becomes intractable due to the complexity arising from handling distributions over possible states.

We argue that the Markov model can still be a very efficient tool for dealing with robotics problems, if only more powerful representations of the defining quantities learned from data and inference algorithms are used. In this thesis we consider and advance the state-of-the-art in three different problem domains, where the applicability of Markov

models for modelling and addressing robotics problems is limited:

1. Learning the observation and transition models from scratch in partially-observable environments.

The ability to learn expressive internal state representations $\mathcal{S}$, transition $\mathcal{T}$, and observation models O, in the absence of ground-truth, using only raw, imperfect and occluded sensor observations is very challenging and has not previously been demonstrated. Considering the complex problem of tracking dynamic objects or environment state in complex partially observable environments, it would be very hard to write down exact state transition and observation models capturing object movement, interactions and appearance. These might be too complex or even unknown. Learning such operations directly from data is therefore desirable. The recent success in machine learning techniques, especially *deep learning*, has advanced the development of new and more accurate perception systems capable of extracting higher-level information, such as object class from raw sensor measurements [52, 91, 7]. These systems develop their capability from training high capacity models on large data sets, internally capturing the high complexity of the real world. However, such perception systems form only a small part of the robotics pipeline. Ability of a robot deployed in a real, partially-observable world with to learn the latent dynamics just by observing them is still a largely unsolved problem. We will engage with this problem by showing how a large and expressive hidden Markov model can be learned directly from partial observations, a possibility which has not demonstrated before.

2. The evaluation of inferred behaviours in large state spaces.

A robot needs to know what it wants to achieve. However, specifying the exact objective function in the form of reward $\mathcal{R}$ is, for many problems, very difficult. A robotic car, for example, does not only want to reach its destination, but also to drive safely and efficiently. A robotic manipulator does not only need to move objects, but also not break them by

doing so. The exact specification of these various properties of successful outcomes is difficult if not impossible. An alternative approach to the explicit specification of an objective function is to learn from demonstrations. The robot first observes a human demonstrator performing a given task, and from their behaviour it infers what it must achieve. This has been successfully applied to many scenarios, e.g. [3, 70, 109, 50]. These learned policies are effective for learning many short-term tasks, but due to the human's suboptimality or insufficient specification, might lead to undesirable long-term results. A robotic car can learn to drive inefficiently, or a robot tasked with cleaning as many dirty areas as possible might achieve its goal by creating more dirty surfaces to clean. The ability to analyse the expected long-term impact of such learned policies is therefore important. However, the size of the state space in these complex scenarios can be very large, and so the evaluation of many scenarios becomes intractable. We contribute to this area by presenting a novel method for learning and fast evaluation of a learned behaviour in large state spaces consisting of millions of states.

3. Planning to minimise perception.

A robot might be equipped with sensors allowing it to sense the environment, to determine world state and its own position in it. However, the amount of information it receives varies depending on its actions. This opens up the possibility of learning a policy actions π in order to optimise the amount of information received. This approach was explored in basic situations such as planning the most information-rich robot trajectories [96, 97, 25, 76]. We take the opposite approach – deliberately minimising the required information, in the form of turning off sensors when they are unnecessary. To the best of our knowledge, this is a novel approach which can be applied to almost any robotic task. The main challenge is that any of these solutions usually requires planning in belief space, and computing the exact solution is usually intractable. We show, that in our case, this can be done efficiently, leading to an effective and practical solution.

1.2 Contributions and Thesis Outline

In this thesis we advance the state-of-the-art in solving robotic problems using Markov models by designing novel solutions specifically designed to overcome the most challenging properties of the above-mentioned areas. We consider real-world problems with a challenging aspect, and design methods that delivering new, unmatched performance, and thus push the envelope of progress in the relevant area. In our choice of application domains we focus on the core requirements of robotics – such as our robots operating in environments involving some form of *motion*, and the solutions need to work *fast and robustly* while being relatively *easy to train and deploy*. At the same time, we are inspired by applications involving *energy-efficiency* in robotics. This area is relatively unexplored, and we consider it very important for the future.

In Chapter 2 we start with the review of the necessary theory of Markov models and their variations, their connections to the robotic pipeline, and the theory of machine learning methods which we use throughout this thesis. We then present our contributions by designing novel solutions for the following problems:

1. Learning to track and understand with recurrent neural networks.

A robot's ability to identify and track dynamic objects in partially-observable environments is a core dependency for many ground mobile robots, such as self-driving cars. Any effective solution heavily depends upon the quality of the state representation, transition, and observation models. In Chapter 3 we describe a new method based on using recurrent neural networks for training an internal model of the world in a partially observable environments, using only raw sensor observations $\mathcal{Z}$. We demonstrate that this method leads to a completely new paradigm for object tracking and data-efficient learning from supervised and unsupervised data. This work has contributed to the following conference publications:

- P. Ondruska and I. Posner, *"Deep Tracking: Seeing Beyond Seeing Using Recurrent Neural Networks,"* in The Thirtieth AAAI Conference on Artificial Intelligence (AAAI), Phoenix, AZ, USA, 2016.

- P. Ondruska, J. Dequaire, D. Wang and I. Posner, *"End-to-End Tracking and Semantic Segmentation Using Recurrent Neural Networks,"* in Workshop of Robotics Science and System (RSS), Detroit, MI, USA, 2016.
  [The best workshop paper award].

2. Predicting the driver-specific range of an electric vehicle.

The ability to learn user driving behaviour and to use it to predict the driver-specific range is a new avenue for mitigating *range anxiety*, a driver's fear of their electric vehicle running out of battery power. This task can be formulated in terms of inferring an unknown reward function of a Markov decision process while the resulting map can be computed by evaluating the resulting policy on a Markov decision process involving millions of states. In Chapter 4, we provide a new probabilistic inference method which is both fast and accurate. This work has contributed to the following conference publications:

- P. Ondruska and I. Posner, *"The Route Not Taken: Driver-Centric Estimation of Electric Vehicle Range,"* in Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS), Portsmouth, NH, USA, 2014.

- P. Ondruska and I. Posner, *"Probabilistic Attainability Maps: Efficiently Predicting Driver-Specific Electric Vehicle Range,"* in IEEE Intelligent Vehicles Symposium (IV), Dearborn, MI, USA, 2014.

and patent application

- UK Patent Application No. 1409620.0 Electric Vehicle Range

3. Minimising energy through perception optimisation.

The information coming from robotic sensors is highly redundant. The ability to perform a particular task with less information than is available opens up new avenues for saving energy. Simply turning off sensors at well-calculated moments can result in significant energy savings in energy-critical applications such as planetary rover missions. However, computing such a perception plan is difficult, as it involves planning in belief space, which is often intractable. By carefully formulating the problem in Chapter 5, we design a fast belief-planning algorithm to compute the optimal sensor schedule for a robot traversing any trajectory while saving the maximum amount of energy. This work has contributed to the following conference publication:

- P. Ondruska, C. Gurau, L. Marchegiani, C. H. Tong, and I. Posner, *"Scheduled Perception for Energy-Efficient Path Following,"* in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 2015.

Finally, we conclude the thesis in Chapter 6 and provide directions for further research, notably the possibility to combine topics to derive a fully-trainable robotic pipeline. This is a promising direction of research which might lead to the construction of highly-capable robots for which all internal representation, sensing and decision-making procedures are fully learned from the environment, rather than specified by human engineer.

Chapter 2

Preliminaries

To be able to effectively model robot operation, the artificial intelligence and robotics communities have developed a range of mathematical tools abstracting the entire process. Here we provide a brief overview of the methods required for understanding the topics discussed in this thesis. A more detailed discussion can be found in an excellent work of [82, 93]. We describe the family of *Markov Models*, the fundamental tool of artificial intelligence for modelling dynamic systems and basics of *machine learning* as a powerful tool to implement them. We will use this formalism to frame and solve all the problems in the thesis, providing clarity, consistency and expressing the relationships between the different problems.

2.1 Markov Models

A Markov model is a stochastic model used to model the evolution of a non-deterministic system such as the environment around the robot and the state of the robot itself. In this framework, all the relevant information about the system at any given time t can be abstracted by the *world state* s_t, capturing information such as the position of the robot, other objects, etc. This process is assumed to satisfy the *Markov property*, which states

Markov Models		*Do we have control over the state transitions?*	
		No	*Yes*
Are the states completely observable?	*Yes*	**MC** *Markov Chain*	**MDP** *Markov Decision Process*
	No	**HMM** *Hidden Markov Model*	**POMDP** *Partially Observable Markov Decision Process*

Figure 2.1: The family of Markov models.

that knowledge of the state s_t captures the complete information capturing all the relevant information about the present and the past of the system necessary for predicting its future evolution, that is,

$$p(s_{t+1}|s_{1:t}) = p(s_{t+1}|s_t). \tag{2.1}$$

This property is appealing because it reduces the information necessary for the inference which would be otherwise intractable. The only information that the robot needs to remember is the last value of the state.

As illustrated in Figure 2.1 there are four common Markov models used in different situations, depending on whether the state is fully observable and whether the robot can affect this state by taking actions. These different models are used as the main tool to frame and solve the three major problems considered in each chapter:

- *Markov Chain* (MC)
- *Hidden Markov Models* (HMM) discussed in Chapter 3
- *Markov Decision Processes* (MDP) discussed in Chapter 4

- *Partially Observable Markov Decision Processes* (POMDP) discussed in Chapter 5

2.1.1 The Markov Chain

The simplest Markov model is a Markov chain formally defined in Definition 1. It models a dynamic fully-observable system like, for example, the environment around a stationary robot equipped by a perfect all-observing sensor. We do not use a Markov chain model in this thesis directly, but we will assume that the underlying model dynamics can be described in a form of Markov chain. The challenging task is choosing an appropriate state representation that captures the full state of such a system. In some cases, such as the ones discussed in Chapter 4 or 5, the choice of appropriate state representation is rather straightforward, but in many real-world scenarios with unclear dynamics such as the one in Chapter 3 this is a challenging task. There we show how an appropriate state-space representation can be automatically learned as a part of a HMM, which described in the next section.

Definition 1. A discrete-time Markov process (MP) is a tuple $(\mathcal{S}, \mathcal{T})$, where

- $\mathcal{S}$ is a set of states,
- $\mathcal{T}$ is the state transition probability $p(s_{t+1}|s_t)$ that state s_t will lead to state s_{t+1} where t is the current time-step.

2.1.2 The Hidden Markov Model

The full state s_t of the system is seldom directly observable when modelling complex processes in real world. This is usually due to natural occlusions or imperfect sensor observations. Such situations can be modelled by a Hidden Markov Model (HMM), as shown in Figure 2.2. In this model the full system state is considered *hidden* and instead the robot has access to only sensor observations z_t. The main considered problem is then

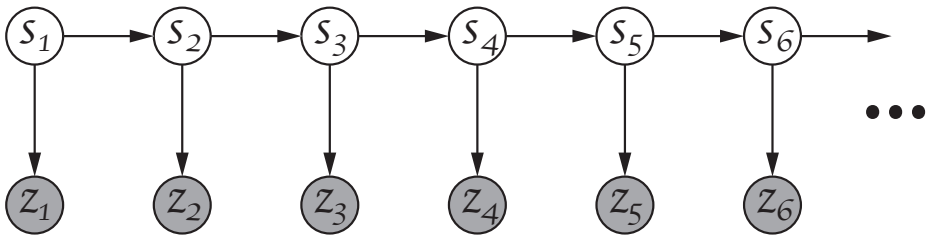


Figure 2.2: The graphical model for a hidden Markov model.

to estimate $p(s_t|z_{1:t})$ the value of the hidden state given the sequence of observations up to the given point. In robotics, the task of inferring the true state is the job of the robot's perception system. Examples of such perception systems in an autonomous car are pedestrians detectors, obstacle detectors, localisation subsystem etc. In chapter 3 we discuss the task of learning a hidden Markov model for complex tasks like tracking dynamic objects in partially-observable environments.

Definition 2. A hidden Markov Model (HMM) is a 5-tuple $(\mathcal{S}, \mathcal{Z}, P_0, \mathcal{T}, O)$, where

- $\mathcal{S}$ is a set of states,
- $\mathcal{Z}$ is a set of observations,
- P_0 is an initial state distribution $p(s_0)$,
- $\mathcal{T}$ is a state transition probability $p(s_{t+1}|s_t)$ that state s_t will lead to state s_{t+1},
- O is a conditional probability $p(z_t|s_t)$ of observing z at state s.

Even with many observations the exact value of the hidden state can not be usually determined with certainty it is often desirable to estimate and maintain its entire distribution. This distribution is called a *Belief state* $\mathcal{B}_t$

$$\mathcal{B}_t = p(s_t|z_{1:t}). \tag{2.2}$$

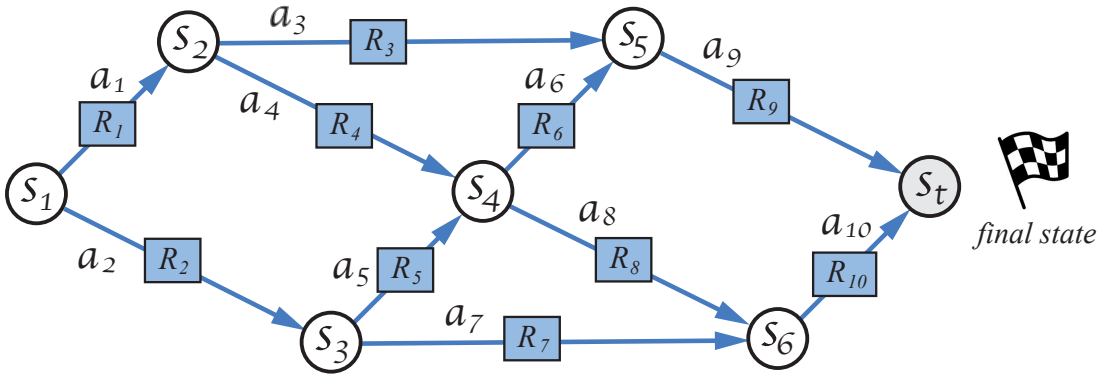


Figure 2.3: An example of a state-transition graph of a Markov decision process. At each moment a system (robot) can be in one of N possible states s_i and can take one of M possible actions a_j immediately receiving reward $\mathcal{R}_j$ (which can be both positive or negative). The goal is to maximize the total cumulative reward over a sequence of K actions.

Its important property is that the sequence of belief states forms a Markov chain, i.e. the knowledge of a belief state $\mathcal{B}_t$ captures the full information about the past observations necessary for future inference. Effective inference of the belief state will play a key part in designing effective and tractable solutions in Chapters 3 and 5.

2.1.3 The Markov Decision Process

In addition to sensing, a robot can decide to take an action $a \in \mathcal{A}$ to achieve its goal. In mobile robotics this usually corresponds to sending the appropriate signal to actuators or motors. Actions can take a higher-level form, such as turning at the intersection (as discussed in Chapter 4) or turning on and off the robot's sensors (as discussed in Chapter 5). All these situation can be described by a Markov decision process (MDP) illustrated in Figure 2.3.

In this model, the robot's intentions are expressed by an objective function $\mathcal{R}_{s,a}$ describing which combinations of states and actions are more desirable than others. It represents rewards for the robot achieving desired outcomes, such as an autonomous car

reaching its desired destination, and penalties for negative outcomes, such as crashing into obstacles or running out of energy.

Definition 3. A Markov decision process is a 4-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$, where

- $\mathcal{S}$ is a set of states,
- $\mathcal{A}$ is a finite set of actions,
- $\mathcal{T}$ is the conditional state probability $p(s_{t+1}|s_t, a)$ that action a in state s_t will lead to state s_{t+1},
- $\mathcal{R}_{s,a}$ is the immediate reward (or expected immediate reward) received after taking action a from state s.

Additionally, one can define a discount factor $\gamma \in [0, 1]$, which represents the relative importance of potential future rewards and the immediate reward. In this thesis we will deal only with undiscounted Markov decision processes where $\gamma = 1$ under assumption that the length of every possible trajectory is bounded by some length N_{max}.

2.1.4 The Partially Observable Markov Decision Process

Finally, and similarly to hidden Markov model, in robotics we often make decision under partial observability of the state. The Markov decision process assumes full observability of the state, which leads us to the Partially Observable Markov Decision Process (POMDP). At each time step t, the robot might receive some measurement z_t giving it *some information* about its current state s_t.

Definition 4. A partially observable Markov decision process is a 7-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{Z}, P_0, \mathcal{T}, \mathcal{R}, O)$, where

- $\mathcal{S}$ is a set of states,
- $\mathcal{A}$ is a finite set of actions,

- $\mathcal{Z}$ is a set of observations,
- P_0 is the initial state distribution $p(s_0)$,
- $\mathcal{T}$ is the probability $p(s'|s,a)$ that action a in state s will lead to subsequent state s',
- $\mathcal{R}_{s,a}$ is the immediate reward received after taking action a from state s,
- O is the observation probability $p(z|s)$ of observing z from state s.

Belief MDP

Alternatively, the POMDP can be expressed in the terms of the classical MDP, in which the POMDP state is a belief distribution over the states in the original MDP. This formulation becomes useful in Chapter 5 where we discuss an effective planning algorithm using the *belief MDP*, saving energy in exchange for increased uncertainty in the belief distribution.

Definition 5. A belief MDP is a 5-tuple $(B, \mathcal{A}, \mathcal{B}_0, \tau, \mathcal{R})$, where

- B is a set of possible belief states over over the MDP states,
- $\mathcal{A}$ is the same finite set of action as for the POMDP,
- $\mathcal{B}_0$ is the initial belief,
- τ is the belief state transition function, the probability $p(\mathcal{B}'|\mathcal{B},a)$ that action a in state $\mathcal{B}$ will lead to subsequent state $\mathcal{B}'$,
- $\mathcal{R}_{\mathcal{B},a}$ is the expected immediate reward received after taking action a from belief state $\mathcal{B}$.

The belief transition function is defined by

$$p(\mathcal{B}'|\mathcal{B},a) = \sum_{s'} p(s'|s,a)\mathcal{B}(s), \tag{2.3}$$

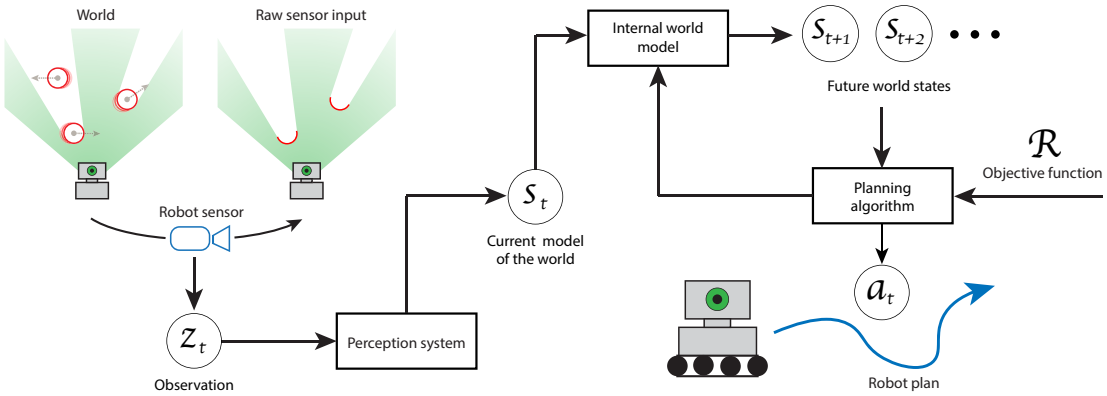


Figure 2.4: An example of a robot pipeline. Sensors provide observations z which are processed by a robot perception system to estimate the current system state s. The robot's decisions are made by its planning system to maximize reward $\mathcal{R}$, accounting for the near future $s_{t:t+n}$ as predicted by the internal world model.

and the belief-action-reward is computed as an average over possible state-action-rewards, weighted by the belief,

$$\mathcal{R}_{\mathcal{B},a} = \sum_{s} \mathcal{B}(s)\mathcal{R}_{s,a}. \tag{2.4}$$

2.1.5 MDP policy

Achieving complicated goals, such as long-range navigation through an environment, usually involves the successive execution of several actions. Computing a schedule of desired actions is handled by a planning system, Figure 2.4, which, given the current state s, produces a desired action to execute $p(a|s)$. Such mapping is called a Markov decision process *policy* π. If the policy assigns a non-zero probability to multiple actions at given state then the policy is called *stochastic*, otherwise it is *deterministic*.

Starting in an intial state s_0 and acting according to a policy π for k steps results into a trajectory $\varsigma = \{s_0, a_0, s_1, a_1, \ldots, s_k\}$ where

$$a_k \sim p_\pi(a|s_t) \tag{2.5}$$

$$s_{t+1} \sim p(s|s_t, a_t). \tag{2.6}$$

The probability of any such trajectory is

$$p_\pi(\varsigma) = \prod_{\{s_t, a_t, s_{t+1}\} \in \varsigma} p(s_{t+1}|s_t, a_t) p_\pi(a_t|s_t) \tag{2.7}$$

while its cumulative reward is

$$\mathcal{R}_\varsigma = \sum_{\{s,a\} \in \varsigma} \mathcal{R}_{s,a}. \tag{2.8}$$

The Policy Value Function

Some policies are better and lead to higher expected reward than others. The reward of a particular policy is captured in a *policy value function*. This captures the average reward when starting in some state s and following given policy,

$$\mathcal{V}_\pi(s) = \sum_\varsigma p_\pi(\varsigma) \mathcal{R}_\varsigma. \tag{2.9}$$

However, this formula is rarely used in practice, because there are usually exponential or even infinitely many possible trajectories ς to be summed over. Instead, the policy value can be computed by exploiting the linear relationships of policy values between states. This is known as *Bellman's equation* [5]:

$$\mathcal{V}_\pi(s) = \sum_a p_\pi(a|s) \left(\mathcal{R}_{s,a} + \sum_{s'} p(s'|s,a) \mathcal{V}_\pi(s') \right). \tag{2.10}$$

Computing Optimal Policy

The optimal solution to many tasks, such as in Chapter 4 and 5 can be formulated as an optimal policy π^*, i.e. one maximizing (2.9). We summarise two basic algorithms for computing the optimal policy, known as *value* and *policy* iteration [6, 90]. Value iteration (summarised in Algorithm 1) is an iterative process of updating the value func-

Algorithm 1 Value iteration

Input: $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$ *Markov decision process*
Output: $\mathcal{V}_k$ *value function*
π *optimal policy*

1. $\mathcal{V}_0 \leftarrow 0$

Iteratively compute optimal value

2. **for** $k = 1, 2, ..., K$ **do**
3. $\quad \mathcal{V}_k(s) = \max_a\{\mathcal{R}_{s,a} + \sum_{s'} p(s'|s,a)\mathcal{V}_{k-1}(s')\}$

Compute optimal policy

4. $\pi(s) = \arg\max_a\{\mathcal{R}_{s,a} + \sum_{s'} p(s'|s,a)\mathcal{V}_K(s')\}$

Algorithm 2 Policy iteration

Input: $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$ *Markov decision process*
Output: $\mathcal{V}_k$ *value function*
π_k *optimal policy*

1. $\pi_0 \leftarrow$ *random initialisation*
2. $k = 0$
3. **do**
4. $\quad k = k + 1$

Solve system of linear equations for $\mathcal{V}_k$

5. $\quad \mathcal{V}_k(s) = p_{\pi_{k-1}}(a|s)\,(\mathcal{R}_{s,a} + \sum_{s'} p(s'|s, \pi_{k-1}(s))\mathcal{V}_k(s'))$

Update optimal policy

6. $\quad \pi_k(s) = \arg\max_a\{\mathcal{R}_{s,a} + \sum_{s'} p(s'|s,a)\mathcal{V}_k(s')\}$
7. **while** $\pi_k \neq \pi_{k-1}$

tions with a one-step look-ahead for K iterations until convergence (i.e. until $\forall s$: $\|\mathcal{V}_K(s) - \mathcal{V}_{K-1}(s)\| < \epsilon$) resulting into approximation of the optimal policy π^*. Alternatively, the policy iteration method (summarised in Algorithm 2) starts with a random policy π_0 and in each iteration computes the value for this policy. Then, the policy is updated based on this value.

2.1.5.1 Policy Learning From Expert Demonstrations

Finally, the topic explored in Chapter 4 is learning the policy π from only a set of state-transition observations. This is useful when the task is to uncover or mimic expert behaviour in the case of unknown rewards $\mathcal{R}$. Given a set of expert demonstrations $D = \{\varsigma_1, \varsigma_2, ..., \varsigma_N\}$, the task of policy learning is to find the expert's policy π explaining

these demonstrations:

$$\pi^* = \arg\max_{\pi} p_\pi(D). \tag{2.11}$$

One way to learn the required policy is through the approach of *policy matching* [21], where the policy is sought to directly mimic expert behaviour. However, this can lead to generalisation problems when too few demonstrations are provided. The approach we take in Chapter 4 is to assume that the agent is rational and aims to accumulate hidden reward $\mathcal{R}$ received through its policy decisions π. The task is then to uncover the reward structure that renders the observed behaviour optimal:

$$\mathcal{R}^* = \arg\max_{\mathcal{R}} p_{\pi^*(\mathcal{R})}(D)p(\mathcal{R}), \tag{2.12}$$

where $\pi^*(\mathcal{R})$ is an optimal policy maximizing reward $\mathcal{R}$ and $p(\mathcal{R})$ is prior over possible rewards. This problem is known as an *inverse reinforcement learning*, and was successfully used to predict driver's route [109], learn user behaviour [55] or learn to park an autonomous car from demonstrations [2].

2.2 Function Approximation

To implement parts of a Markov model in a real-world application, function approximation methods are often used to approximate the true underlying quantities of the model. In Chapter 3 we apply this to the state transition function, and in Chapter 4 we apply it to the reward function. In this section we review basic methods used to create an approximate function from a set of observations. A typical work-flow for doing this in machine learning is shown in Figure 2.5. The corresponding parts are discussed below.

A classical function approximation is a mapping $F : X \to Y$, from a set of evidence data $x \in X$ (such as sensor observations z) to corresponding predictions $y \in Y$ (such as

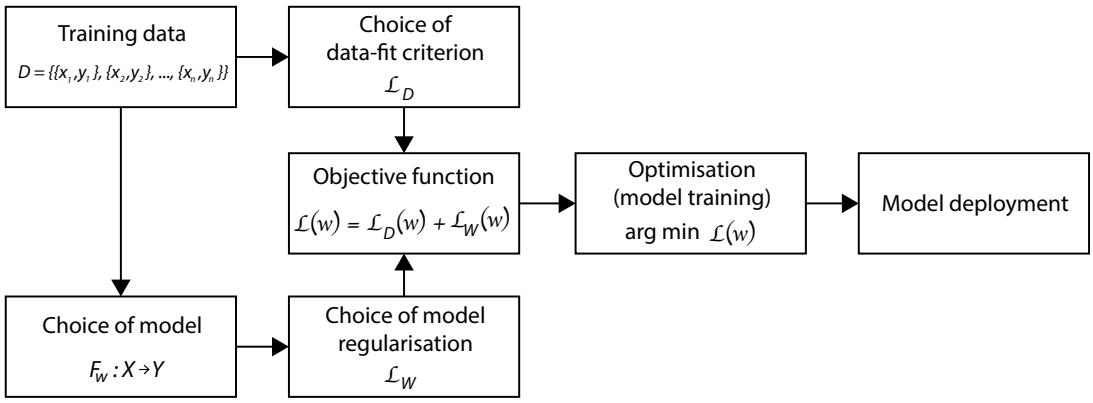


Figure 2.5: A typical machine learning work-flow. See text for more details.

the underlying hidden state s) governed by parametrisation weights w,

$$y = F_w(x). \tag{2.13}$$

Since the mapping varies with the weights, we measure the quality of mapping via a loss function $\mathcal{L}(w)$, and the goal is to find weights minimising the loss

$$w^* = \arg\min_w \mathcal{L}(w). \tag{2.14}$$

This loss function expresses the quality of approximating a set of training data $D = \{\{x_1, y_1\}, \{x_2, y_2\}, ..., \{x_N, y_N\}\}$ we have collected from the relevant domain. In addition, it takes into account the probability of the model itself, to avoid what known as *over-fitting*, where the model is trained to fit training data perfectly but generalises poorly to new, unobserved data as a result of fitting to the noise present in the training data. These are measured respectively by a data fit term $\mathcal{L}_D(w)$ and model regularisation $\mathcal{L}_W(w)$,

$$\mathcal{L}(w) = \mathcal{L}_D(w) + \mathcal{L}_W(w). \tag{2.15}$$

To find parameters w that minimise the loss function, we are interested in computing the

gradient of the loss function, given by

$$\frac{\partial \mathcal{L}}{\partial w}(w) \quad = \quad \frac{\partial \mathcal{L}_D}{\partial w}(w) + \frac{\partial \mathcal{L}_W}{\partial w}(w). \tag{2.16}$$

We use a simple gradient-descent optimisation procedure discussed in Section 2.2.3 to find the weights and hence the approximating function.

2.2.1 Data Fit

To measure how well a function $y = F_w(x)$ approximates the observed noisy data $D = \{\{x_1, y_1\}, \{x_2, y_2\}, ..., \{x_N, y_N\}\}$ we need a mechanism to express this property in the form of data-loss function $\mathcal{L}_D(w)$. To do this we restrict ourselves to a probabilistic interpretation and consider that the predicted value by function $F_w(x)$ does not produce a single value but rather defines a predictive probability distribution P, and the observed data D are sampled from this distribution:

$$P_i \quad \leftarrow \quad F_w(x_i), \tag{2.17}$$

$$y_i \quad \sim \quad P_i. \tag{2.18}$$

The principle is that the quality of the approximation $F_w(x)$ increases with the likelihood of generating the real-observed data. This gives rise to the required loss function

$$\mathcal{L}_D(w) = -\log p_w(y|x). \tag{2.19}$$

The predictive distribution P varies from case-to-case and depends on whether the variable to predict is *continuous* $y \in \mathbb{R}$, resulting into regression problem, or *categorical* where it can have values of one of K classes $y \in \{1, 2, ..., K\}$. In what follows we cover these main two cases.

2.2.1.1 Regression

In the case of predicting a continuous variable, this is done through predicting its mean

$$\mu = F_w(x). \tag{2.20}$$

The most commonly used assessment of the quality of this fit is the L2-loss, where we assume that the observed values are the predicted ones affected by Gaussian noise, resulting in the predictive distribution

$$P_w(y|x) = \prod_{\{x,y\}} \mathcal{N}(y|\mu, 1). \tag{2.21}$$

The loss function is then given by computing the negative logarithm of this distribution,

$$\mathcal{L}_D(w) = -\frac{1}{2} \sum_{\{x,y\}} (y - F_w(x))^2 \tag{2.22}$$

with gradient

$$\frac{\partial \mathcal{L}_D}{\partial w}(w) = \sum_{\{x,y\}} (y - F_w(x)) \cdot \frac{\partial F_w}{\partial w}(x). \tag{2.23}$$

2.2.1.2 Classification

In the case of predicting a categorical variable (which has one of K classes), we predicting the probability $p \in [0,1]^K$ of belonging to each class. This probability is usually computed by normalising a real-valued output $r \in \mathbb{R}^{\mathbb{K}}$ into the $[0,1]$ interval using soft-max operation

$$r = F_w(x), \tag{2.24}$$

$$p_i = \frac{\exp\{r_i\}}{\sum_{i=1}^{K} \exp\{r_i\}}. \tag{2.25}$$

where r_i resp. p_i is the i-th component of a K-dimensional vector. The predictive distribution of the $i-th$ class has a form

$$P_w(y'|x) = \prod_{\{x,y\}} \prod_i {p_i}^{y'_i}, \tag{2.26}$$

where y' is a vector of length K in a form 1-of-K encoding of $y \in \{1, 2, ..., K\}$ with exactly one 1 corresponding to the observed class. The loss function is

$$\mathcal{L}_D(w) = -\sum_{\{x,y\}} \sum_i y'_i \log p_i, \tag{2.27}$$

and gradient given by

$$\frac{\partial \mathcal{L}_D}{\partial w}(w) = -\sum_{\{x,y\}} \sum_i \frac{y'_i}{p_i} \cdot \frac{\partial p_i}{\partial w}(w) \tag{2.28}$$

where $\frac{\partial p_i}{\partial w}(w)$ is derived from (2.25) as

$$\frac{\partial p_i}{\partial w}(w) = -\sum_{\{x,y\}} \sum_i p_i \cdot (1 - p_i) \cdot \frac{\partial r_i}{\partial w}(w). \tag{2.29}$$

The Binary Case

If $K = 2$ we have a special case, known as *binary classification*. If the probability of the first class is p and the second class $1 - p$, the loss function is simplified to

$$\mathcal{L}_D(w) = -\sum_{\{x,y\}} y \log p + (1 - y) \log(1 - p), \tag{2.30}$$

with gradient

$$\frac{\partial \mathcal{L}_D}{\partial w}(w) = -\sum_{\{x,y\}} \frac{y - p}{p \cdot (1 - p)} \cdot \frac{\partial p}{\partial w}(w). \tag{2.31}$$

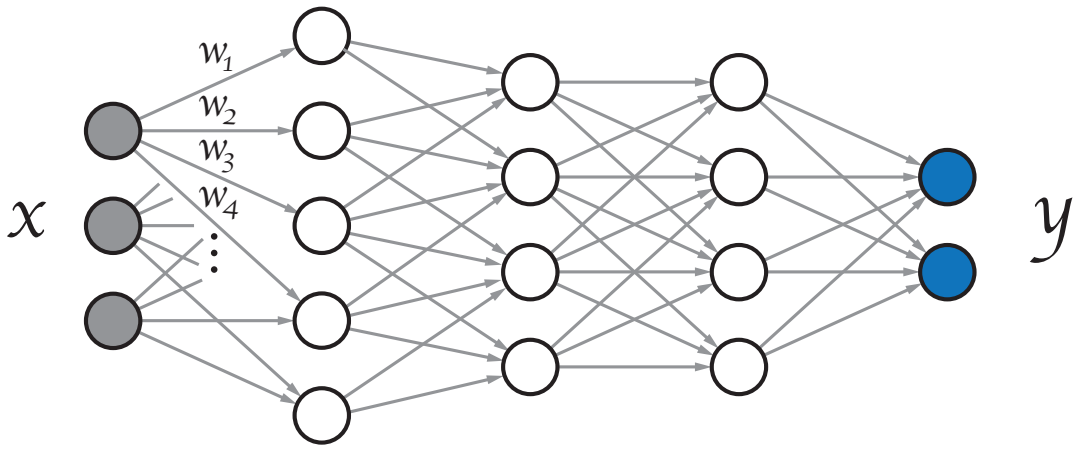


Figure 2.6: An example of a feed-forward multi-layer neural network. An input x of size 4 is mapped through a series of hidden layers to an output y of size 2.

2.2.2 Common Families of Models

Here we summarise two parametric models, one linear and one neural network, both of which modelling $F_w : \mathbb{R}^N \to \mathbb{R}^M$.

Linear model

The simple linear model is given by

$$F_w(x) = w^\top x, \tag{2.32}$$

where w is a $\mathbb{R}^{N \times M}$ matrix. The gradient with respect to weights is

$$\frac{\partial F_w}{\partial w}(x) = x \cdot (\mathbf{1}^M)^\top, \tag{2.33}$$

where $\mathbf{1}^M$ is a vector of ones of length M. We use this model extensively in Chapter 4 to model the reward $\mathcal{R}$ and energy consumption E as a function of road-segment features f.

Neural network

A more complex model we use in Chapter 3 is a multi-layer neural network, as illustrated in Figure 2.6. Here, the output is a composition of functions which correspond to activations of units in different layers:

$$F_w(x) = F_w^n(F_w^{n-1}(\cdots F_w^1(x)\cdots)). \tag{2.34}$$

The output of the i-th layer being y^i, the network gradient is given by

$$\frac{\partial F_w}{\partial w}(x) = \frac{\partial y^n}{\partial w}(x), \tag{2.35}$$

where

$$\frac{\partial y^i}{\partial w}(x) = \frac{\partial F_w^i}{\partial w}(y^{i-1}) \cdot \frac{\partial y^{i-1}}{\partial w}(x). \tag{2.36}$$

As there are many possible ways to implement individual layers resulting in different network architectures, such as convolutional, fully-connected, recurrent etc. we refer the reader to a comprehensive description in [7].

2.2.2.1 Model Regularisation

To measure the probability over parameters w of the chosen model in order to avoid overfitting, we usually make an assumption that the model parameter weights come from a multinomial, zero-mean distribution resulting in the model-loss function

$$\begin{aligned} \mathcal{L}_W(w) &= -\log p(w) && (2.37)\\ &= -\lambda \cdot \frac{1}{2} \sum_{w_i} w_i^2, && (2.38) \end{aligned}$$

where λ is the regularisation strength and the gradient is given by

$$\frac{\partial \mathcal{L}_W}{\partial w}(w) = -\lambda \cdot w. \tag{2.39}$$

Optimising for this regularisation term together with the previously described data-term in the form of Equation 2.15 results into model which both fits observed data and generalises better to new, unobserved data.

2.2.3 Training

Gradient descent is commonly used to find the optimal weights, minimising the loss function for a chosen model. We start with random weights w_0 and in each iteration we evaluate the model and compute gradient of the loss function with respect to weights. Next, we update the weights in the opposite direction to the gradient, using the update rule

$$w_t = w_{t-1} - \alpha \frac{\partial}{\partial w} \mathcal{L}(w_{t-1}). \tag{2.40}$$

Parameter α is the update step size, usually set to a small positive value. We repeat this procedure for a number of steps until model convergence, i.e. until either the value of loss function or the weights do not change from one iteration to the next.

There are many variations to this simple training procedure, resulting in faster training times. For example, in practice it is preferred to use stochastic gradient descent [12], where the gradient is computed based only on a randomized subset of the data, and the update procedure is computed using the more robust Adagrad method [27]. In Adagrad the learning rate of each weight parameter is estimated independently from others improving the overall learning speed.

Chapter 3

Learning Representations for State Estimation and Tracking

One of the difficulties in applying Markov Models is the choice of appropriate models for the internal representation $\mathcal{S}$, transition $\mathcal{T}$ and observation O that capture the high complexity of the world while maintaining the ability to perform inference over them. In this chapter we demonstrate a new method for learning these quantities efficiently, in the context of object tracking using a planar laser sensor. We use only raw sensor observations in a partially observable environment where the state of the full scene is due to the occlusions rarely ever fully seen. In the context of Hidden Markov Models from Chapter 2 this relates to learning the structure of states s internally capturing the full state s_t of the world and corresponding state update operation given new observation z_t $P(s_{t+1}|s_t, z_t)$. For clarity, and to express that this state is never directly observed, we will use the symbol h_t instead of s_t across this chapter.

The main idea is that, as opposed to specifying the exact form and structure of the hidden state representation h_t and the transition model $p(h_{t+1}|h_t, z_t)$, we aim to use general, high capacity neural network models to learn appropriate representation directly from the data. This bypasses the potential misspecification of a hand-engineering approach and can

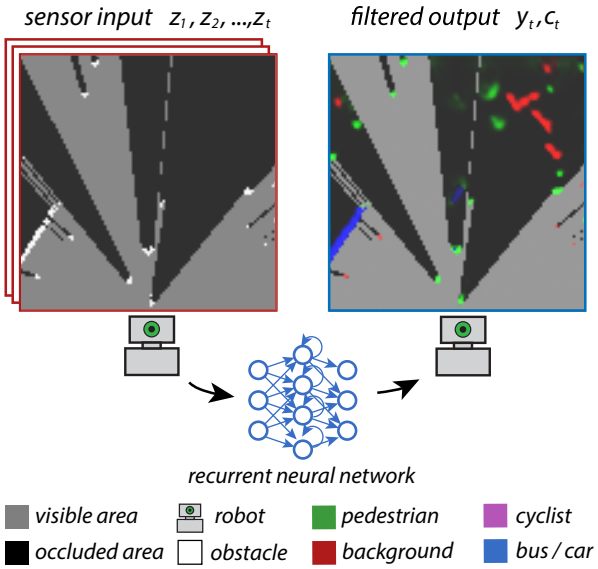


Figure 3.1: Typical output of the proposed system capturing the situation around the robot in the form of a semantic map. The stream of raw sensor data is filtered by a recurrent neural network and produces classification of both directly visible and occluded space into one of several semantic classes.

scale better to high complexity of the real world for which the exact model representation is usually not known.

At the same time we present a new training procedure to train such a network directly from raw sensor observations. We apply this principle to design a novel end-to-end framework for tracking and classifying a robot's surroundings in realistically complex, dynamic and only partially observable environments. To the best of our knowledge this is the first end-to-end object tracking neural network-based approach which directly maps from raw sensor input to object tracks and classes in sensor space without requiring any feature engineering or system identification in the form of plant or sensor models.

We achieve this by first framing the tracking problem as a deep learning task and exploiting sequence models in the form of recurrent neural networks. The neural network filters an input stream of raw laser measurements in order to directly infer object locations, along with their identity in both visible and occluded areas, Figure 3.1. In particular, we propose a method which allows learning in an *unsupervised* manner, only based on

raw, occluded sensor data without access to ground-truth annotations. We show that by learning to track using a large amount of unsupervised data, the network creates a rich internal representation of its environment. We, in turn, exploit this learned representation through the principle of inductive transfer of knowledge to perform the task of its semantic classification. As a result, we show that only a small amount of labelled data is required to steer the network towards mastering this additional task.

We propose two different neural network architectures and demonstrate the tracking and classification performance of the method both in a simulated scenario and on real-world data collected at a busy road junction. Our evaluation shows that the proposed end-to-end framework compares favourably to a state-of-the-art, model-free tracking solution and that it outperforms a conventional one-shot training scheme for semantic classification.

The main contributions are:

- A novel, end-to-end way of filtering partial observations of dynamic scenes using recurrent neural networks to provide a full, unoccluded scene estimation.
- A new method of unsupervised training based on a novel form of dropout encouraging the network to correctly predict objects in unobserved spaces.
- A data-efficient method of tracked object classification based on transfer learning.
- A tailored recurrent neural network architecture to enable tracking and semantic classification in complex, dynamic, *real-world* scenarios.

The work described in this chapter directly contributed to the following publications:

- P. Ondruska and I. Posner, *"Deep Tracking: Seeing Beyond Seeing Using Recurrent Neural Networks,"* in The Thirtieth AAAI Conference on Artificial Intelligence (AAAI), Phoenix, Arizona USA, 2016.

- P. Ondruska, J. Dequaire, D. Wang and I. Posner, *"End-to-End Tracking and Semantic Segmentation Using Recurrent Neural Networks,"* in Workshop of Robotics Science and System (RSS), Detroit, Mitchigan USA, 2016.
  [The best workshop paper award].

The rest of the chapter is organised as follows. We start with reviewing the relevant literature in Section 3.1. In Section 3.2 we formally define the problem of model-free end-to-end tracking and semantic classification, providing its formal solution in Section 3.3. Next, in Section 3.4 we describe the *DeepTracking* framework, which trains a recurrent neural network using large amounts of unsupervised data to track the objects. Then, in Section 3.5 we extend the framework to produce semantic labels using a small amount of supervised data and transfer learning. In Section 3.7 we present an empirical evaluation of our contributions. We conclude in Section 3.8.

3.1 Related Works

Our work concerns modelling partially-observable stochastic processes with a particular focus on the application of object tracking. This problem is commonly solved by Bayesian filtering, which gives rise to tracking methods for a variety of domains [104]. Commonly in these approaches the state representation is designed by hand and the prediction and correction operations are made tractable under a number of assumptions on model distributions or via sampling-based methods. The Kalman filter [45], for example, assumes a multivariate normal distribution to represent the belief over the latent state leading to the well-known prediction/update equations but limiting its expressive power. In contrast, the particle filter [26, 93] foregoes any assumptions on belief distributions and employs a sample-based approach. In addition, where multiple objects are to be tracked, correct data association is crucial for the tracking process to succeed.

In contrast to using such hand-designed pipelines we provide a novel end-to-end train-

able solution, allowing the robot to automatically learn an appropriate belief state representation as well as the corresponding predict and update operations. It is effective in an environment containing multiple objects with different appearance and potentially very complex behaviour. Deep learning approaches have been successful in a number of domains (see, for example, [52, 24, 100]) where they have benefited from large amounts of supervised data in order to learn appropriate internal representations leading to significant performance gains above and beyond that achievable by classical methods. In our case the neural network is trained end-to-end to predict space occupancy and semantic labels directly from the raw laser data. While doing so, it learns to perform *implicit tracking* where the optimal internal representations about the hypotheses of moving objects and respective update procedures of classical tracking are inferred directly from the data.

While we are not the first to leverage the expressive power of neural networks in the context of tracking, prior art in this domain primarily concerned only the detection part of the tracking pipeline [30], [43]. A full neural-network pipeline requires the ability to learn and simulate the dynamics of the underlying system state in the absence of measurements such as when the tracked object becomes occluded. Modelling complex dynamic scenes however requires modelling distributions of exponentially large state representations, such as real-valued vectors having thousands of elements. This was previously explored by various techniques such as non-linear Hilbert Space Embedding [86, 11]. A seminal work in modelling high-dimensional sequences using neural networks was based on using temporal Restricted Boltzman Machines (RBM) [88, 89, 67]. This generative approach allows the modelling of the joint distribution of $p(y, z)$. The downside of using an RBM, however, is the need of sampling, making the inference and learning computationally expensive. In our work, we assume an underlying generative model but, in the spirit of Conditional Random Fields (CRF) [53] we directly model only $p(y|z)$. Instead of an RBM our architecture features feed-forward recurrent neural networks [60, 38] making the inference and computation of the weight gradients exact using a standard back-propagation

learning procedure [81]. Moreover, unlike undirected models, a feed-forward network allows the freedom to straightforwardly apply a wide variety of network architectures such as fully-connected layers and Long-Short Term Memeory (LSTM) networks [41] to design the network. Our architecture is similar to recent Encoder-Recurrent-Decoder [32] and similarly to [83] which features convolutions for spatial processing.

To successfully apply deep learning, an appropriate neural network architecture for the task must be chosen. Abundant literature exists on the topic of finding optimal architectures for different tasks such as convolutional networks for image processing [84] and recurrent neural nets such as LSTMs or Gated Recurrent Units (GRU) [22] for processing sequences. We propose a novel neural network architecture specifically tailored to real-world object tracking. The network shares similarity with architectures for semantic labelling of natural images [105] in terms of the ability to produce output of the same resolution. In addition, we provide effective mechanisms to track objects of different sizes over time, learn place-specific information and recurrent mechanisms to remember information for long periods of time in order to track objects effectively even through long occlusions.

A common drawback of deep learning approaches is the need for large amounts of supervised data for training. We show that our network can be, in fact, trained very efficiently. The network first learns to track by just observing a stream of raw unlabelled sensor data and by trying to predict the next input. In turn we exploit the fact that the learned representation captures latent higher-order information in the data such that we can easily infer semantic labels for the tracked objects, using only a small amount of labelled data. This is a form of *inductive transfer* of knowledge between machine learning tasks [73]. In the context of neural networks it was successfully applied to a range of tasks, in the areas of multi-task learning [66, 16] and in the form of unsupervised pre-training and supervised fine-tuning [75, 54].

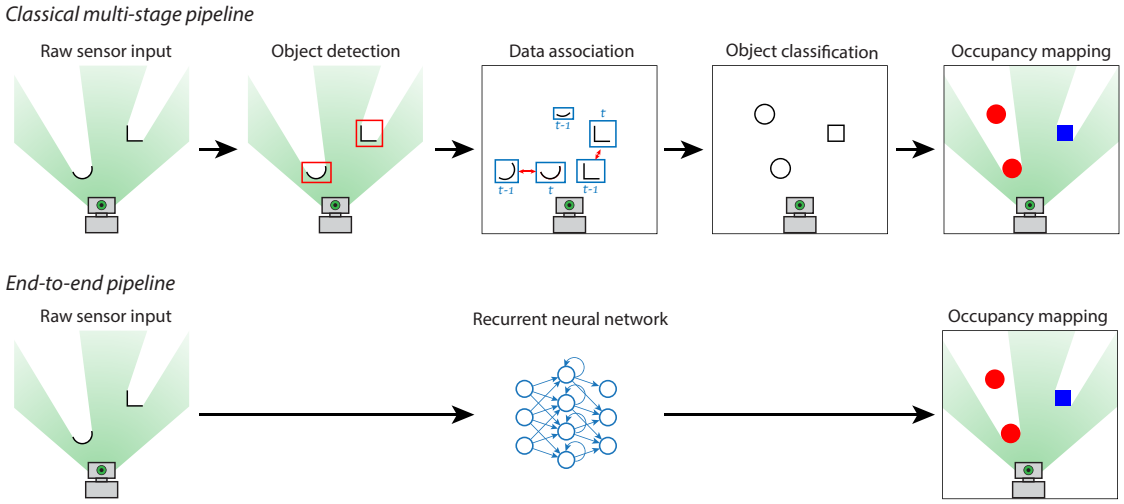


Figure 3.2: Comparison of a classical multi-stage perception pipeline **[top]** to the proposed end-to-end framework **[bottom]**. A multi-stage pipeline requires a significant amount of design effort, and each step in the pipeline introduces its own simplifying assumptions resulting in restrictions on the general applicability of the overall system.

3.2 Problem Formulation

In this section we formulate the problem of effectively tracking the state of the environment around the robot. Classical approaches to this problem such as [64, 99, 104] typically involve a multi-stage tracking pipeline as illustrated in the top row of Figure 3.2. This pipeline features a sequence of explicit and largely hand-engineered steps consisting of object detection considering a stream of sensor input, semantic classification, data association, state estimation (including motion modelling) and, finally, occupancy grid generation. Instead, here we take an end-to-end approach which directly maps from raw laser data to a semantically annotated and unoccluded occupancy grid. This is illustrated in the bottom row of Figure 3.2.

In framing the problem we are inspired by probabilistic occupancy grids [92, 93] and their natural connection in planning algorithms [28]. The input to our problem is a sequence $z_1, z_2, \ldots, z_t$ of *partially observed* occupancy states of the world, computable directly from raw sensor measurements. We represent this state as a discretised 2D grid of

size $M \times M$, parallel to the ground, built locally around the sensor. The partially observed state of the world is represented by two $M \times M$ binary matrices $v_t, q_t \in \{0,1\}^{M \times M}$, collectively referred to as $z_t = \{v_t, q_t\}$. The first matrix v_t encodes whether a cell is directly observable (value of 1, 0 otherwise) by the sensor at time t, while the second matrix q_t encodes whether a cell is observed to be free (a value of 0), or occupied (a value of 1). The output we wish to obtain also consists of two parts. The first part is an *occlusion-free* occupancy state of the world $y_t \in \{0,1\}^{M \times M}$, represented by an occupancy matrix similar to the occupancy matrix in z_t. The second part is a semantic map $c_t \in \{1, ..., K\}^{M \times M}$, revealing, for each cell, which of K types of objects (such as *pedestrian*, *bicyclist*, *car* etc.) is currently occupying it[1]. Note, that for the observed values of the space, where the value of $v_t = 0$ the desired output y_t is equal to q_t.

The problem therefore resolves to solving for $p(y_t, c_t | z_{1:t})$, the probability of the complete state of the world and its semantics at time t, given the observed input at all previous time steps $z_{1:t}$. This formulation can also be used to predict a future state $p(y_{t+n}, c_{t+n} | z_{1:t})$ by simply providing an empty input for $z_{t+1:t+n} = \varnothing$ corresponding to a situation of a fully-occluded sensor. In the next section we first outline a probabilistic formulation of estimating both $p(y_t | z_{1:t})$ and $p(c_t | z_{1:t})$. Then in Sections 3.4 and 3.5 we describe how to train a recurrent neural network to execute this task.

3.3 The Model

In this section we describe a probabilistic formulation to predict the current, unoccluded occupancy scene around the robot given the sequence of sensor observations, i.e. to model $p(y_t | z_{1:t})$ and then extend this formulation to predict semantic labels $p(c_t | z_{1:t})$.

[1] Whereas c_t is modelled for all the cells in practice, it is ignored for cells that are not occupied, i.e. $y_t^i = 0$.

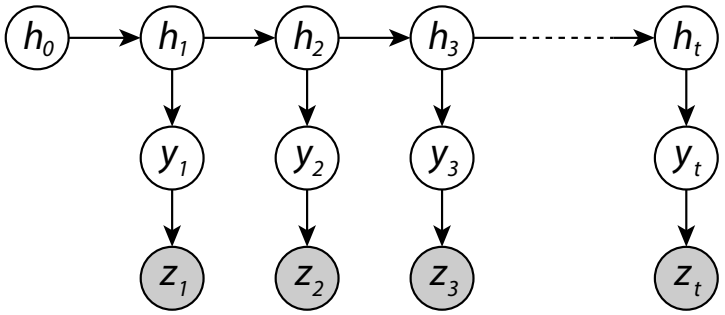


Figure 3.3: The assumed graphical model of the state transition and observation process. Latent world state h_t and its dynamics are modelled by a Hidden Markov Process with appearance y_t which is partially observed by sensor measurements z_t.

3.3.1 Predicting Occupancy

In general, the problem of predicting $p(y_t|z_{1:t})$ cannot be solved effectively if the process y_t is purely random and unpredictable, as in such a case there is no way to determine the unobserved elements of y_t. In practice, however, the sequences y_t and z_t exhibit structural and temporal regularities which can be exploited for effective prediction. For example, objects tend to follow certain motion patterns and have certain appearance. This allows the behaviour of the objects to be estimated when they are visible, such that this knowledge can be used later for prediction when they become occluded.

Like many approaches to object tracking, our model is inspired by Bayesian filtering [20]. Note, however, that the occupancy process y_t does not satisfy the Markov property: $p(y_t|y_{1,2,t-1}) \neq p(y_t|y_{t-1})$ knowledge of y_{t-1} at any particular time $t-1$ provides only partial information about the state of the world such as object positions, and does not contain all necessary information such as their speed or acceleration required for prediction. The latter can be obtained only by relating subsequent measurements. Methods such as the Hidden Markov Model [78] form Chapter 2 are therefore not directly applicable.

To handle this problem we assume the generative model in Figure 3.3 such that along-

side y_t there exists another underlying Markov process, h_t [2], which completely captures the state of the world with the joint probability density

$$p(y_{1:N}, z_{1:N}, h_{1:N}) = \prod_{t=1}^{N} p(z_t|y_t)p(y_t|h_t)p(h_t|h_{t-1}), \tag{3.1}$$

where

- $p(h_t|h_{t-1})$ denotes the hidden state transition probability capturing the dynamics of the world;
- $p(y_t|h_t)$ is modelling the instantaneous *unoccluded* sensor space;
- and $p(z_t|y_t)$ describes the actual sensing process.

The original task of estimating $p(y_t|z_{1:t})$ can now be framed in the context of recursive Bayesian estimation [8] of the *belief* $\mathcal{B}_t$ which, at any point in time t, corresponds to a distribution $Bel(h_t|z_{1:t}) = p(h_t|z_{1:t})$. This belief can be computed recursively for the considered model as

$$Bel(h_t|z_{1:t-1}) = \int_{h_{t-1}} p(h_t|h_{t-1})Bel(h_{t-1}|z_{1:t-1}) \tag{3.2}$$

$$Bel(h_t|z_{1:t}) \propto \int_{y_t} p(z_t|y_t)p(y_t|h_t)Bel(h_t|z_{1:t-1}), \tag{3.3}$$

where $Bel(h_t|z_{1:t-1})$ denotes *belief prediction* one time step into the future and $Bel(h_t|z_{1:t})$ denotes the *corrected* belief after the latest measurement has become available. The cell occupancy y_t can be predicted simply from this belief,

$$p(y_t|z_{1:t}) = \int_{h_t} p(y_t|h_t)Bel(h_t|z_{1:t}). \tag{3.4}$$

This formulation expresses the desired output in the form of an abstract probability distribution computable from elementary inference steps. In Section 3.4 we describe a method

[2] In our implementation h_t is a vector of size $M \times M \times Z$.

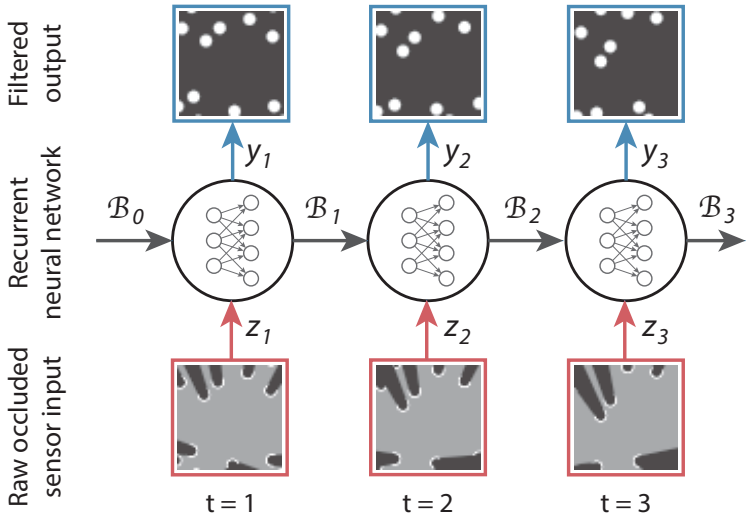


Figure 3.4: An illustration of the filtering process using a recurrent neural network.

of training a recurrent network to provide their effective implementation.

3.3.2 Semantic Segmentation

As the hidden state h_t captures the state of the world including class labels, the semantic segmentation can be defined similarly to Equation 3.4:

$$p(c_t|z_{1:t}) = \int_{h_t} p(c_t|h_t)Bel(h_t|z_{1:t}). \tag{3.5}$$

This implies a similar method can be applied for its inference.

3.4 Learning to Track with a Recurrent Neural Network

The previous chapter provided a general Bayesian framework for describing $p(y_t|z_{1:t})$. Here we describe how this formulation leads to a method of training a neural network to compute its effective inference. This procedure can be then, in theory, used to train any recurrent network to perform the task. Choosing an appropriate network architecture is

however still important, and we describe this part in Section 3.6. Computation of $p(y_t|z_{1:t})$ can be carried by iteratively updating the current belief $\mathcal{B}_t$

$$\mathcal{B}_t = F(\mathcal{B}_{t-1}, z_t), \tag{3.6}$$

defined by (3.2) and (3.3), and simultaneously providing prediction

$$p(y_t|z_{1:t}) = p(y_t|\mathcal{B}_t), \tag{3.7}$$

as defined in (3.4). For realistic applications, this approach is limited by the need for the suitable belief state representation as well as the explicit knowledge of the distributions in (3.1) modelling the generative process.

The key idea of our solution is to avoid the need to specify this knowledge and instead use a highly expressive neural networks, governed by weights w_F and w_P, to model both $F(\mathcal{B}_{t-1}, z_t)$ and $p(y_t|\mathcal{B}_t)$, allowing us to learn their function directly from the data. Specifically, we assume $\mathcal{B}_t$ can be approximated and represented as a vector, $\mathcal{B}_t \in \mathbb{R}^M$. This is a form of distributed representation that captures all relevant objects and properties on the scene. As illustrated in Figure 3.4, the function F representing state evolution is then a recurrent mapping from $\mathbb{R}^M \times \mathbb{R}^{2N} \rightarrow \mathbb{R}^M$ corresponding to an implementation by a feed-forward recurrent neural network [60] where the $\mathcal{B}_t$ acts as the network's memory, passed from one time step to the next.

Importantly, we do not impose any restrictions on the content or function of this hidden representation. Provided that both networks for $F(\mathcal{B}_t, z_t)$ and $p(y_t|\mathcal{B}_t)$ are differentiable, they can be trained together end-to-end as a single recurrent network with a cost function directly corresponding to the data likelihood provided in (3.4). This allows the neural networks to adapt to each other and to learn an optimal hidden state representation for $\mathcal{B}_t$ together with procedures for belief updates using partial observations z_t, and to use this information to decode the final scene state y_t. This network can be trained both in

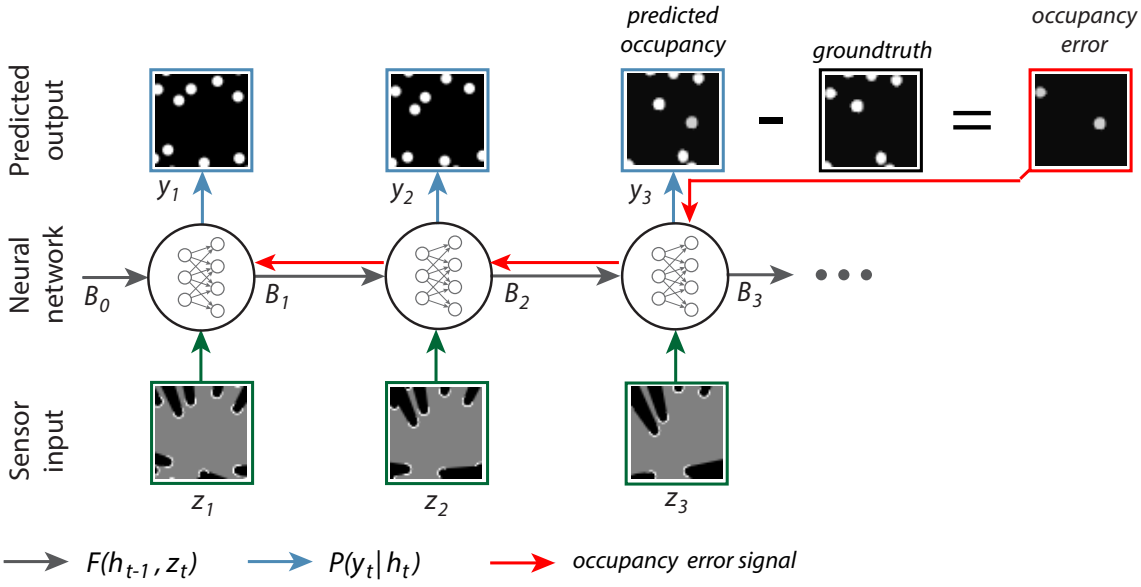


Figure 3.5: Supervised training of the recurrent neural network to produce space occupancy y_t. The network is trained to predict output consistent with future inputs.

a supervised mode, i.e. with both known $y_{1:N}, z_{1:N}$ and in an unsupervised mode where only $z_{1:N}$ is known. This ability to train the network in an unsupervised mode is a crucial property, allowing us to train the system rapidly and efficiently by simply collecting a long enough data log from a sensor placed in the field.

3.4.1 Supervised Training

A common method of supervised training is to minimise the negative log-likelihood of the true scene state given the sensor input,

$$\mathcal{L} = -\sum_{t=1}^{N} \log p(y_t|z_{1:t}). \tag{3.8}$$

This objective can be optimised by the gradient descent with partial derivatives given by backpropagation through time, [80]

$$\frac{\partial \mathcal{L}}{\partial \mathcal{B}_t} = \frac{\partial F(\mathcal{B}_t, z_{t+1})}{\partial \mathcal{B}_t} \cdot \frac{\partial \mathcal{L}}{\partial \mathcal{B}_{t+1}} - \frac{\partial \log p(y_t|\mathcal{B}_t)}{\partial \mathcal{B}_t} \tag{3.9}$$

$$\frac{\partial \mathcal{L}}{\partial w_F} = \sum_{t=1}^{N} \frac{\partial F(\mathcal{B}_{t-1}, z_t)}{\partial w_F} \cdot \frac{\partial \mathcal{L}}{\partial \mathcal{B}_t} \tag{3.10}$$

$$\frac{\partial \mathcal{L}}{\partial w_P} = -\sum_{t=1}^{N} \frac{\partial \log p(y_t|\mathcal{B}_t)}{\partial w_P}, \tag{3.11}$$

as is illustrated in Figure 3.5.

However, a major disadvantage of the supervised training is the requirement of the ground-truth scene state y_t contains parts not visible by the robot sensor. One way of obtaining this information is to place additional sensors in the environment which jointly provide information about every part of the scene. Such an approach, however, can be impractical, costly or even impossible. In the following section we describe a training method which only requires raw, occluded sensor observations alone. This dramatically increases the practicality of the method, as the only necessary learning information is a long enough record of $z_{1:N}$, as obtained by the sensor in the field.

3.4.2 Unsupervised Training

The aim of unsupervised training is to learn $F(\mathcal{B}_{t-1}, z_t)$ and $p(y_t|\mathcal{B}_t)$ using only $z_{1:t}$. This might seem impossible without knowledge of y_t as there would be no way to correct the network predictions. However, *some* values of y_t as $z_t \subseteq y_t$ are known in the directly observed sensor measurements. A naive approach would be to train the network using the objective from (3.8) of predicting the known values of y_t and not back-propagating from the unobserved ones. Such an approach would fail, however, as the network would only learn to copy the observed elements of z_t to the output and never predict objects in occlusion.

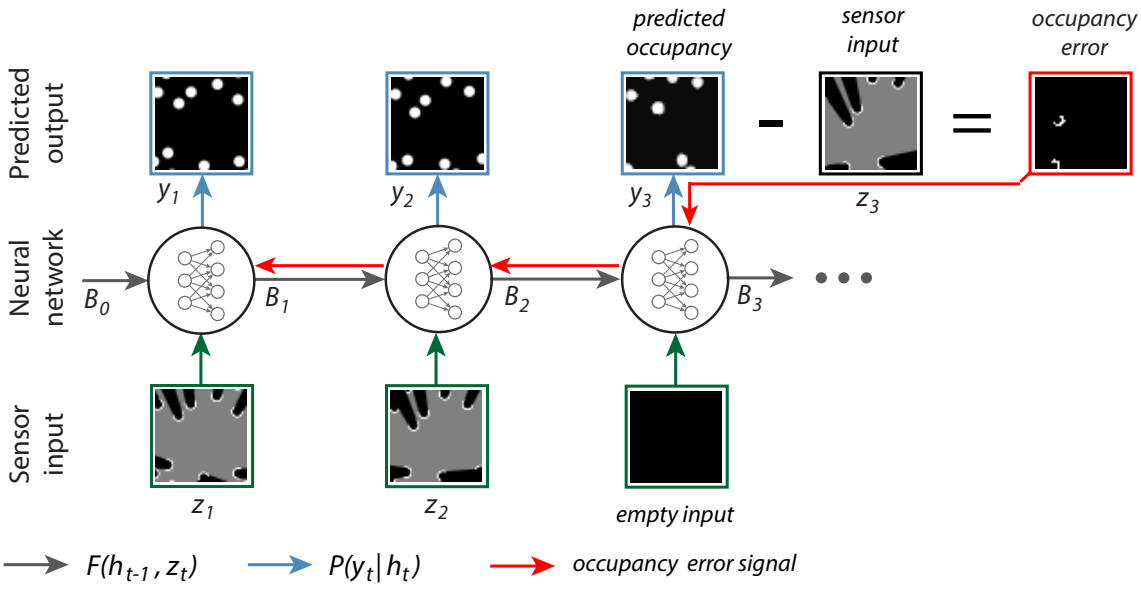


Figure 3.6: Unsupervised training of the recurrent neural network to produce space occupancy y_t. The network is trained to predict output consistent with future inputs. This allows training without the need of ground-truth information of the full, unoccluded scene.

Instead, and crucially, we propose to train the network not to predict the current state $p(y_t|z_{1:t})$ but a state in the future $p(y_{t+n}|z_{1:t})$. In order for the network to correctly predict the visible part of the object after several time steps the network must learn how to represent and simulate object dynamics during the time when the object is not seen using F and then to convert this information to predict $p(y_{t+n}|\mathcal{B}_{t+n})$.

This form of learning can be implemented easily by considering the cost function from (3.8), predicting visible elements of $y_{t:t+n}$ and leaving out all $z_{t:t+n}$ by setting values of both cell visibility and obstacle to 0 marking the input unobserved. This procedure is demonstrated in Figure 3.6.

The key result is that, in order for the network to predict the true occupancy, it is forced to learn how to use its memory, propagate information over time and use this information when it lacks direct scene observations. This situation occurs both in predicting the future occupations and predicting true occupation in occluded parts of the scene. As demonstrated in Section 3.7, this training procedure leads to the regularised network's

ability to predict true positions of the objects whenever they are not visible, effectively providing tracking.

3.5 Learning to Understand with Knowledge Transfer

In this section, we extend our solution to the partial problem $p(y_t|z_{1:t})$ presented in Section 3.4 to the full problem of simultaneously estimating both occlusion-free occupancy y_t and scene semantics c_t. An appealing way to achieve this is by exploiting the knowledge the network has already learned to predict y_t, through the principle of inductive transfer [73]. The significance of this is we show that *only a small amount* of labelled training data is needed to allow the same network to master this additional task.

The clue resides in the hidden representation h_t learned in the unsupervised training for tracking, which can be viewed as a universal descriptor of the state of the world. It captures not only the positions of individual objects, but also their motion patterns, shapes and other properties necessary for the successful prediction of scene dynamics. Because the network was trained to perform well in this task, a reasonable assumption is that the information about object shapes and velocities necessary for predicting their positions in the near future is already contained in this hidden representation. This pre-trained kind of information provides rich and structured context for predicting object semantic class. Similarly to predicting y_t from h_t in (3.7), extracting c_t can be therefore be achieved simply by building a classifier to predict $p(c_t|h_t)$.

3.5.1 Training

However, training a classifier to extract semantic information from $\mathcal{B}_t$ is not straightforward. This is because a supervisory signal would need to be provided for all the pixels, whether they contain an actual object or not, and for both visible and occluded areas. Such a supervised data set would be very difficult to produce if only occluded raw laser

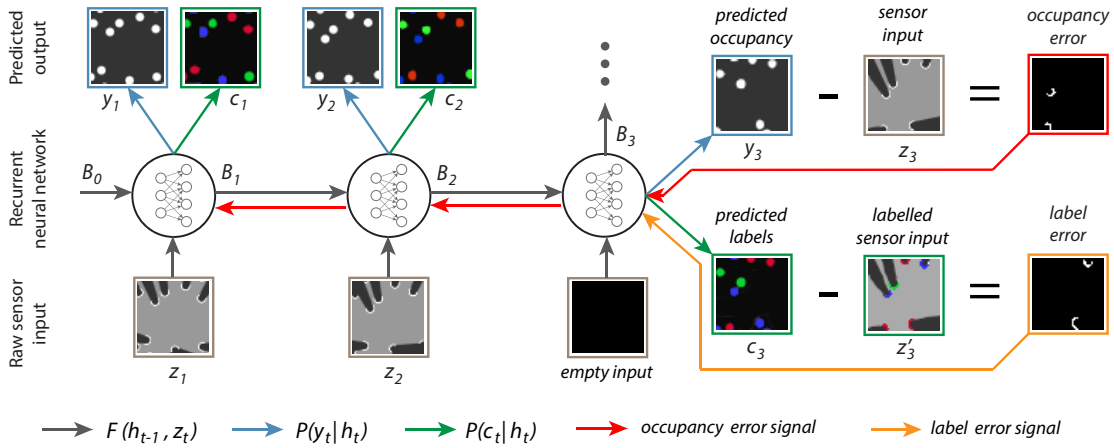


Figure 3.7: Training of the recurrent neural network to produce both space occupancy y_t and semantic labels c_t. The network is trained to predict output consistent with future inputs. This allows training without the need of ground-truth information of the full, unoccluded scene. First the network learns how to track by predicting correct occupancy using large amounts of unlabelled data, then a small set of labelled data is used to induce semantic classification.

scans are available.

Instead, we label only the visible cells of the available raw input data which contain an actual object. Then, we predict labels for all the pixels, but back-propagate the error only on those with a label. However, if used off-the-shelf, this procedure suffers from generalisation problems as the network is trained to predict labels of only visible objects and never the occluded ones. To ensure good performance when classifying *occluded* objects, we used the same principle as in Section 3.4.2: we train the network to predict the future semantic label c_{t+n} while providing only inputs $z_{1:t}$, forcing the network to use information stored in memory $\mathcal{B}_t$ and then back-propagate the error compared to the true label of z_{t+n}. The entire process is illustrated in Figure 3.7.

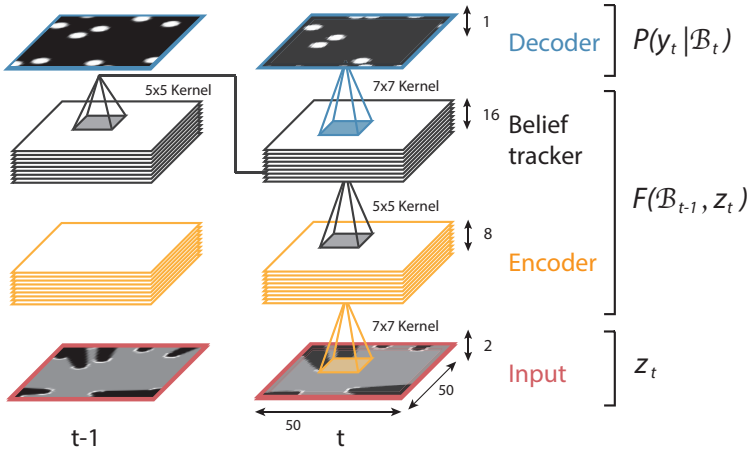


Figure 3.8: The four-layer simple recurrent network architecture used for the experiments on the synthetic data set. Directly visible objects are detected in the *encoder* and fed into the *belief tracker* to update belief $\mathcal{B}_t$ used for scene de-occlusion in *decoder*.

3.6 Neural Network Architectures

The previous section described a general training procedure which can be used to train a general black-box recurrent neural network to perform the task. Selecting an appropriate network architecture is however a critical part to successfully apply deep learning methods. In this section we detail two different network architectures used in our experiments. We start with with a simple small-scale neural network for proof-of-principle and to gain insights into the problem, conducting experiments on our synthesised data set described in Section 3.7. Then, we scale the architecture up for the needs of realistic tracking in a real-world environment. Finally, we describe how this architecture can be used to produce both cell occupancy and semantic labels.

3.6.1 Simple Architecture

To jointly model $F(\mathcal{B}_{t-1}, z_t)$ and $p(y_t|\mathcal{B}_t)$ we use a small feed-forward recurrent neural network as illustrated in Figure 3.8. This architecture has four layers and uses convolutional operations followed by a sigmoid non-linearity as the basic information pro-

cessing step at each layer. The network has, in total, 11k parameters, and its hyper-parameters such as number of channels in each layer and size of the kernels are set by cross-validation. The belief state $\mathcal{B}_t$ is represented by the third layer of size $50\times50\times16$, which is propagated between time steps.

The mapping $F(\mathcal{B}_{t-1}, z_t)$ is composed of input-preprocessing (the *encoder*) followed by a stage of hidden state propagation (the *belief tracker*) both implemented by a neural network. The aim of the *encoder* is to analyse the sensor measurements, to perform operations such as visible object detection and to convert this information into a $50\times50\times8$ embedding, used as input to the *belief tracker*. This information is concatenated with the previous belief state $\mathcal{B}_{t-1}$ and is updated in the *belief tracker* into a new belief state $\mathcal{B}_t$, effectively providing recurrent information flow.

Finally, $p(y_t|\mathcal{B}_t)$ is modelled by a Bernoulli distribution by interpreting the final network layer output as a *probabilistic occupancy grid* p_t, denoting the likelihood of a given pixel being part of an object, giving rise to the total joint probability of observing occupancy y_t,

$$p(y_t|p_t) = \prod_i (p_t^i)^{y_t^i}(1-p_t^i)^{(1-y_t^i)}, \tag{3.12}$$

with its logarithm corresponding to the binary cross-entropy. One pass through the network takes 10ms on a standard laptop with a 2.6GHz Intel Core i7 CPU and 8GB of RAM. We evaluate performance of this network in Section 3.7 and show that this network is able to effectively learn to predict the positions of objects in a simple simulated scenario of moving balls.

3.6.2 Complex Architecture

The network in Section 3.6.1 is designed to track small same-size objects in a simulated scenario. An effective deployment in real-world robotics applications, however, poses a set of challenges for which a more appropriate architecture must be chosen. In particular

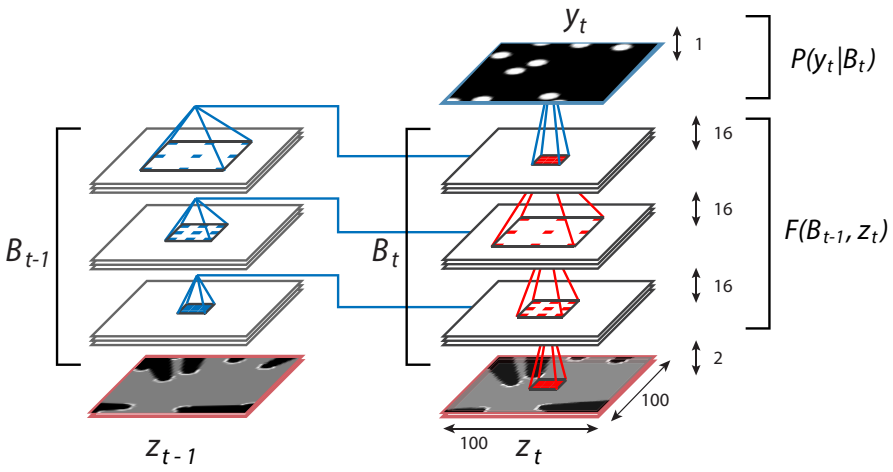


Figure 3.9: The proposed complex architecture for tracking in realistic scenarios. It features dilated convolution, enhanced static and dynamic memory capabilities producing information of cell occupancy at the output.

it requires the ability to simultaneously track objects of different sizes such as cars and pedestrians, to remember information for long periods of time, to deal with occlusions, learn and exploit place-dependent information such as the presence of static obstacles, and lastly to produce output for both space occupancy and its semantic label. In this section we present a new network architecture designed to address these issues.

An overview of our proposed network is depicted in Figure 3.9. Similarly to the simple architecture from the previous section, the input z_t at time t is processed by a multi-layer network. At each layer the output of the previous layer is combined with its own activations at time $t-1$ implementing the recurrence. This allows the network to extract and remember the information from the past and use it for prediction at time t. The output of the final layer is then converted into the resulting output y_t through a simple convolutional decoder.

Unlike classical convolutional networks such as [85], this network *does not* feature max pooling and maintains the same resolution in each layer. This is important as we specifically want to keep translational equivariance of convolution but avoid translational invariance of pooling, a feature which is otherwise desirable in object classification, but

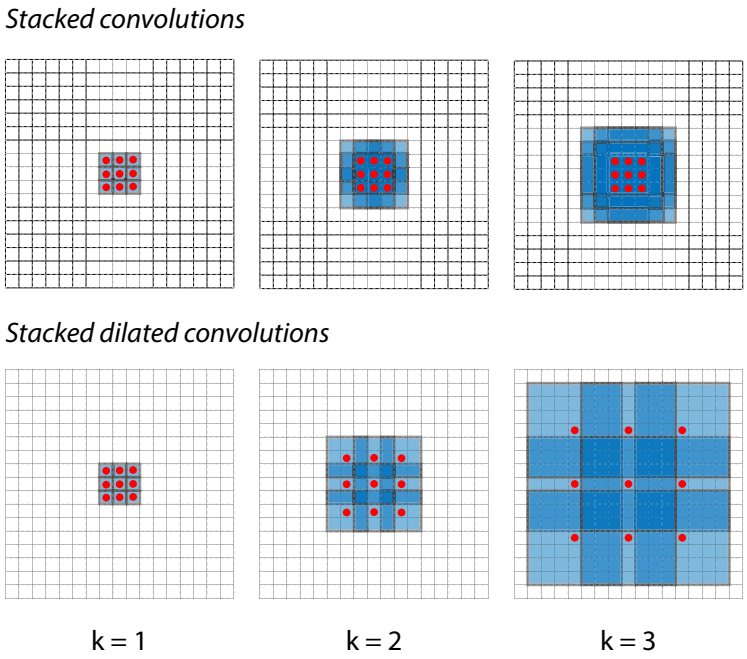


Figure 3.10: Multi-scale context aggregation, preserving the image resolution by stacking *dilated convolutions* [105]. At layer k the convolution is performed over the window of 3×3 pixels 2^{k-1} pixels apart. This results in exponential growth of the blue *receptive field size* **[bottom]** as opposed to stacking classical convolutions resulting only in linear growth **[top]**.

results in a decrease in spatial accuracy. To achieve properties necessary for tracking in realistic scenarios (tracking objects of different sizes, remembering information for long time, and learning place-specific information) we employ four components: *multi-scale convolution*, *dynamic memory*, *static memory*, and a *pair of decoders* which we describe below.

3.6.2.1 Multi-Scale Convolution

For the network to correctly predict whether any particular cell is occupied by a moving object this object must fall in the *receptive field* of the output unit in the final layer. The *receptive field* is the part of the input affecting the value of the unit. In the case of classical convolution, the *receptive field* is the $K \times K$ neighbourhood where K is the size of the convolution kernel. The size of the receptive field, however, puts an upper limit on the

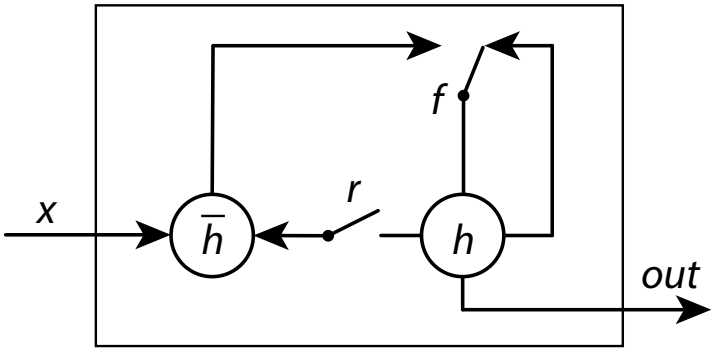


Figure 3.11: Diagram of *Gated Recurrent Unit* [22] used to remember information over time.

size of effectively tractable objects in the input, which can be of vastly different sizes in realistic settings. Two ways to increase the size of the receptive field are to increase the kernel size or stack multiple convolutions on top of one other. These, however, creates a computational challenge as the number of parameters and computational complexity grow quadratically with K in the first case and linearly in the second case.

Instead, we use a stack of dilated convolutions [105] where the receptive field grows exponentially with the number of layers, leading to the ability to track large objects. The basic idea is to perform the classical 3×3 convolution but skipping $2^{k-1} - 1$ pixels in between convolved pixels at layer k, as illustrated in Figure 3.10. This gives a $(2^{K+1} - 1) \times (2^{K+1} - 1)$ receptive field at the final layer K. This dilated convolution is then used as an elementary computation step to implement the *dynamic memory* described below.

3.6.2.2 Dynamic Memory

To be able to track a moving object through extended periods of occlusion, the network must remember the location of the object and other properties such as its shape and velocity. Findings from studies on recurrent neural networks stress the importance of specially dedicated units such as *long short term memory* [41] to support information-caching, as otherwise training suffers from the vanishing gradient problem [74]. Inspired by [103],

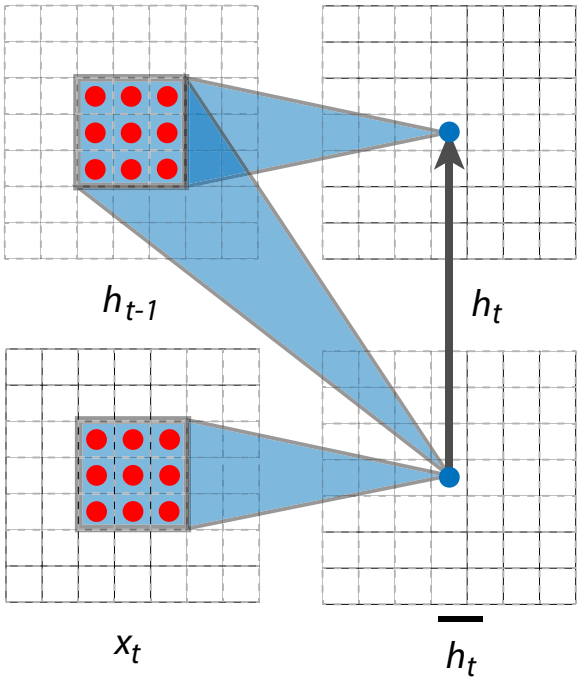


Figure 3.12: Remembering the information for extended periods of time with gated convolutional recurrent units. The new activation h_t is formed by mixing its old activation h_{t-1} and the input z_t.

we implement a convolutional variant of *gated recurrent units* [22] as the processing step at each layer. As illustrated in Figure 3.11, the output of each unit is given by the weighted combination of its previous output at time $t-1$ and a candidate memory $\bar{h}_t$ computed from the output of the layer, with the forgetting of information controlled by the reset gate r_t:

$$f_t = \sigma(W_{xz} * x_t + W_{hz} * h_{t-1} + b_z) , \tag{3.13}$$

$$r_t = \sigma(W_{xr} * x_t + W_{hr} * h_{t-1} + b_r) , \tag{3.14}$$

$$\bar{h}_t = \tanh(W_{xh} * x_t + r_t \circ W_{hh} * h_{t-1} + b_h) , \tag{3.15}$$

$$h_t = f_t \circ h_{t-1} + (1 - f_t) \circ \bar{h}_t . \tag{3.16}$$

Here $*$ denotes dilated convolution described earlier, and $\circ$ denotes element-wise multiplication also known as a Hadamard product. The entire operation is illustrated on Figure 3.12.

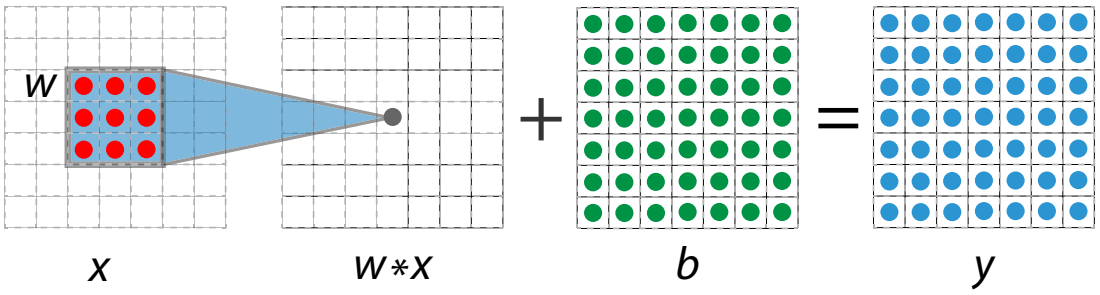


Figure 3.13: Implementation of static memory. As opposed to classical convolution each unit has its own variable bias allowing to remember unit-specific information.

3.6.2.3 Static Memory

We allow each cell to learn a unique piece of information, different from all other cells, to remember place-specific information. This is achieved by biases b_z, b_r, and b_h in (3.13)-(3.15), which are not a per-layer constant as in the case of classical convolution, but instead are learned individually for each unit during the training, Figure 3.13. This is shown in Section 3.7, and allows the network to learn place-specific information such as the static occupancy of the cell or the usual motion patterns and classes in a particular area, which can then be used to aid the network prediction.

3.6.3 Multi-task Architecture

To extend the ability of the network to produce multiple outputs, we employ a pair of simple convolutional decoders to decode output of the final layer to the cell occupancy y_t and class label c_t. The difference is that for the class label we employ softmax (multinomial logistic regression) output for the K classes instead of a sigmoid function.

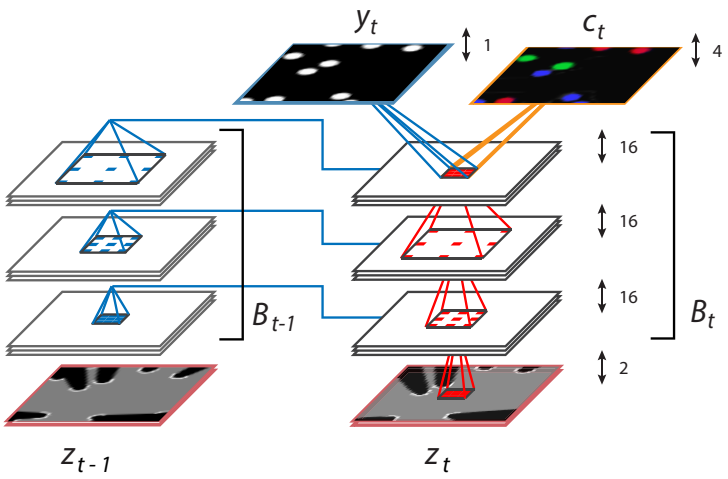


Figure 3.14: The proposed architecture for tracking and semantic classification. It features a pair of decoders, producing both cell occupancy and semantic class information.

3.7 Experiments

In this section we demonstrate the efficacy of the proposed system in both the tracking and semantic classification tasks, in a complex, real-world scenario evaluated in a series of experiments. We show that the trained network is able to track and classify a variety of different objects even through complete occlusion, and is able to predict the evolution of the scene in the near future. We will investigate several properties of the proposed system, such as its absolute accuracy of predicting object positions and their class, it's robustness to noise, ability to predict future evolution several time steps into the future, and compare it against the best alternative methods to solve this task. We show that in all these tasks the system provides excellent performance.

We start by demonstrating the effectiveness of our approach in estimating the full scene state in a simulated scenario, representing a robot equipped with a 2D laser scanner surrounded by many dynamic objects with the simple network proposed in Section 3.6. This scenario provides an important insight into the ability of the proposed approach to successfully solve a simplified task. Next, we scale the method to real-world settings by conducting experiments on real data coming from a range sensor and using the more

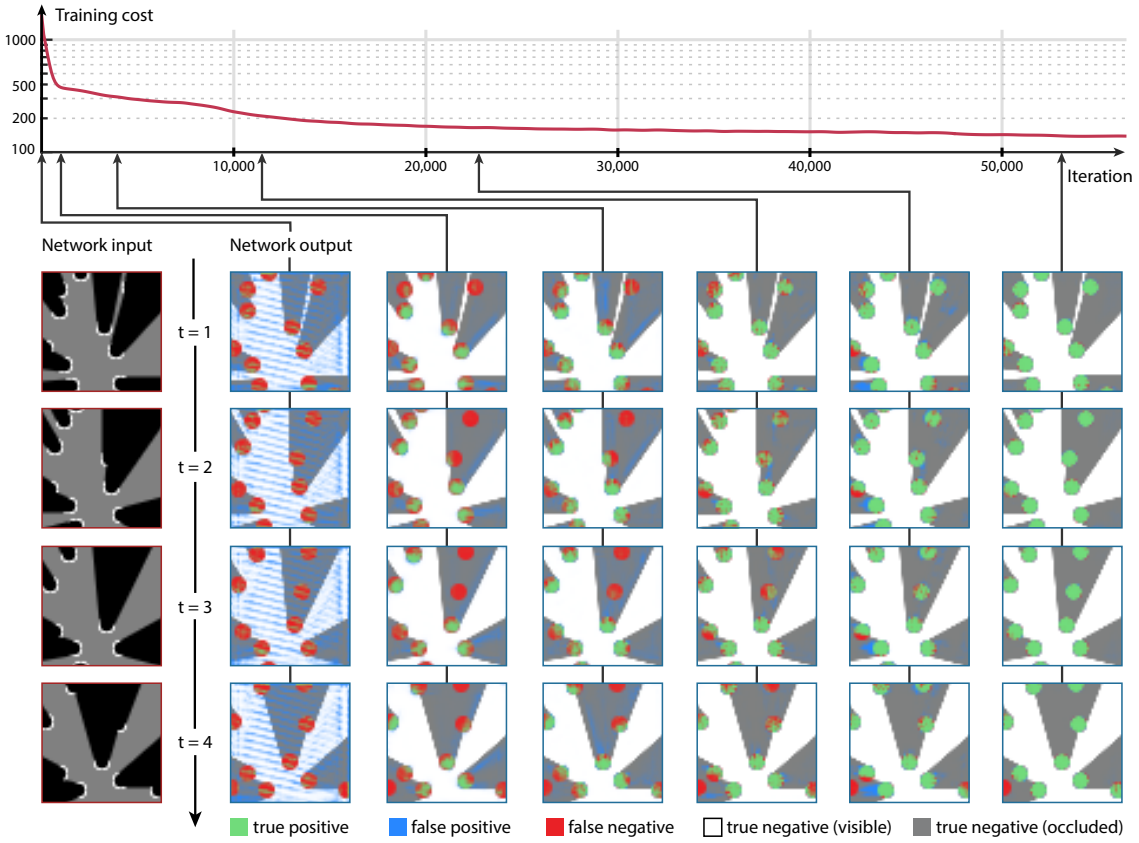


Figure 3.15: The unsupervised training progress. From left to right, the network first learns to complete objects and then it learns to track them through occlusions. Some situations cannot be predicted correctly, such as the object in occlusion on the fourth frame, because it is never observed. Note that the ground-truth for comparison is used only for evaluation, and not during training.

complex neural network from Section 3.6. Finally, we measure the ability of the network to predict both cell occupancy and class labels. We show that the achieved tracking accuracy in both tracking and classification is superior to that of an alternative state-of-the-art model-free tracking solution targeted at the same problem [99].

3.7.1 Synthetised Experiments

Objects in our synthesised data set are modelled as circles independently moving with constant velocity in random directions, but never colliding with each other or with the

robot. Our inspiration here is a situation with a robot in the middle of a busy street surrounded by a crowd of moving people. The number of objects present in the scene is varied over time, with between two and twelve new objects randomly appearing and later disappearing, when they become too distant from the sensor.

At each time step we simulate the sensor input as commonly provided by a planar laser scanner. We divide the scene y_t around the robot into a 2D grid of 50×50 pixels with the robot placed towards the bottom centre. To model sensor observations z_t, we ray-trace the visibility of each pixel and assign values $z_t^i = \{v_t^i, r_t^i\}$ indicating pixel visibility (by the robot) and the presence of an obstacle, if that pixel is both visible and occupied. An example of the input y_t and corresponding observation z_t is shown in Figure 3.4. Note that, as in the case of a real sensor, only the surface of the object as seen from a particular viewpoint is ever observed. This input is then fed into the network as a stream of 50×50 2-channel binary images for filtering.

3.7.1.1 Training

We generate 10,000 sequences of length 200 time steps and train the network for a total of 50,000 iterations using stochastic gradient descent with learning rate 0.9. The initial belief $\mathcal{B}_t$ is modelled as a free parameter, being jointly optimised with the network weights W_F, W_P.

3.7.1.2 Evaluation

The most appealing property of our framework is the ability to train it without knowledge of the actual ground truth. In this section we empirically demonstrate and evaluate this property. Figure 3.15 illustrates the unsupervised training progress. Initially, the network produces random output, then it gradually learns to predict the correct shape of the visible objects and finally it learns to track their position even through complete occlusion. The fully trained network is able to confidently and correctly start tracking an object immedi-

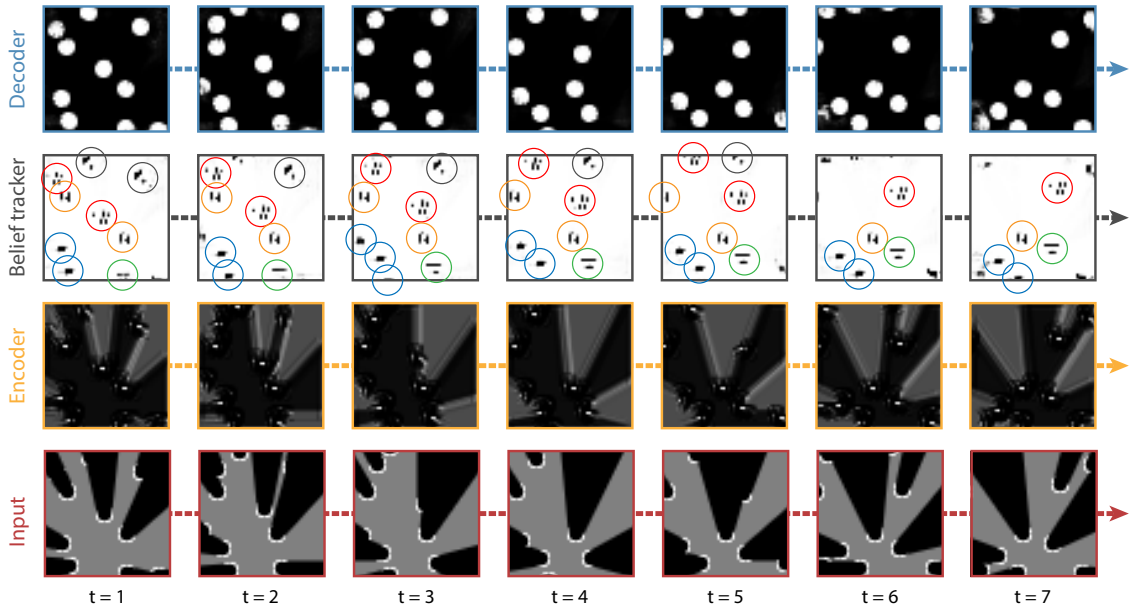


Figure 3.16: Examples of the filter activations from different parts of the network (one filter per layer). The *encoder* adapts to produce a spike at the position of directly visible objects. Note the structure of the *belief* $\mathcal{B}_t$: as highlighted by super-imposed circles in the second row, it adapted to assign different activation patterns to objects having different motion patterns. (Best viewed in colour.)

ately after it has seen even only partially, and is then able to confidently keep predicting its position even through long periods of occlusion.

Finally, Figure 3.16 shows example activations of layers from different parts of the network. It can be seen that the *encoder* module learns to produce, in one of its layers, a spike at the center of visible objects. Particularly interesting is the learned structure of the *belief state*. Here, the network learned to represent hypotheses of objects having different motion patterns with different patterns of unit activations, and to track this pattern from frame to frame. None of this behaviour is hard-coded and it is a pure result of the network adaptation to the task.

Additionally, we perform experiments where the shape of objects is changed from circles to squares, and also simulating sensor noise on 1% of all pixel observations. This is demonstrated in Figure 3.17. In all cases the network correctly learned and predicted the world state, suggesting the approach and the learning procedure is able to perform well in a variety of scenarios.

3.7.2 Real-world Experiments

We collected a 75 minute data set from a stationary robotic platform equipped with a *Hokuyo UTM-30LX* 2D laser scanner, positioned in the middle of a busy urban intersection, as depicted in Figure 3.18. The area features dense traffic composed of buses, cars, cyclists and pedestrians, which results in extensive amounts of occlusion. Consequently, at no point in time is the complete scene fully observable. As there is a lot of redundancy in laser sensor measurements, we sub-sample the data set at 8Hz and split it into a 65 minute unsupervised set to train the network, and a 10 minute test set to measure the occupancy prediction performance. In addition, we hand-labelled 800 scans from the training set into four classes for the purpose of network semantic training, and 200 scans from the test set for the evaluation of its semantic classification accuracy. The classes considered are *pedestrian*, *car/bus*, *cyclist* and *background* (static obstacle).

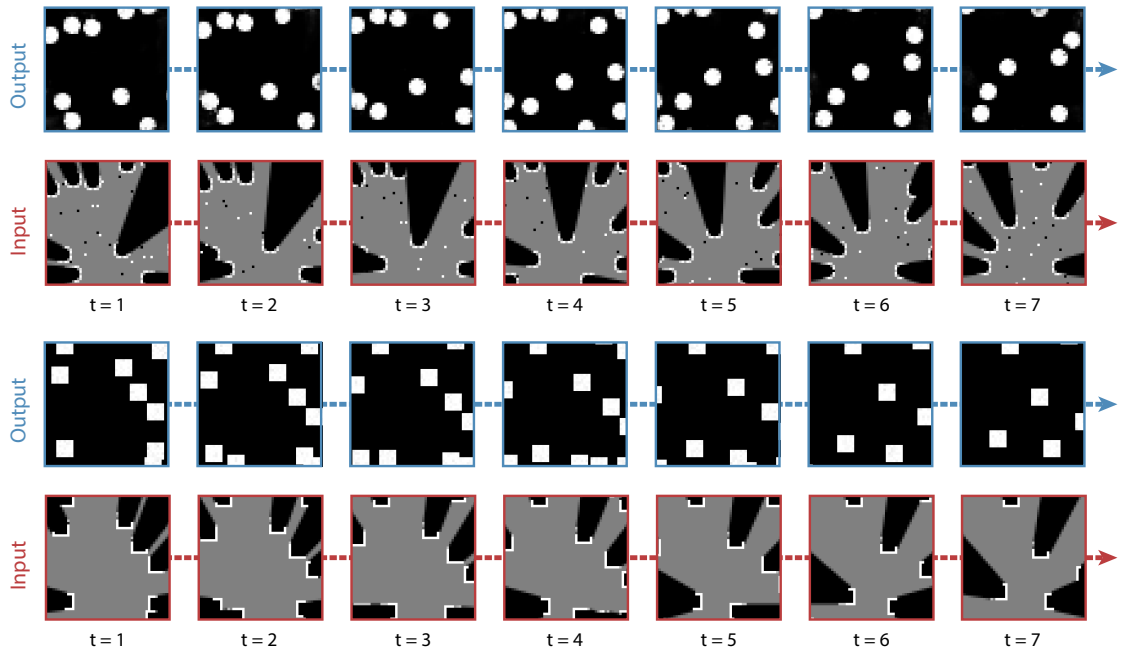


Figure 3.17: Evaluating robustness of the trained model by adding noise by randomly flipping 1% of the pixels **[top]** and on objects of different shape **[bottom]**. In both cases the network correctly learns to predict true object locations.



Figure 3.18: Location of the experiment from the robot's point of view with a superimposed illustration of laser measurements. The area is occupied by a variety of different dynamic objects such as pedestrians, cyclists and cars.

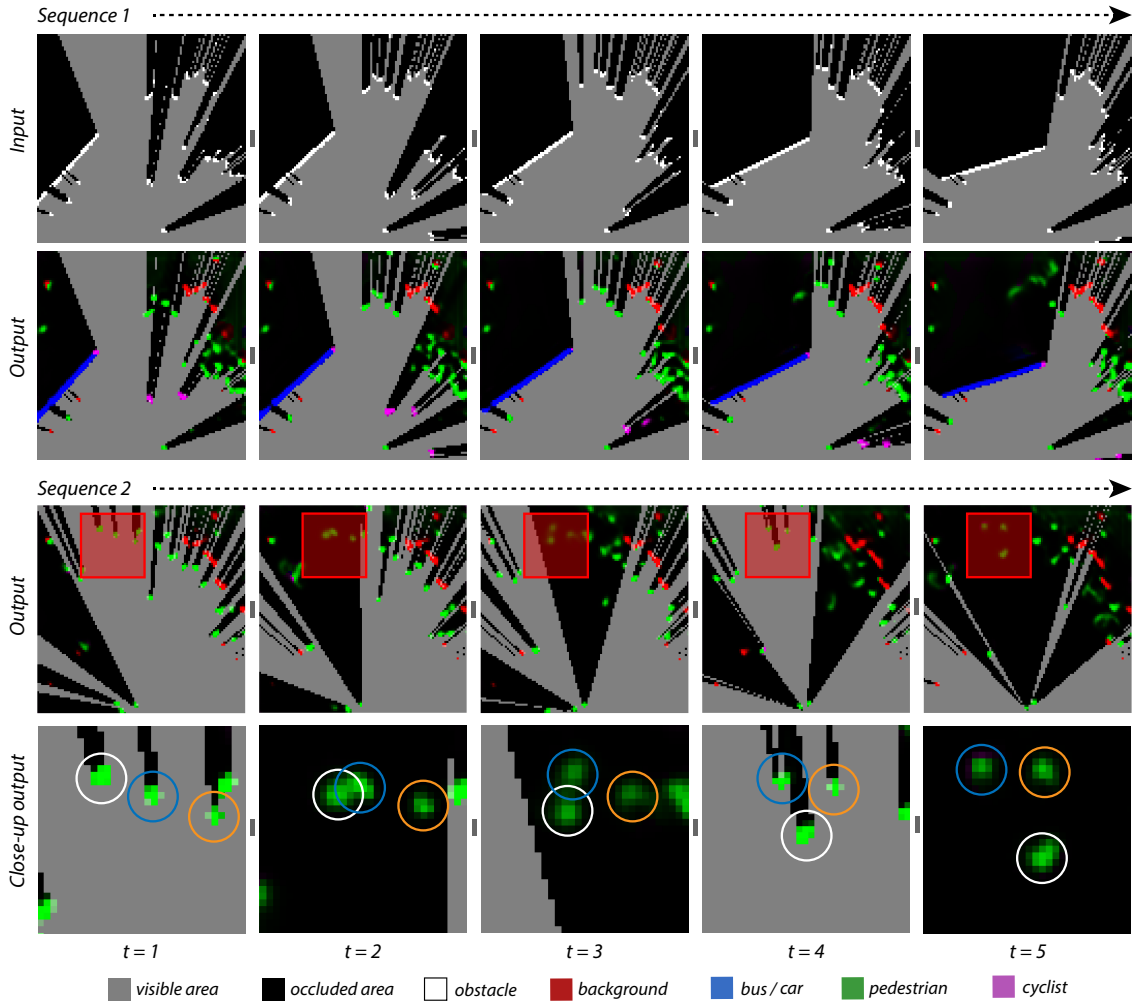


Figure 3.19: Example of produced outputs of the system. As displayed in the highlighted close-up of the output of second sequence, the network is able to propagate assumed motion of the objects (highlighted by circles) even when in complete occlusion.

The input to the network z_t (the partially observed occupancy grid) is computed from raw 2D laser scans by ray-casting. Cells where a laser measurement ends are marked as occupied, all cells from the sensor origin up to the end of the ray are marked as free, and cells beyond the ray are marked to be unobserved. We build a 100×100 grid around the sensor, where each cell covers a 20×20 cm^2 area, the input grid thus spanning a total area of 20×20 m^2.

3.7.2.1 Network Training

We test the proposed large-scale network architecture described in Section 3.6.2, configured with three hidden layers of 16 channels per layer. This network has a total number of 1,010,193 parameters and is trained for 72 hours until convergence on a single Nvidia Titan GPU with 6 GB of RAM, using the unsupervised training procedure described in Section 3.4.2. The training sequence is split into mini-batches of length 40 scans (5 seconds). For each mini-batch, the network is shown 10 frames and trained to predict the next 10 frames, leading to two such sequences per mini-batch. This is chosen to cover the usual length of the occlusions in the scene and we expect it would need to be increased for longer-lasting occlusions.

Next, the classifier giving $p(c_t|h_t)$ is trained in just 2 hours on the semantic classification task using the 2 minute long labelled data set. Only a relatively small amount of data is necessary, due to fact that the network is already pre-trained for the tracking task and only fine-tuning is required to produce class labels. Because the data set is skewed and contains many more objects of a particular type, e.g. pedestrians, we use a weighting scheme, assigning class weights equal to the inverse of class frequency to mitigate this effect and put emphasis on the objective of correctly predicting every class equally.

The forward propagation of a single input through the network takes 15ms on an Nvidia Titan GPU and 83ms on a commodity laptop CPU. This is sufficient to enable real-time processing of the considered stream of laser data at 8Hz.

3.7.2.2 Comparison to Classical Methods

To compare the performance of the proposed end-to-end system to more traditional multi-stage pipelines, we evaluate the ability to predict the future movement of dynamic objects of the proposed framework against a recently proposed approach [99], based on model-free tracking of dynamic objects using a Kalman filter. This method accepts raw laser scans, performs data clustering and association, as well as velocity estimation of moving

objects. This information is then used to predict the positions of individual points in the future. We choose this approach due to its universality and its commensurate performance with state-of-the-art systems. We tuned the parameters of this method on the training set, then collected results from the test set, and converted the output into occupancy grids for comparison with the proposed method.

3.7.2.3 Evaluation

Two typical input sequences and their corresponding predicted network output are shown in Figure 3.19. The network is able to uncover the un-occluded scene including the space occupancy y_t and object labels c_t. Moreover, it is able to update the positions of dynamic objects through temporary occlusion, demonstrating that it has learned to track and recognise objects in the scene.

In what follows, we quantitatively evaluate the performance of the proposed system to justify these qualitative observations.

Occupancy Accuracy

As the ground-truth occupancy annotations y_t of the full un-occluded state are not available, we instead measure the accuracy of predicting the future occupancy y_{t+n} with respect to the visible part of the scene in z_{t+n}. This does not directly measure the true filtering accuracy of the network in occluded areas but serves as a proxy to the ability of the network to generalise to the unobserved regions, essential for both occlusion filtering and future prediction. This is the same principle as the one used to train the network in Section 3.4.2. For binary obstacle prediction, in Figure 3.21 we show the computed F1-scores when predicting 20 consecutive frames spanning 2.5 seconds, averaged over the test set, and compare to both the simple network architecture described in Section 3.6.1 (that we train in a similar manner), and to the state-of-the-art multi-stage pipeline approach of [99]. The prediction is accurate in near horizon and progressively decreases

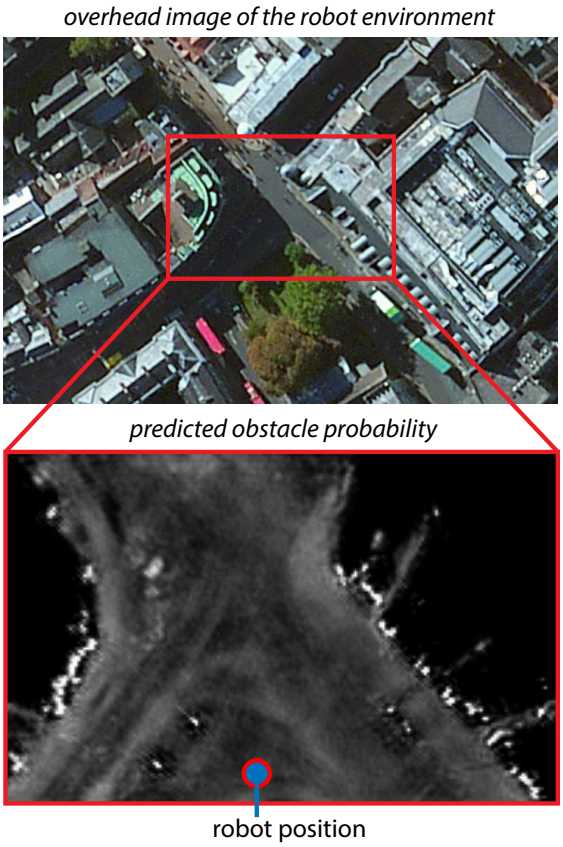


Figure 3.20: The aerial view of the robot environment **[top]** and corresponding network output when provided no input **[bottom]**. The network's ability to learn per-pixel information allows adaptation to the training environment. Thus, the network confidently predicts the position of static obstacles such as buildings, as well as the probability of any given cell being occupied even without any sensor input. Pavements show higher probabilities than the centre of the roads. For clarity, we show the log of the occupation probabilities.

over time. This is expected, as the uncertainty of the state of the world increases with the prediction horizon. Example of this behaviour is demonstrated in Figure 3.22. Our results outperform the two alternative approaches, demonstrating the effectiveness of the proposed network architecture and advantages of end-to-end learning.

Another experiment is concerned with measuring the effectiveness of the desired ability to learn place-specific information. One way to evaluate this is to visualise the network prediction y_1 without providing it with any input, as shown in Figure 3.20. Even without input sensor information, the network is able to provide an estimate of the expected oc-

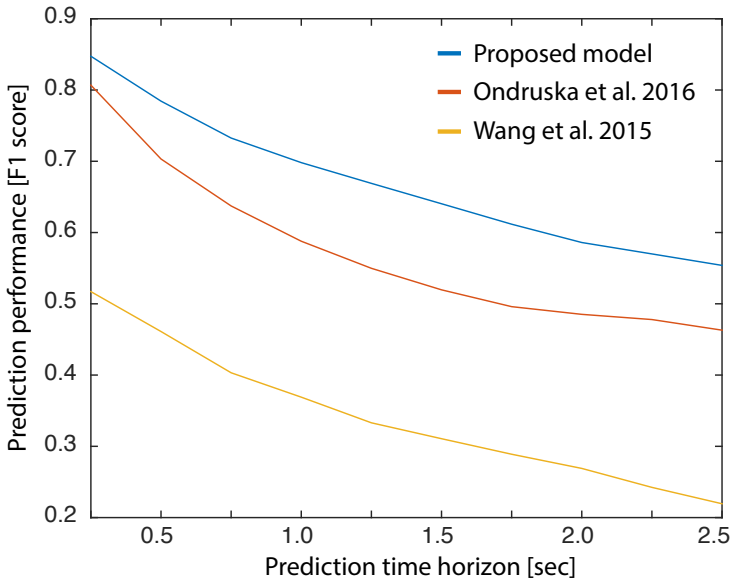


Figure 3.21: Network ability to correctly predict the future occupancy of the scene in the following 20 measurements equal to the time horizon of 2.5 seconds, measured by consistency with the future input. As the time horizon increases the quality of prediction degrades. The proposed complex neural network architecture performs better than the simple architecture as well as a state-of-the-art multi-stage pipeline approach [99].

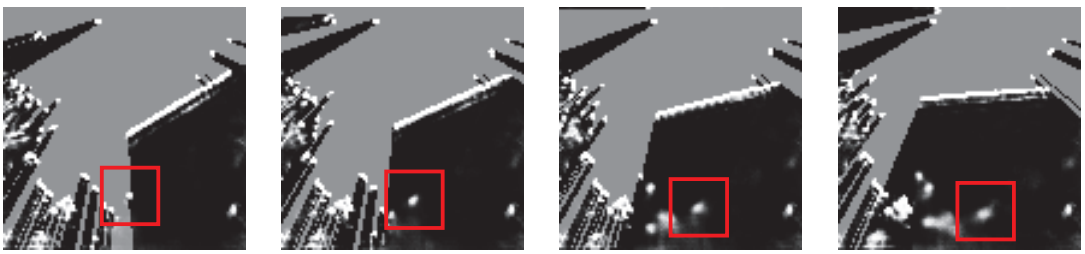


Figure 3.22: Example of system failure modes. As highlighted by the red square, uncertainty of the prediction is increasing with the prediction horizon in the case of lack of direct observations.

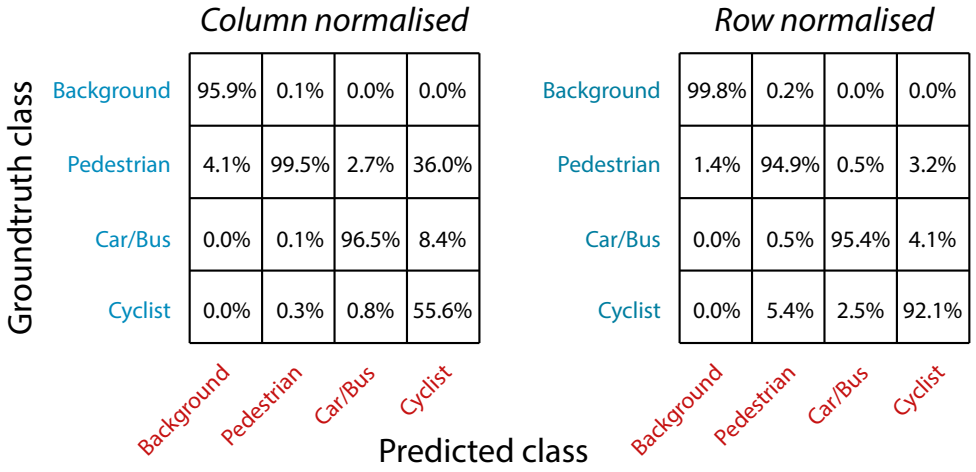


Figure 3.23: The confusion matrix of the semantic classification performance. Similarly to classical precision/recall metrics in binary classification, the column-normalised matrix provides insight into how often a particular type of object is classified into four classes in multi-class classification settings. Inversely, the row-normalised matrix provides insight into true identity of objects predicted to be of any particular class.

cupancy probabilities, which is higher at the locations of static obstacles and at crowded areas of the scene such as pavements. As no propagation of the information through the network occur, this is clearly only made possible by the ability of the network to remember this information in its static memory during training.

Semantic Accuracy

To quantify the network's ability to classify scene semantics, we compute the confusion matrix which is shown in Figure 3.23. As can be seen, the network is able to produce reliable classification for the object classes considered. The main source of error lies in distinguishing cyclists from pedestrians, as they often exhibit similar shapes in 2D laser data.

To verify the value of the proposed inductive transfer of knowledge, we compare this result to an alternative approach of classifying scene semantics, which takes the form of a one-shot classification of all directly visible obstacles from a single raw-sensor input. As a representative solution to such an approach, we train a three-layer deep convolu-

tional classifier predicting c_t directly from z_t. Despite conducting rigorous parameter tuning, directly classifying the input z_t yields inferior classification accuracy compared to classifying the hidden representation h_t, the negative log likelihood of correct labels being respectively $\sum_t -\log p(c_t|z_t) = \mathbf{101.967}$ and $\sum_t -\log p(c_t|h_t) = \mathbf{49.129}$. This demonstrates that h_t offers a powerful semantic descriptor of the scene and can be used as input for accurate semantic classification.

3.8 Conclusions

In this chapter we considered the problem of learning hidden representations of large state-spaces directly from the data. In particular, we presented a novel end-to-end trainable solution for real-time object tracking and classification in complex and partially-observable real-world environments. Leveraging the representational power of recurrent neural networks and employing efficient training procedures, the method surpasses a representative state-of-the-art model-free method, while substantially reducing the requirement for hand-engineered knowledge.

The method can be extended or applied in a number of ways. The universal schema of input and output opens up a possibility to apply the method to situations beyond those evaluated, such as accounting for robot motion, which can be handled by moving the robot inside the grid, or multi-sensor or multi-robot fusion. Additionally, the knowledge of the environment captured in learning to track can be further exploited to provide different kinds of semantic information such as bounding boxes. Finally, the inherent ability to predict the future evolution of the environment around the robot can be leveraged in more far-sighted planning scenarios.

Chapter 4

Learning and Evaluating Policy for Personalised Prediction

In the last chapter we covered the learning of appropriate representation of $\mathcal{S}$, $\mathcal{T}$ and O to model complex system using HMM. In this chapter we look into the problem of learning MDP rewards $\mathcal{R}$ and the effective evaluation of the resulting MDP policy π in a MDP containing millions of states. We present a new method for predicting effective drivable range of an electric vehicle. This method is based on learning user driving behaviour, affecting electric vehicle energy consumption and then exploiting this information in evaluating whether particular destination on the map is within drivability range. In the context of the Markov decision process from Chapter 2. We assume that user driver behaviour can be modelled as observable MDP policy π but optimising an unknown reward $\mathcal{R}$. We uncover these rewards and then deploy a new and effective graph-search algorithm to rapidly predict attainability for many destinations, leading to the generation of an intuitive drivability map.

The motivation for this problem is that according to recent market forecasts, the number of electric vehicles (EVs) on the roads worldwide is set to increase from ca. 150,000 in 2013 to over two million by 2020 [1]. The adoption of this technology is driven largely



Figure 4.1: Range map prediction installed in a real electric car showing destinations with estimated attainability probability at least 90%.

by environmental, economic and political factors. However, recent studies have shown that one of the primary impediments to such mass-market adoption is *range anxiety*, due to inaccurate in-situ estimates of available vehicle range [71].

Our aim is to address these shortcomings by providing the driver with a personalised range map, which exhaustively specifies attainability for *every* destination in a realistically sized map in real-time, without the burden on the driver to provide route or destination information (see Figure 4.8). Instead of such explicit user-interaction we propose a life-long learning system, which continuously adapts to driver-specific energy needs by learning a route preference model. Attainability estimates are derived by comparing predictions of likely energy consumption with the current state of charge of the EV battery. While our work leverages a commonly employed route preference model, it is the requirement for efficient consideration of all routes to every possible destination in the map, which sets our endeavour apart from the otherwise richly studied area of route prediction. In particular, we show that framing this problem in the context of sequential decision making provides a natural mechanism not only for the incorporation of driver-specific information but also allows for an efficient implementation suitable for real-time deployment.

In particular, we show that energy consumption in reaching a particular destination can be formulated as policy evaluation in a Markov decision process. This policy is learned and, over time, refined using *inverse reinforcement learning* to reflect the driver's generalised route preferences. Similarly, the vehicle energy consumption is learned from the data, taking into account the driver, the environment, on-board auxiliary systems and the vehicle battery system as potential sources of estimation noise. Specifically, we show how this approach can be efficiently employed at a large scale, providing a confidence level with which a particular driver will be able to reach a particular destination, given the current state of charge of the battery of an electric vehicle. This probability of attainability is simultaneously computed in real time for all destinations in a realistically sized map and presented in the form of a *probabilistic attainability map*.

Our approach is evaluated using a publicly available data set providing real trajectory data of 50 individuals spanning approximately 10,000 miles of travel. We show that by accounting for driver-specific route preferences, our system significantly reduces the relative error in energy prediction compared to more common, driver-agnostic heuristics such as shortest-path or shortest-time routes. The system was also deployed on an all-electric Nissan Leaf, as illustrated on Figure 4.1, and evaluated using data from over 140 miles of driving. The proposed system produces results commensurate with state-of-the-art approaches in terms of prediction accuracy. To the best of our knowledge, this is the first work aimed at efficiently providing personalised range maps by accounting for both a driver's generalised route preferences and particular driving style, based only on observing driving behaviour.

The rest of the chapter is organised as follows. We start with reviewing related work in Section 4.1. Then, in Section 4.2 we formalise the problem of estimating electric vehicle range and computing the electric vehicle range map in terms of policy learning and its evaluation. In Section 4.3 we address the problem of learning a driver's policy from demonstrations, and in Section 4.4 its effective evaluation. Sections 4.5 and 4.6 provide

options over modelling a driver's policy and energy evaluation measures, respectively. Experimental evaluation of our approach is provided in Section 4.7, with the implementation details of the entire system in Section 4.8. We conclude in Section 4.9.

The work described in this chapter directly contributed to the following conference publications:

- P. Ondruska and I. Posner, *"The Route Not Taken: Driver-Centric Estimation of Electric Vehicle Range,"* in Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS), Portsmouth, NH, USA, 2014.

- P. Ondruska and I. Posner, *"Probabilistic Attainability Maps: Efficiently Predicting Driver-Specific Electric Vehicle Range,"* in IEEE Intelligent Vehicles Symposium (IV), Dearborn, MI, USA, 2014.

patent filling

- UK Patent Application No. 1409620.0 Electric Vehicle Range

and partially to publication

- M. Wulfmeier, P. Ondruska, and I. Posner, *"Deep Inverse Reinforcement Learning,"*, NIPS Reinforcement Learning Workshop, Montreal, Canada, 2015

4.1 Related Works

Prediction of energy consumption is a richly studied area. Especially with the rise of electric and hybrid vehicles there has been a considerable effort to model and predict their energy consumption with the aim to better utilise their capabilities. One of the main goals is to estimate their driving range, the distance one can travel given the current state of batteries' charge.

The simplest methods for on-line prediction consider past several minutes of driving, and make predictions based on the assumption that the energy consumption rate won't

change much [72, 17]. The main limitation of these methods is that the energy consumption can vary rapidly in the future as the car travels along roads with different factors like traffic congestion, average speed or slope of the road. Moreover, these factors substantially differ based on what trajectory the user is going to follow. This means that the real attainable driving range is not a single number but very much depends on the intended destination and route taken. Predicting the intended route and modelling influencing factors along therefore lie in the core of accurate driving range prediction.

Several accurate models were developed to predict the energy consumption of the vehicle along user-specified trajectory with known influencing factors [49, 46].

Prediction of the intended destination and trajectory is a more challenging task. One method is to use anonymised GPS trajectory records of people traversing the area to predict expected trajectory in the near future [19, 33]. Although this method is powerful in some scenarios, it requires a good coverage of GPS trajectories from the past and can often provide only short-distance predictions.

In the case of the destination being known, a simple heuristic to estimate the trajectory assuming the user will take shortest or fastest route [37] is often used. This prediction is, however, often inaccurate in the real world because people tend to have more complex preferences like avoiding highways or preferring particular routes at different times of the day. In the energy context, this can account for a 10% difference in of energy requirement [51]. A particularly useful method leading to more accurate predictions is to use inverse reinforcement learning to uncover these preferences by observing past driving behaviour and using them as a cost function during planning [109].

Destination prediction itself is again a non-trivial task. One way to avoid it is to make energy prediction for every possible destination in the form of the drivable map of reachable destinations. This can provide very intuitive visual reference of whether the user's intended destination is attainable or not. One implementation of this idea uses the querying of an external route planning engine to estimate expected trajectories to nearby

destinations, and model the energy usage along them to construct a sparse drivable map [31]. The greatest disadvantage of this method is its computational speed, since every destination requires a separate query and therefore generation of the map can take a very long time. Another disadvantage is that the planning engine does not take into account individual user preferences.

Our aim is similar - to compute drivable maps. However, we aim to generate dense maps of high detail, taking into account individual user driving preferences at a speeds suitable for on-line applications. We achieve this by incorporating Ziebart's inverse reinforcement learning framework to learn the user preferences and then to effectively integrate energy consumption over distributions of possible paths to every possible destination directly into the planning step.

Learning user route preferences and making planning-based predictions in context of energy is a relatively new combination. From this point of view our work is closely related to Vogel et. al [98], who also used IRL to learn preferences to optimise the powertrain controller of a hybrid car, claiming fuel saving of 1.22% over a traditional approach. In contrast to this work, the prediction of drivable range is a very different problem, setting a new set of challenges. Most importantly, we need to take into account user behaviour to reach every possible destination and not only the behaviour in the near future.

4.2 Electric Vehicle Range Prediction

In this section we describe how the problem of predicting electric vehicle range can be framed in terms of Markov decision process policy learning, and evaluation as described in Chapter 2. The computation consists of two parts. We first describe a formal way to compute required energy $E_{s \to g}$ to reach particular destination, and then extend the approach to confidence computations reaching all possible destinations presented in the form of an attainability map.

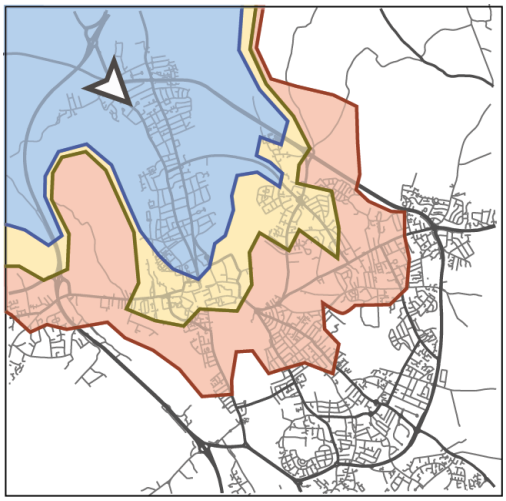


Figure 4.2: Attainable range with 90% (blue), 50% (yellow) and 10% (red) probability.

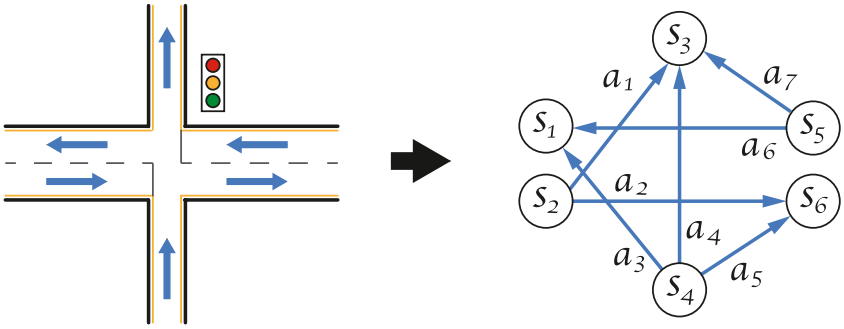


Figure 4.3: The state-action space of the road network. States correspond to oriented road segments, and actions correspond to possible turns at the end of a road segment.

4.2.1 Energy Prediction as MDP

Consider an electric vehicle which is routinely deployed in a given area for which a map is available. Specifically, we consider a map to consist of a road network composed of individual road segments $\mathcal{S} = \{s_1, s_2, \ldots, s_N\}$ which are joined at intersections, and a set of actions a_i corresponding to possible turns at the end of each road segment. The transition model $\mathcal{T}$ in this case is deterministic and encodes the connectivity graph. The deterministic nature of the transition model implies that a particular state-action pair leads to a particular next state with absolute certainty, such that $p(s'|s, a) = 1$. In fact, the specific MDP we consider here encompasses all *oriented* road segments and associated

actions as illustrated in Figure 4.3. Such a road network together with associated route infrastructure (e.g. number and location of traffic lights, stop signs, etc) can be readily obtained from community projects such as OpenStreetMap [40].

Now assume the car is located at state s_s and the driver plans to reach the destination s_g. He or she chooses one from a number of possible trajectories $\Lambda_{s\to g}$. Each such trajectory $\varsigma \in \Lambda_{s\to g}$ can then be expressed as a sequence of state-action pairs $\varsigma = \{s_s, a_0, s_1, a_1, ..., s_{n-1}, a_{n-1}, s_g\}$. Importantly, the exact trajectory the driver chooses to reach s_g is not known to the system beforehand, but the choice of likely trajectories is modelled by a stochastic policy π, defining a probability distribution over possible actions $p_\pi(a_j|s_i)$ at the end of every road segment. The probability that the user takes a particular trajectory can be expressed in terms of the probability of observing the corresponding state-action sequence, such that

$$p_\pi(\varsigma) = \prod_{\{s_t, a_t, s_{t+1}\}\in\varsigma} p(s_{t+1}|s_t, a_t) p_\pi(a_t|s_t). \tag{4.1}$$

As shown in Figure 4.4, traversing any trajectory costs energy E_ς, which is decomposed into energies spent on individual road segments

$$E_\varsigma = \sum_{\{s,a\}\in\varsigma} E_{s,a}. \tag{4.2}$$

Computing the energy $E_{s\to g}$ required to reach destination s_g is now equivalent to evaluating the value of policy π

$$E_{s\to g} = \sum_{\varsigma} p_\pi(\varsigma) E_\varsigma. \tag{4.3}$$

Importantly, note that the driver's policy is not assumed to minimise the energy $E_{s\to g}$, but as discussed in Section 4.3 it captures actual routing preferences, and $E_{s\to g}$ is the required energy to reach the destination given these preferences.

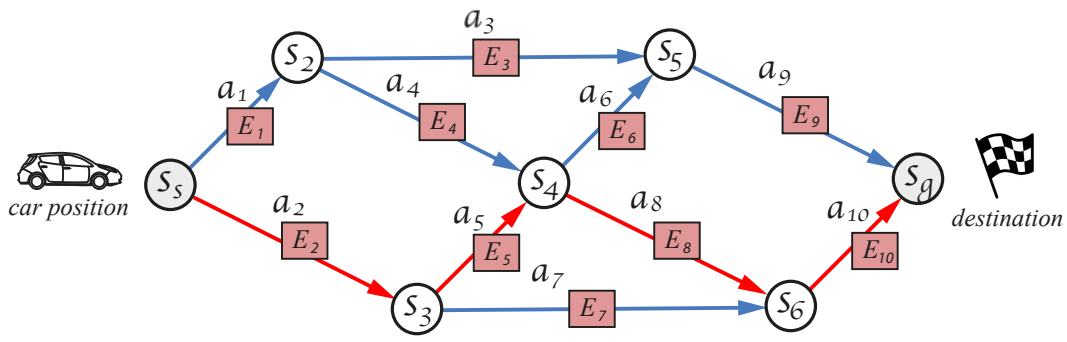


Figure 4.4: The road network represented as states of the MDP with the car at state s_s and intended destination s_g. Each trajectory in the road network corresponds to a trajectory in the MDP with its energy cost corresponding to the sum of energies spent on individual segments $E_{s,a}$.

4.2.2 Attainability Map

The previous section described a general framework for computing required energy $E_{s\to g}$ to reach a single destination in the map. In this section we use this formulation to describe an attainability map, which captures the confidence the driver is able to reach *every* destination on the map. In particular, we estimate a probability p_i that energy $E_{s\to i}$ required to travel to destination s_i is at most the available energy E_{soc}

$$p_i = p(E_{soc} \geq E_{s\to i}), \tag{4.4}$$

and consider this destination attainable if this probability is greater or equal to a user-specified confidence threshold p_t

$$\Gamma_i = \begin{cases} 1 & \text{if } p_i \geq p_t \\ 0 & \text{otherwise.} \end{cases}. \tag{4.5}$$

An example of an attainability map for different values of p_t is shown in Figure 4.2. Equivalently to (4.4), the attainability p_i of a single destination s_i can be expressed in

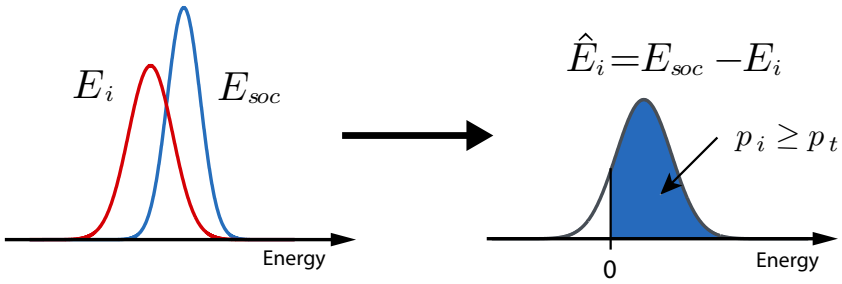


Figure 4.5: A statistical interpretation of destination attainability p_i given the distributions of required energy $E_{s \to i}$ and available energy E_{soc}.

terms of a transformed random variable $\hat{E}_i = E_{soc} - E_{s \to i}$ (see Figure 4.5) such that

$$p_i = p(\hat{E}_i \geq 0), \tag{4.6}$$

which leads to an intuitive formulation in terms of the cumulative distribution function of $\hat{E}_i$, $P_{\hat{E}_i}(x)$, as

$$p_i = 1 - P_{\hat{E}_i}(0). \tag{4.7}$$

If we approximate that the original quantities involved are normally distributed (i.e. $E_{s \to i} \sim \mathcal{N}(\mu_{s \to i}, \sigma_{s \to i})$ and $E_{soc} \sim \mathcal{N}(\mu_{soc}, \sigma_{soc})$), it follows that $\hat{E}_i$ is also normally distributed such that

$$\hat{E}_i \sim \mathcal{N}(\hat{\mu}_i, \hat{\sigma}_i), \tag{4.8}$$

$$\hat{\mu}_i = \mu_{soc} - \mu_{s \to i}, \tag{4.9}$$

$$\hat{\sigma}_i^2 = \sigma_{soc}^2 + \sigma_{s \to i}^2. \tag{4.10}$$

(4.7) can then be expressed in terms of the cumulative distribution function of the standard



Figure 4.6: Examples of most likely trajectories between the same start and destination induced by three different route preferences. [**left**] minimum time travelled (the trajectory mainly follows motorways), [**middle**] minimum distance travelled and [**right**] custom preference learned from observations.

normal distribution Φ, such that

$$p_i = 1 - \Phi\left(-\frac{\hat{\mu}_i}{\hat{\sigma}_i}\right). \tag{4.11}$$

Note that this formulation allows for the state of the charge estimate to be a random variable with associated measurement noise, which can be considerable [18]. While it is conceivable that these parameters are also learnt as part of this model, in the following we will use typical values for μ_{soc} and σ_{soc} as discovered in prior art.

In this context, the challenge of estimating the chance of reaching a particular destination with a given state of charge reduces to first estimating the driver's policy π, and then efficiently calculating quantities $\mu_{s\to i}$ and $\sigma_{s\to i}$ for all destinations s_i. These are discussed in the following sections.

4.3 Learning Driver's Policy

A driver's route preferences can considerably skew the energy required to reach a particular destination [65, 51], Figure 4.6. To account for this variation we learn personalised route preferences directly from observed driving behaviour. In modelling the driver's

behaviour we take inspiration from [109] and consider the probability of traversing a particular trajectory to be characterised by a reward, $\mathcal{R}_\varsigma$, capturing this preference such that

$$p_\pi(\varsigma) = \frac{1}{Z} e^{\mathcal{R}_\varsigma}, \tag{4.12}$$

where the partition function Z is defined as

$$Z = \sum_{\varsigma \in \Lambda_{s \to g}} e^{\mathcal{R}_\varsigma}. \tag{4.13}$$

The reward for the trajectory is derived from rewards of individual state-action pairs, $\{s, a\}$, along the trajectory, Figure 4.7

$$\mathcal{R}_\varsigma = \sum_{\{s,a\} \in \varsigma} \mathcal{R}_{s,a}. \tag{4.14}$$

We model this reward to be a function of road features $f_{s,a}$ governed by parameters w_ρ,

$$\mathcal{R}_{s,a} = F_{w_\rho}(f_{s,a}). \tag{4.15}$$

Later, in the Section 4.5, we describe different choices for F: a fixed model or a linear model.

The features f considered in this work are extracted from OpenStreetMap [40], and consist of the road class (highway, 1st, 2nd, 3rd class, residential, service, etc.), the segment length, the amount of turn at the end of the segment (straight, turn left, turn right), whether a stop is likely at the end of the segment (e.g. due to road priorities or the presence of a traffic light or stop sign) as well as the total elevation increase and decrease along the segment obtained from Google Maps Elevation Service.

As shown in [109], the policy π expressed in the form of (4.12) has an equivalent representation in the form of a probability over taken actions at a given state $p(a|s)$ ((4.1)).

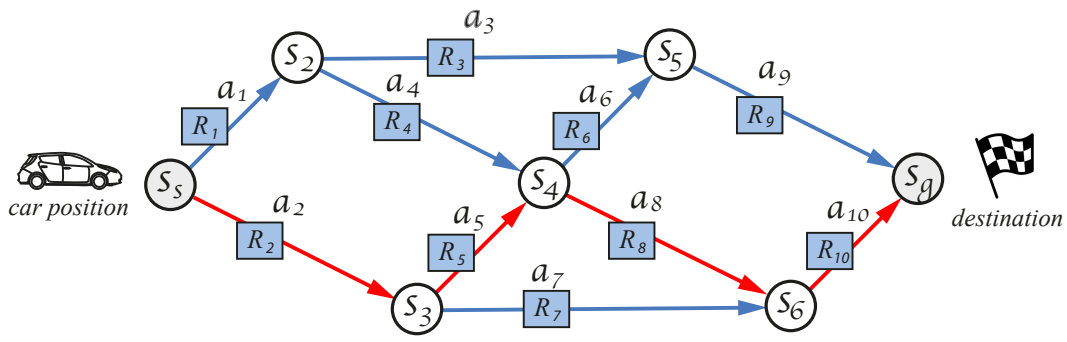


Figure 4.7: The road network represented as states of MDP with car at state s_s and intended destination s_g. A driver's preference over traversing each trajectory is modelled by individual segment rewards $\mathcal{R}_{s,a}$.

Algorithm 3 Computation of policy π from rewards $\mathcal{R}$.

Input:	$\mathcal{R}_{s,a}$	*reward structure*
	$p(s'\|s,a)$	*transition model*
	s_g	*destination*
Output:	$p_\pi(a\|s)$	*policy*

1. $Z_i^0 = 1$
2. **for** $k = 1, 2, 3, .., N$ **do**
3. $\quad Z_{s,a}^k = \sum_{s'} p'(s'|s,a) e^{\mathcal{R}_{s,a}} Z_{s'}^{k-1}$
4. $\quad Z_i^k = \sum_a Z_{s,a}^k + 1_{\{s=s_s\}}$
5. $p_\pi(a|s) = \frac{Z_{s,a}^N}{Z_i^N}$

This probability is equivalent to the cumulative probability of all trajectories $\Lambda_{s\to s_g}^a$, starting with that action:

$$p_\pi(a|s) = \sum_{\varsigma \in \Lambda_{s\to sg}^a} p_\pi(\varsigma). \tag{4.16}$$

The computation of this representation can be carried out by an effective dynamic programming algorithm described in [109] and summarised in Algorithm 3. This representation is important as it allows fast and effective policy evaluation, discussed later.

The learning of the reward structure for particular driver has the following form. Given the features, structure of the road network f and a set of observed driver trajectories $D = \{\varsigma_1, \varsigma_2, ..., \varsigma_n\}$ between various $\{s_s, s_g\}$ pairs we seek to optimise parameters w_ρ

governing the reward structure. Following the Chapter 2 our aim is to minimise the negative log likelihood of observing these trajectories under the policy considered in (4.12),

$$\begin{aligned} \mathcal{L}(w_\rho) &= -\log p(D, \pi) && (4.17) \\ &= -\sum_{\varsigma \in D} \log p_\pi(\varsigma) - \log p(w_\rho). && (4.18) \end{aligned}$$

To learn optimal w_ρ we use a gradient descent to minimise $\mathcal{L}(w_\rho)$, with gradients given as

$$\frac{\partial \mathcal{L}(w_p)}{\partial w_\rho} = -\sum_{\varsigma \in D} \frac{\partial \log p_\pi(\varsigma)}{\partial \mathcal{R}} \cdot \frac{\partial \mathcal{R}}{w_\rho} - \frac{\partial \log p(w_\rho)}{\partial w_\rho}. \tag{4.19}$$

The derivative of the trajectory probability is given as a difference between expected and observed reward,

$$\frac{\partial \log p_\pi(\varsigma)}{\partial \mathcal{R}} = \mathcal{R}_\varsigma - \mathcal{R}_{s \to g}, \tag{4.20}$$

where the expected reward $\mathcal{R}_{s \to g}$ under policy π takes the form of (2.9),

$$\mathcal{R}_{s \to g} = \sum_{\varsigma \in \Lambda_{s \to g}} p_\pi(\varsigma) \mathcal{R}_\varsigma \tag{4.21}$$

Its computation is therefore similar to evaluating expected energy consumption $E_{s \to g}$ and will be discussed in the next section. The gradient of the rewards with respect to weights,

$$\frac{\partial \mathcal{R}}{\partial w_\rho} = \frac{\partial}{\partial w_\rho} F_{w_\rho}(f), \tag{4.22}$$

depends on the choice of F and will be discussed in Section 4.5. Finally, to model the regularisation term $p(w_\rho)$ we use a standard L_2 norm (see Chapter 2), expressing our belief of zero-mean normally-distributed weights with gradient given by

$$\frac{\partial \log p(w_\rho)}{\partial w_\rho} = \lambda w_\rho. \tag{4.23}$$



Figure 4.8: *Range maps* personalised for particular drivers as typically provided by our system. The attainability of every possible destination is indicated (shaded area) for the same vehicle location (triangle) and battery state of charge. Our work specifically accounts for the variation of these maps induced by a driver's route preferences by continuously observing trajectories travelled by a specific user.

4.4 Evaluating Driver's Policy

Here we outline the framework used to model energy consumption on route to a particular destination $E_{s\to g}$. We consider this quantity not to be a constant but a random variable and we aim to not only to compute its mean expected value $\mu_{s\to g}$ but also to estimate its variance $\sigma_{s\to g}$ in order to estimate drivability uncertainty. Moreover, we want to compute these quantities for all possible destinations s_g, to allow us to compute a very dense drivability map.

As in (4.3), we consider the energy required to reach a particular destination s_g to be computed with respect to a distribution over possible trajectories to that destination. Similarly, we consider E_ς to be a random variable with distribution parameters derived from real data. $E_{s\to g}$ is therefore a mixture of distributions describing the energy consumed along possible trajectories, E_ς, with the mixing coefficients provided by the probability of a driver taking that particular trajectory, as shown in Figure 4.9. The required mean

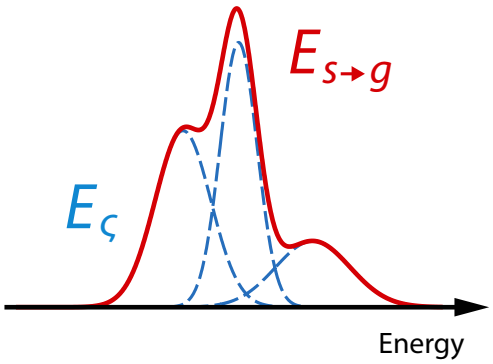


Figure 4.9: The distribution of required energy to given destination $E_{s\to s_g}$ as a mixture of distributions over possible trajectories E_ς.

and variance of $E_{s\to g}$ are given by

$$\mu_{s\to g} = \sum_{\varsigma\in\Lambda_{s\to g}} p_\pi(\varsigma)\mu_\varsigma, \text{ and} \tag{4.24}$$

$$\sigma^2_{s\to g} = \mathbb{E}E^2_{s\to g} - (\mathbb{E}E_{s\to g})^2 \tag{4.25}$$

$$= \sum_{\varsigma\in\Lambda_{s\to g}} p_\pi(\varsigma)\left(\mu^2_\varsigma + \sigma^2_\varsigma\right) - \mu^2_{s\to g}. \tag{4.26}$$

While intuitive, this approach to range map computation suffers from two significant drawbacks. Firstly, the set $\Lambda_{s\to g}$ can consist of exponentially many trajectories, which renders a direct computation of $E_{s\to g}$ infeasible in a real-time context. Secondly, this problem is compounded by the need to compute $E_{s\to g}$ for every possible destination s_g in the map. In the remainder of this paper we set out a computationally efficient solution to the task of computing such a driver-specific range map.

4.4.1 Computing the Mean

Here we outline computation of the mean value of $E_{s\to g}$ to be equivalent to evaluating driver's policy π. (4.2) gives rise to a simple equation for mean expected energy expressed

as a sum over individual road segments,

$$\mu_{\varsigma} = \sum_{\{s,a\} \in \varsigma} \mu_{s,a}. \tag{4.27}$$

Using (4.1), the closed form of policy π in the format of (4.24) for the mean energy is given by

$$\mu_{s \to g} = \sum_{\varsigma} \left(\prod_{\{s,a\} \in \varsigma} p_{\pi}(a|s) \sum_{\{s,a\} \in \varsigma} \mu_{s,a} \right), \tag{4.28}$$

which is equivalent to computing the *value* of the policy π with respect to $\mu_{s,a}$. This formulation is important as it allows us to employ an efficient algorithm for policy evaluation from Section 2.1.5. Here we employ a method where $\mu_{s \to g}$ can be found as a solution of a system of linear-equations:

$$\mu_{s \to g} = \sum_{a} p_{\pi}(a|s) \left(\mu_{s,a} + \sum_{s'} p(s'|s,a) \mu_{s' \to g} \right).$$

Moreover, this method has the advantage that it produces value $\mu_{s \to g}$ and hence $\mu_{i \to g}$ for all possible starting states s_i at once. Computation of a range map, however, requires the opposite, $E_{s \to i}$ for all possible destinations s_i, an expected energy to reach every destination. In a later section we show how our formulation and usage of the specific policy class we considered in the previous section allows for computation of this quantity feasible at no extra cost.

4.4.2 Computing Variance

To make computing the variance of $E_{s \to g}$ tractable, we make an assumption that energy expenditures on individual road segments $E_{s,a}$ are independent from each other. (4.2)

then implies

$$\sigma_{\varsigma}^2 = \sum_{\{s,a\} \in \varsigma} \sigma_{s,a}^2. \tag{4.29}$$

By substituting

$$\vec{\sigma}_{s,a}^2 = \sigma_{s,a}^2 + \left(\mu_{s,a} + \mu_{s \to g} - \sum_{s'} p(s'|s,a) \mu_{s' \to g} \right)^2, \tag{4.30}$$

the values $\sigma_{i \to g}$ can be found as a solution of the following linear system

$$\sigma_{s \to g}^2 \;\; = \;\; \sum_{a} p_{\pi}(a|s) \left(\vec{\sigma}_{s,a}^2 + \sum_{s}^{'} p(s'|s,a) \sigma_{s' \to g}^2 \right). \tag{4.31}$$

Thus, we can use the same algorithm used to solve the system in (4.29).

4.4.3 Predicting All Destinations

As described above, policy evaluation can be used to compute the expected energy requirement $E_{s \to g}$ in reaching a destination s_g from the current location s_s. The generation of a range map, however, in principle requires the application of such a policy evaluation step for every possible destination, which is computationally infeasible. Instead, we present a computational shortcut, which allows for the simultaneous evaluation of $E_{s \to i}$ for every possible goal s_i in a map.

In particular, instead of the MDPs described in Section 4.2 we now consider related MDPs consisting of the same set of states, actions and rewards but with a transition model $\mathcal{T}'$ which encodes the *transpose* of the original road graph. Effectively, the outcome of every action is reversed, such that $p'(s'|s,a) = p(s|s',a)$. Note that every trajectory ς in the original MDPs is also feasible in the new MDPs, but that it is traversed in reverse (see Figure 4.10). Importantly, we keep the same energy requirements and rewards. For example, an uphill one-way road in the original MDP is a downhill road in the transposed MDP but it consumes the same amount of energy as driving uphill. As the reward struc-

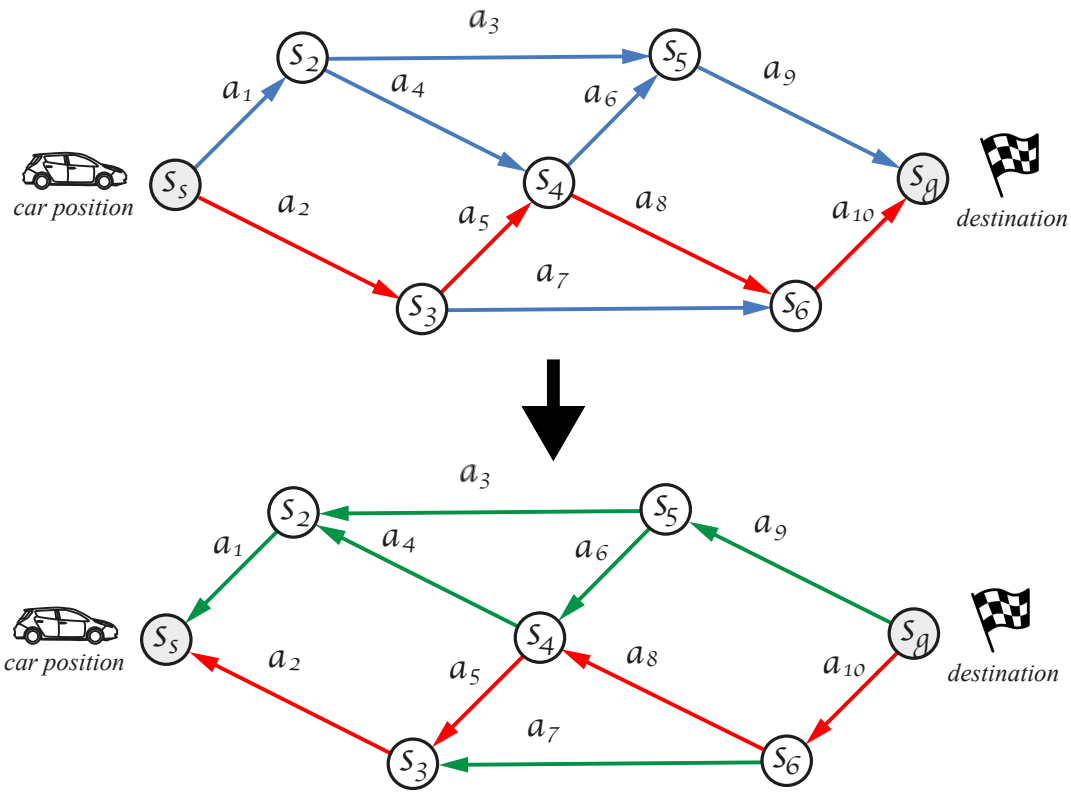


Figure 4.10: A trajectory spanning states and actions in the original road network **[top]** and the same trajectory in the transposed network **[bottom]**.

ture remains the same, (4.12) implies that the probability distribution over trajectories $\Lambda_{s\to g}$ from s_s to s_g in the original MDPs is the same as the probability distribution over trajectories $\Lambda_{s\leftarrow g}$ from s_g to s_s in the new MDPs. It follows, therefore, that $E_{s\to g}$ in the original road network and $E_{g\leftarrow s}$ in the transposed version are equivalent,

$$\begin{aligned} E_{s\to g} &= \sum_{\varsigma\in\Lambda_{s\to g}} p_\pi(\varsigma)E_\varsigma \\ &= \sum_{\varsigma\in\Lambda_{s\leftarrow g}} p_{\pi'}(\varsigma)E'_\varsigma = E_{g\leftarrow s}. \end{aligned} \tag{4.32}$$

As a result, $E_{s\to i}$ can be computed for every s_i at once by first constructing MDPs based on the transposed road network and then solving for the expected energy consumption $E_{i\leftarrow s}$ from every starting state s_i to the destination s_s as described previously.

The method is summarised in Algorithm 4. The computation of $\mu_{s\to i}, \sigma^2_{s\to i}$ consists of three steps. First, we transpose the road network. Then we compute a policy to reach state s_s based on a particular driver's route preferences in the transposed route network using Algorithm 3. The value of $\mu_{s\to i}, \sigma_{s\to i}$ for every state is then computed by solving a system of linear equations. The final range map is obtained by thresholding the confidence in $E_{s\to i}$ being less than E_{soc}, the current state of charge of the battery.

Algorithm 4 Efficient Range Map Computation.

Input:	s_s	*position of the car*
	$\mu_{soc}, \sigma^2_{soc}$	*battery charge*
	$\mu_{s,a}, \sigma^2_{s,a}$	*segment energy consumption*
	$\mathcal{R}_{s,a}$	*driver routing preferences*
	p_t	*confidence threshold*
Output:	Γ_i	*destinations attainability*

Transpose the road network.

1. $p'(s_k|s_i, a_j) = p(s_i|s_k, a_j)$

Compute transposed driver policy π' to reach state s_s using Algorithm 3.

2. $p_{\pi'}(a|s) = ComputePolicy(p'(s'|s,a), \mathcal{R}'_{s,a}, s_s)$

Solve system of linear equations for $\mu_{s\to i}$.

3. $\mu_{s\to i} = \sum_a p_{\pi'}(a|s)\Big(\mu_{s,a} + \sum_{s'} p'(s'|s,a)\mu_{s\to s'}\Big)$

Solve system of linear equations for $\sigma^2_{s\to i}$.

4. $\vec{\sigma}^2_{s,a} = \sigma^2_{s,a} + \left(\mu_{s,a} + \mu_{s\to g} - \sum_{s'} p(s'|s,a)\mu_{s'\to g}\right)^2$
5. $\sigma^2_{s\to i} = \sum_a p_{\pi'}(a|s)\Big(\vec{\sigma}^2_{s,a} + \sum_{s'} p'(s'|s,a)\sigma^2_{s\to s'}\Big)$

Compute resulting drivability map.

6. $\hat{\mu}_i = \mu_{soc} - \mu_i$
7. $\hat{\sigma}^2_i = \sigma^2_{soc} + \sigma^2_i$
8. $p_i = 1 - \Phi\left(-\frac{\hat{\mu}_i}{\hat{\sigma}_i}\right)$
9. $\Gamma_i = \begin{cases} 1 & \text{if } p_i \geq p_t \\ 0 & \text{otherwise} \end{cases}$

4.5 Reward Models

In this section we summarise two different approaches to modelling reward $\mathcal{R}_{s,a}$ that the driver is assumed to be optimising and which explain the choice of the route he or she takes.

4.5.1 Fixed Reward

The most basic assumption of the driver's behaviour is the one where the driver is assumed to follow the shortest or the fastest path to the destination. In this case the reward is directly proportional to the length of the segment,

$$\mathcal{R}_{s,a} = -\alpha \cdot l_s \tag{4.33}$$

if the shortest-path preference is assumed and

$$\mathcal{R}_{s,a} = -\alpha \cdot t_s \tag{4.34}$$

if the min-distance preference is assumed. α is a large constant describing how strongly this assumption is followed. The length of the road segment l_s is easily obtained from the road map information and the time to traverse road segment t_s can be computed using the method described in Section 4.6.1.

4.5.2 Linear Model

The linear model represents the driver's routing preferences in [109] and uses the linear regression from Section 2.2 to model reward,

$$\mathcal{R}_{s,a} = w_\rho^\top \cdot f_{s,a}. \tag{4.35}$$

The advantage of this model is its simplicity, and as shown in [109], it allows a direct interpretation of different factors such as the class of the road or driver routing preferences.

4.6 Energy Cost Models

Similarly to reward models we investigate three different approaches to model driver energy consumption $E_{s,a}$.

4.6.1 Hand-Engineered Model

First we consider a fixed model, where only $\mu_{s,a}$ is modelled. The energy consumed over the course of a trajectory, E_ς, is modelled as the sum of energy costs of state-action pairs along the trajectory and given by (4.2). This cost incorporates events specific to the transition, such as potentially stopping at an intersection or changing velocity according to the law of the land, as well as the energy required while traversing road segment s_{t+1} itself. To estimate these values we combine a canonical model of an expected velocity profile with a physical model of the resulting energy demand of the vehicle. We model a driver's chosen velocity v, and acceleration $\dot{v}$ as a road segment is traversed by adapting the Intelligent Driver Model (IDM) by [94]. The IDM was originally applied in the context of a car-following scenario. However, it can be employed here by assuming the end of the

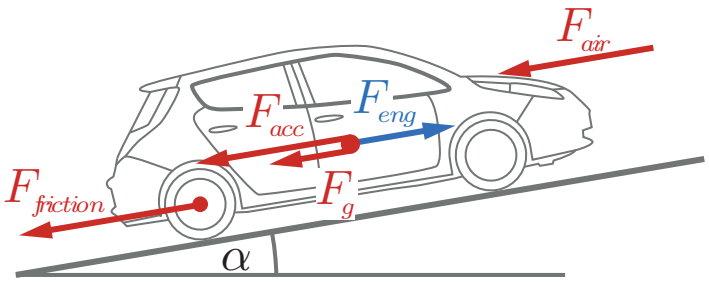


Figure 4.11: The forces acting on a car [39].

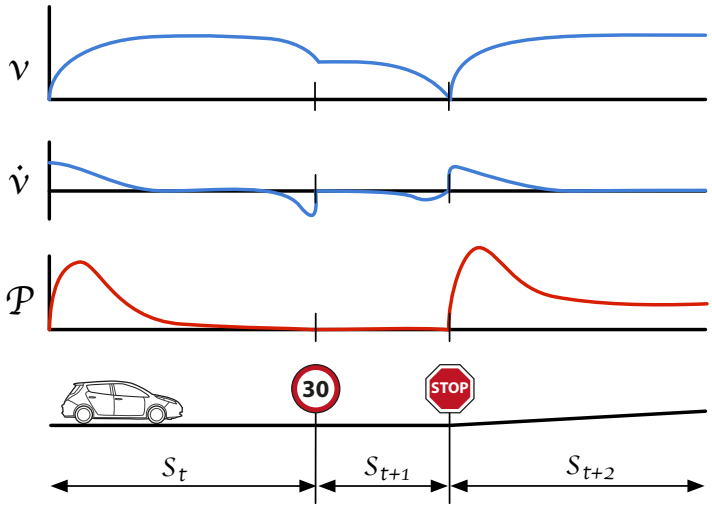


Figure 4.12: Velocity, acceleration and power demand profile for three adjacent road segments containing a speed limit, a stop sign and a slight elevation.

road segment to be a car moving at velocity v_t, equivalent to the desired velocity at the end of the segment. Given the road speed limit v_0 and distance to the end of the segment s, the acceleration of the driver is then given by the ordinary differential equation

$$\dot{v} = \dot{v}_{max}\left(1 - \left(\frac{v}{v_0}\right)^{\delta}\right) + \dot{v}_{min}\left(\frac{(v - v_t)^2}{4s^2}\right). \tag{4.36}$$

Values of speed limits, positions of stop signs and traffic lights can be extracted from sources such as OpenStreetMap [40]. δ denotes a smoothness parameter. The values used are summarised in Table 4.1.

The energy cost of an action $E_{s,a}$ is obtained by integrating the vehicle's power demand P_{car} during action a,

$$E_{s,a} = \int P_{car}(t)dt. \tag{4.37}$$

The power demand of the car is the sum of the engine power P_{eng} and the auxiliary power demand P_{aux} due to car accessories such as lights, air conditioning, etc, such that

$$P_{car} = P_{eng} + P_{aux}. \tag{4.38}$$

The engine power demand P_{eng} is a function of velocity and acceleration. We assume an efficient system such that

$$P_{eng} = F_{eng} \cdot v, \tag{4.39}$$

where F_{eng} is the force produced by the engine required to overcome forces acting on the vehicle at a given speed. As depicted in Figure 4.11, these forces decompose into components due to acceleration, friction, air resistance and gravitation [39],

$$F_{eng} = F_{acc} + F_{friction} + F_{air} + F_g, \tag{4.40}$$

where

- $F_{acc} = m \cdot \dot{v}$ is the force needed to accelerate the vehicle,
- $F_{friction} = c_{rr} \cdot m \cdot g$ is the rolling resistance,
- $F_{air} = \frac{1}{2} c_d \cdot A_f \cdot \rho \cdot v^2$ is the aerodynamic resistance force,
- $F_g = m \cdot g \cdot sin(\alpha)$ is the hill-climbing force.

The physical constants required to compute these quantities are inspired by those for a Nissan Leaf and are detailed in Table 4.1. Figure 4.12 shows an example of the resulting acceleration, velocity and power demand profile of a vehicle traversing a typical segment of the road network.

4.6.2 Linear Model

As our choice of a simple distribution of segment energy $E_{s,a}$ we considered a normal distribution, a commonly used distribution for modelling continuous variables, with certain mean and variance

$$E_{s,a} \sim \mathcal{N}(\mu_{s,a}, \sigma_{s,a}) \tag{4.41}$$

Constant	Description	Value
$\dot{v}_{max}$	maximum acceleration	$1m/s^2$
$\dot{v}_{min}$	comfortable deceleration	$-3m/s^2$
δ	smoothness coefficient	4
m	vehicle mass	$1521kg$
c_{rr}	rolling friction coefficient	0.015
c_d	aerodynamic drag coefficient	0.25
A_f	frontal area	$2.846m^2$
ρ	air density	$1.22kg/m^3$
P_{aux}	auxiliary power consumption	$490W$

Table 4.1: Physical quantities and values used to model energy consumption.

The parameters $\mu_{s,a}, \sigma_{s,a}$ can be modelled using a feature-based linear model similar to the linear model employed to model segment rewards,

$$\mu_{s,a} = w_\mu^\top \cdot f_{s,a}, \tag{4.42}$$

$$\sigma_{s,a}^2 = w_\sigma^\top \cdot f_{s,a}, \tag{4.43}$$

where $f_{s,a}$ again denotes a feature vector associated with every state-action pair and w_μ and w_σ denote driver specific weights parameterising the energy consumption. The weights are learned by maximising the likelihood of observed energy consumption (4.41) over the lifetime of the system by constrained optimisation subject to $w_\sigma^\top \cdot f_{s,a} \geq 0 \; \forall s, a$ [15]. To accommodate for different energy demands caused by different accessory settings and selecting ECO mode[1], we learn the weights for specific settings of variables thus we obtain different set of weights for different settings of the car. During prediction the appropriate weights are selected to reflect the current car settings.

[1]ECO mode reduces car acceleration, energy provided to the air conditioning system and increases regenerative braking in order to extend the drivable range.

4.7 Experiments

This section explores the efficacy of the method proposed and provides benchmarking with respect to alternative approaches. In particular, we set out to demonstrate that integration of driver-specific route preference information provides a significant performance gain in terms of reduced prediction error as compared to more standard, driver agnostic methods assuming, for example, shortest path or traversal time models.

4.7.1 Routing Prediction Accuracy

First, we evaluated the impact of our ability to predict driver's route on the accuracy of the predicted drivability range. This was done in a quantitative manner considering a large anonymised dataset of drivers trajectories and using simulated vehicle consumption model. We empirically observed the ability to correctly predict drivers route results into 30% reduction in estimation error compared to using classical methods.

4.7.1.1 Data

For evaluation we use Microsoft's GeoLife dataset (see, for example, [106, 108, 107]), which contains GPS traces collected from a number of different users. Specifically, our experiments consider 50 different users, each having on average 100 trajectories spanning the urban area of Beijing, China. The road network and infrastructure information for this region covering about $100km^2$ are obtained from the OpenStreetMap project [40]. This road network is very dense and contains a multitude of possible routes between any two places. The resulting MDP contains 80,000 states and 130,000 actions. As the dataset contains GPS traces only, these are segmented into individual trajectories based on time and position information. Next, we use the hidden Markov model-based method described in [69] to match the GPS trajectories to the traversed road segments. Following this preprocessing step, for every user we consider 30 randomly sampled trajectories from the

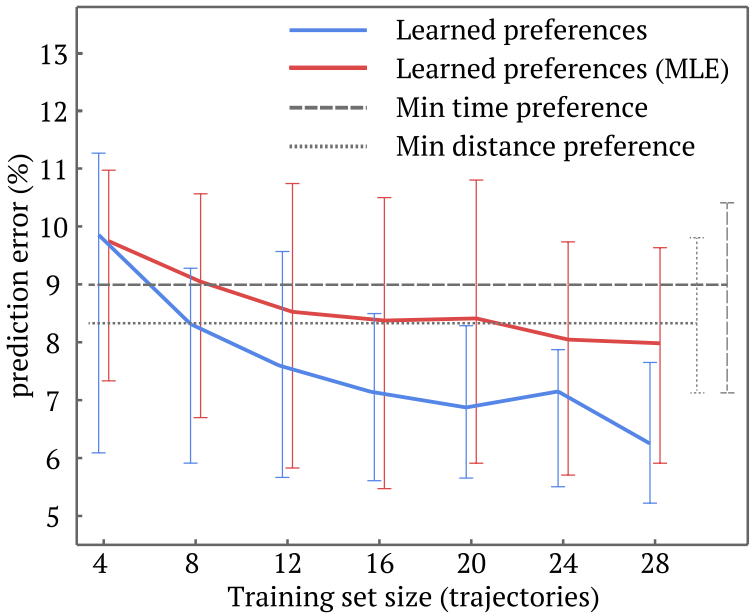


Figure 4.13: Mean predictive error with 25% and 75% qunatiles as the number of training trajectories increases for a variety of models. The approach proposed considers preference learning on a per-driver basis and accounts for the entire distribution over potential trajectories. It achieves the lowest overall prediction error compared to the maximum-likelihood trajectory estimate as well as more common modelling assumptions such as preferences for minimum travel time or distance.

entire set. The remainder are used for testing. Across all users this results in total training and test set sizes of 1500 and 3500 trajectories, respectively.

4.7.1.2 Evaluation

For each driver, a subset of the training set trajectories are used to learn the preference weights w. The prediction accuracy of the learned model is then evaluated using all trajectories in the test set. In particular, for every $\{s, g\}$ pair observed in the test set, a prediction is made of the expected energy expense incurred by that driver for that trip, $E_{s\to g}$. This value is then compared to the energy requirement computed for the actual trajectory taken. As an evaluation metric, we employ the relative prediction error ϵ, for

trajectory ς given the learned weight vector w, as compared to ground-truth,

$$\epsilon = 100 \cdot \left| \frac{E_{s \to g}}{E_{\varsigma}} - 1 \right|. \tag{4.44}$$

Figure 4.13 shows the relative prediction error for 50 users as the amount of training data is increased. This is akin to more trajectory information becoming available over time. The figure indicates that the prediction error decreases rapidly from that obtained using the initial model as the number of training trajectories is increased. Beyond 16 training trajectories, improvements are more marginal until, at 28 training trajectories, the overall prediction error reaches ca. 6.5%. In our formulation we compute $E_{s \to g}$ as an expectation over the entire set of possible trajectories from the start to the goal state. The benefit of this approach over considering, for example, only the most likely trajectory given a driver's preferences is also shown in Figure 4.13. The trajectory from start to goal state corresponding to the maximum-likelihood estimate (MLE) is found by computing the shortest path through the road network using the learned segment cost. While the trend of the MLE solution is similar to that found when computing the full expectation, in that the prediction error decreases as information is added to the system, the results suggest overall a performance decrease of ca. 1% compared to using the entire distribution. In contrast, our approach computes an expectation over the entire distribution over trajectories.

We further consider two alternative models for path cost: travel time and distance. Results for both of these are also shown in Figure 4.13, which indicates that these commonly used estimates result in considerably higher overall prediction error. This is confirmed by a statistical sign test [36] at the 99% confidence level, and suggest that drivers indeed exhibit more complex preferences when it comes to route choice.



Figure 4.14: The RobotCar platform (a modified Nissan Leaf) used for data collection.

4.7.2 Energy Model Accuracy

In this section we provide the second part of the system evaluation, measuring the accuracy of $p(E_{s\to g} \geq E_{soc})$ when driver using policy π is perfectly known and thus evaluation of $E_{s\to g}$ is driven by the ability to accurately predict energy E_{ς} spent on a particular trajectory ς.

4.7.2.1 Data

Our approach is evaluated using data gathered from over 140 miles of driving using an all-electric Nissan Leaf (Figure 4.14). The vehicle has been modified to record positioning information provided by a GPS unit. Energy usage information for various vehicle components and state of the battery charge are recorded as reported by the CAN-bus. The corresponding road-network information (including values for road segment features) required for attainability map computation is extracted from OpenStreetMap [40] for a surrounding area with a radius of 100 miles. For data association, we matched GPS trajectories to sequences of traversed road segments using a hidden Markov model as described in [69]. In order to perform a meaningful quantitative evaluation, a single route is chosen (covering a variety of road types) and traversed ten times by the same driver.

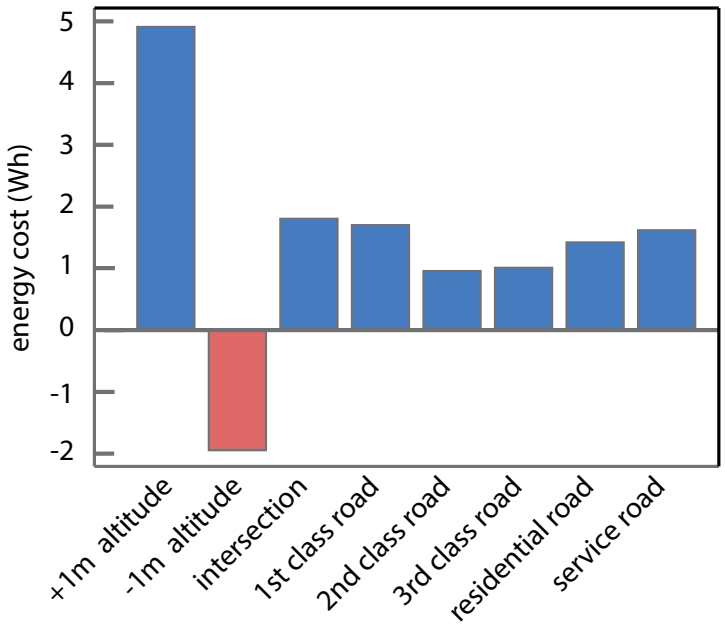


Figure 4.15: The impact of different factors on the energy consumption in the linear model.

4.7.2.2 Evaluation

Figure 4.16 gives an indication of the types of maps produced by our model for a range of scenarios. As the contribution of this work is a probabilistic model of energy consumption, for these plots the driver's route preferences are set by manually specifying w_ρ. Parameters for the energy model are learned from the data. In particular, the figure shows the attainability map for a given vehicle location and remaining battery charge $\mu_{soc} = 1.2kWh$ and $\sigma_{soc} = \epsilon_{soc} \cdot \mu_{soc}$ for an RMS state-of-charge estimation precision ϵ_{soc} which was, apart from Figure 4.16f, set to $\epsilon_{soc} = 3\%$ as indicated in [18].

The evolution of the maps is rather intuitive. Figures 4.16a-c demonstrate the effect of different driving preferences on the attainability maps. While a detailed interpretation is heavily dependent on the particular road network considered, in this case a minimum-time route preference allows to travel furthest, while both motorway and non-motorway preferences are more restrictive. Figure 4.16d shows an extended driving range when parameters w_μ, w_σ are trained while the ECO mode is ON, thereby limiting engine power.

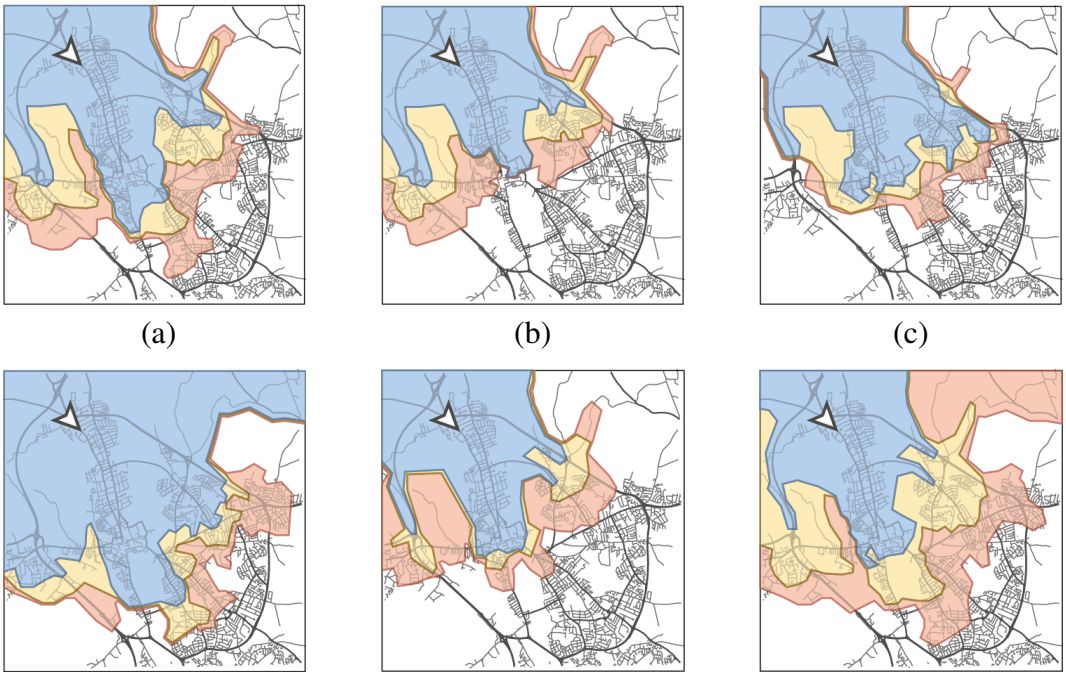


Figure 4.16: Examples of driver specific probabilistic attainability maps showing 90% (blue), 50% (yellow) and 10% (red) confidence bounds for various route preferences, driving styles and levels of battery charge estimation noise. Top row: [**a**] minimum-time route preference, [**b**] motorway route preference and [**c**] motorway avoidance route preference. Bottom row: minimum-time route preference with [**d**] ECO mode ON, [**e**] air conditioning ON and [**f**] 10% uncertainty in battery charge.

Similarly, Figure 4.16e shows a reduced range when the air conditioning is turned ON. Finally, increasing the uncertainty in the battery state-of-charge estimate leads to increased width of the confidence bands reflecting increased uncertainty in the attainability of destinations. The advantage of using linear model is that we can directly associate the impact of different environmental factors on the energy consumption, as shown in Figure 4.15.

In order to quantitatively evaluate the system, we split the data set into eight non-overlapping trajectories, each of which is traversed ten times. Leave-one out cross-validation is used to provide energy prediction results for each of these trajectories. For these experiments we consider the battery state-of-charge estimate to be noise free. Table 4.2 compares the predictions obtained from our system with the empirically measured values. On average, our method is able to estimate μ with an average error of 6%, which

#	energy used [Wh]	energy predicted [Wh]	μ error [%]	σ error [%]
1	280.69 ± 27.17	291.52 ± 24.28	3.86	10.63
2	102.20 ± 27.48	102.41 ± 11.81	0.21	57.00
3	132.99 ± 13.49	132.60 ± 27.90	0.29	106.80
4	230.53 ± 27.35	219.42 ± 19.76	4.82	27.74
5	368.20 ± 56.05	403.36 ± 43.02	9.54	23.23
6	362.58 ± 50.32	313.76 ± 66.89	13.46	32.93
7	244.38 ± 19.23	259.08 ± 21.90	6.01	13.91
8	464.24 ± 24.16	433.97 ± 29.85	6.51	23.54

Table 4.2: Predicted mean μ, and standard deviation σ, compared to the empirically observed distributions for eight independent trajectories. The difference in accuracy is caused by the fact that some trajectories, such as the ones in the city, exhibit more variance in traffic conditions and thus also in the corresponding energy consumption.

is commensurate with state-of-the-art approaches such as [65].

To assess the accuracy of our system's predictions of attainability, we compare for each one of the eight test trajectories the predicted probability of attainability with the empirically computed one. In particular, for each test trajectory, we vary the available battery energy to control the probability of traversing it all the way. We then compare this to the empirical frequency of successful traversals which require up to that amount of energy. Assuming that the car operation remains unaltered by the amount of energy in its battery this procedure is exactly equivalent to charging the battery to a given level and testing whether the car reaches its destination. In reality the battery had additional charge for a safe return trip. The results of these experiments are shown in Figure 4.17, where we plot the empirical frequencies against the ones obtained by our model. If the system were perfectly calibrated (i.e. the predictions are exactly right) the relationship would be perfectly linear (grey line). The figure suggests that, while noise exists due to prediction errors, on average the predictions made indeed correspond to those observed.

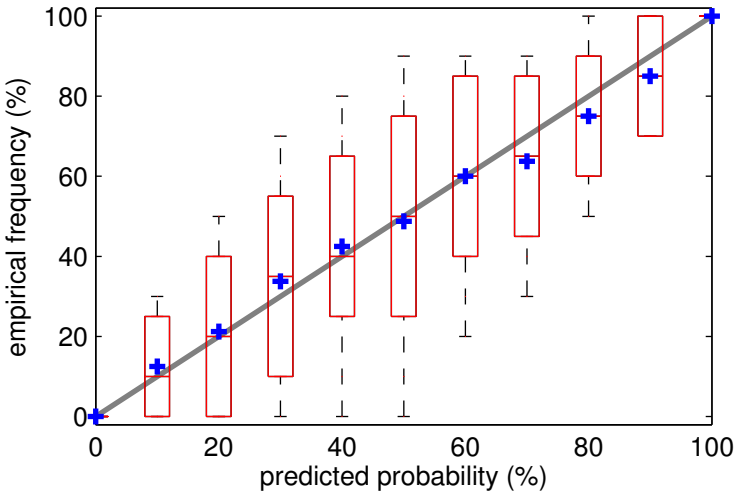


Figure 4.17: The empirical proportion of successful attempts to reach the destination as a function of predicted attainability probability. Optimal response is linear, marked in grey.

4.8 Implementation

The presented method is implemented as a part of a more complex system providing the user real-time updates of attainable range (see Figure 4.18). First, in the preprocessing part, records of traversed routes together with extracted road segment features $f_{s,a}$ are used to learn the model parameters of energy consumption w_μ, w_σ and route preferences w_ρ. These are subsequently used to predict the energy consumption $\mu_{s,a}$ $\sigma^2_{s,a}$ and user preference cost $\mathcal{R}_{s,a}$ for all road segments of the map. Next, in the online part, these are taken as input to the algorithm, together with the car position s_s and battery charge μ_{soc}, σ_{soc} to produce the attainability of all destinations for a specified confidence threshold p_t. This information is then presented to the user in the form of a visual attainability map.

Our implementation of the algorithm uses a GPU-accelerated computation carried out on a standard MacBook Pro capable of performing prediction every 5 seconds, for a map in radius of 100 miles surrounding the car, containing on average 1,000,000 road segments. This is sufficient for a real-time application given the car's maximum drivable

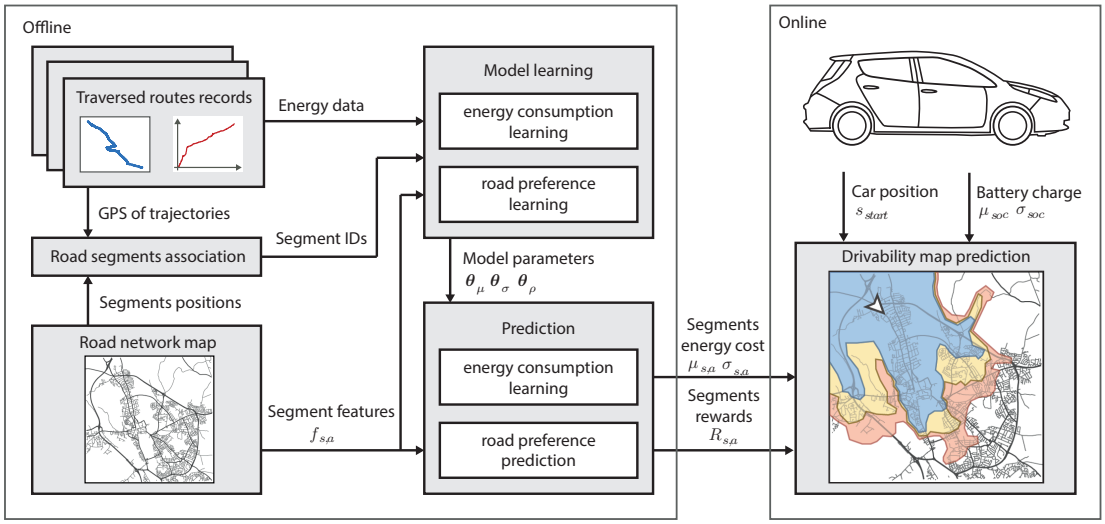


Figure 4.18: System architecture overview.

range, position and battery charge rate of change. If a prediction for a larger area is desired then we could apply methods such as road-network graph pruning based on road preference, considering only the most important roads.

4.9 Conclusions

In this chapter we learned and evaluated policies in a large-scale Markov decision process, to solve the problem of efficiently predicting user drivable range.

We proposed an efficient framework to account for both a driver's routing preferences and the energy consumption of the car. Both of these factors are learned directly from data as a life-long learning system, in a manner entirely transparent to the user – simply by observing the trajectories along which a vehicle travels. In particular, we leverage an inverse reinforcement learning approach to learn generalised driving preferences and machine learning regression to learn vehicle energy consumption. We show that our approach can reduce the error in predicted energy use to almost one half of that obtained when predicting energy consumption based on common heuristics.

Chapter 5

Planning in Partially Observable Environments for Sensing Optimisation

So far in this thesis we have covered learning internal representations, transition and observation models $\mathcal{S}, \mathcal{T}, O$ for partially observable environments modelled by a Hidden Markov Model (HMM) and learning and evaluating appropriate reward function $\mathcal{R}$ from expert observations in a Markov Decision Process (MDP). In this chapter we cover a method for belief planning in partially observable environments to compute an optimal policy π of a Partially Observable Markov Decision Process (POMDP). In particular this policy is optimised for a robot to succeed in a given task and, at the same time, minimising necessary sensing to do so. We present a new method for optimising necessary sensing energy in a trajectory-following scenario. While planning in partially-observable environments is a notoriously difficult, we observe that in this scenario the belief space is particularly limited and thus can be explored very efficiently. This leads to a fast polynomial-time solution for computing the optimal policy, which is superior to a more simple greedy-based approach.

More specifically, in this chapter we explore the idea of reducing a robot's energy consumption while following a trajectory, by turning off the main localisation subsystem

and switching to a lower-powered, less accurate odometry source at appropriate times. This applies to scenarios where the robot is permitted to deviate from the original trajectory, allowing for energy savings. Sensor scheduling is formulated as a probabilistic belief planning problem. Two algorithms are presented which generate feasible perception schedules: the first is based upon a simple heuristic; the second leverages dynamic programming to obtain optimal plans. Both simulations and real-world experiments on a planetary rover prototype demonstrate over 50% savings in perception-related energy, which translates into a 12% reduction in total energy consumption.

To the best of our knowledge this is the first work to investigate sensor scheduling in the context of robot navigation as a means to save energy during operation. Specifically, beyond exploiting localisation accuracy for energy management, the contributions of this work are:

- the framing of the problem as a belief Markov decision process over robot poses;
- the description of a greedy algorithm which employs a simple heuristic, ensuring the feasibility of the perception schedules;
- the description of a belief planning algorithm which leverages dynamic programming to provide *optimal* perception schedules.

The work described in this chapter directly contributed to the publication:

- P. Ondruska, C. Gurau, L. Marchegiani, C. H. Tong, and I. Posner, *"Scheduled Perception for Energy-Efficient Path Following,"* in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 2015.

The rest of the chapter is organised as follows. First, in Section 5.1 we define the problem of perception scheduling as a paradigm of saving energy in a path-following scenario. This formulation explicitly accounts for drift and trajectory shape (though the

full trajectory need not be available a-priori) as well as the time required to power-cycle the perception system in order to compute perception schedules. Next, in Section 5.2 we frame the problem as belief-MDP planning and precisely define the considered model and assumptions taken. In Section 5.3 we describe two methods for computing feasible perception schedules, one optimal and the second suboptimal but faster to evaluate. Both methods are probabilistically bounded to limit path deviations to a user-specified value. The evaluation of the approach is described in Section 5.5. Finally, we discuss the related work in Section 5.6, and conclude in Section 5.7.

5.1 Perception Scheduling Problem

Robots require energy to operate, and they only have access to limited energy storage during missions. As we extend the reach of autonomous systems to operate in remote locations, over long distances and for long periods of time, energy considerations are becoming increasingly important. To date, these considerations are often brought to bear in schemes where trajectories or speed profiles are optimised to minimise the energy required for actuation (see, for example, [61, 10, 42]). Here we take a different yet complementary approach in considering the energy expenditure for sensing (and, implicitly, computation) associated with navigation. In particular, our goal is to activate the perception system only as required to maintain the vehicle within a given margin around a predetermined path. As the main navigation sensors are switched off and the robot reverts to a lower-powered, less accurate odometry source for parts of the trajectory, any associated computation will also be reduced, leading to further savings in energy.

Naively, such *perception schedules* could be constructed by switching sensors on and off randomly or according to, for example, a fixed frequency. However, the drawback is that no heed is paid to drift in the robot's position with respect to the original trajectory: it may not be desirable to deviate by more than an allowed margin from the predetermined

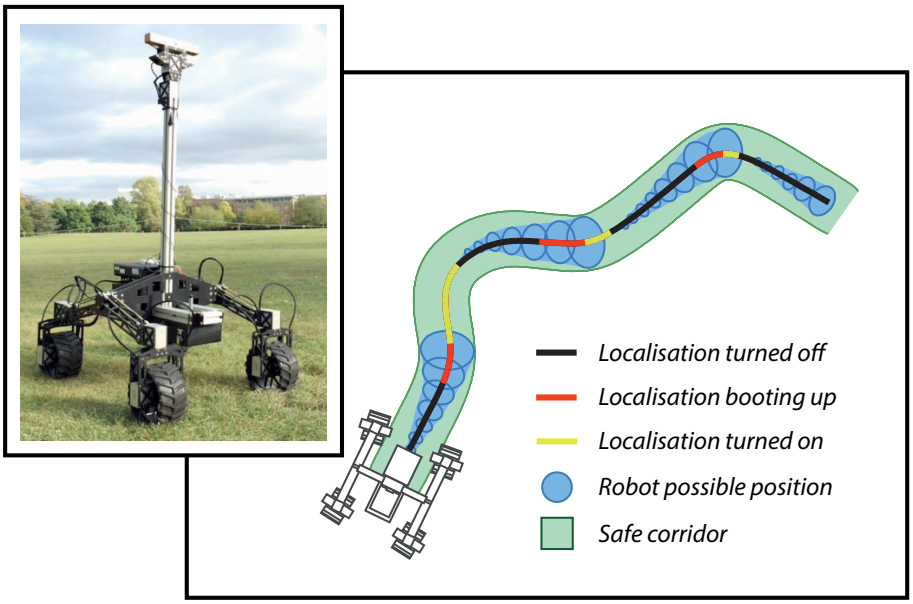


Figure 5.1: An example of an energy-optimal perception plan as generated by our method in a path-following scenario. The robot needs to run the sensor only in selected parts of the trajectory (red and yellow) which guarantees that it stays within the specified corridor (green) with 90% confidence. This plan saves 50% of the energy consumed by the localisation system, which is 12% of the total system consumption.

route. This arises, for example, in a planetary exploration scenario when conducting long traverses over featureless terrain. Other possible considerations include the traversability, obstacles, and robustness of the localisation system to deviations from the original path. Such naive approaches would also need to be tuned to individual trajectories as savings would depend significantly on trajectory shape.

Consider a situation where the robot has to follow a particular trajectory φ specified as a sequence of poses such that

$$\varphi = \{x_1, x_2, ..., x_N\}. \tag{5.1}$$

The total energy E_{total} required to traverse the trajectory can be split into the energy required to run the navigation system (sensors and respective processing), E, and the energy consumed by rest of the system (without loss of generality we attribute this mainly

to the motors), E_M,

$$E_{total} = E + E_M. \tag{5.2}$$

Our goal is to minimise the total energy E_{total} by minimising E while maintaining E_M constant. This is achieved by enabling/disabling the localisation system according to a perception schedule ς. Once localisation is turned on, the robot continuously determines its pose and follows the trajectory. When localisation is off, the robot tries to blindly follow the shape of the trajectory using dead reckoning. In this case, the robot will deviate from the intended trajectory as a result of odometry drift. When localisation is re-enabled, the robot observes its position and manoeuvres back towards the path. Example of such a perception plan is displayed in Figure 5.1 .

As discussed in Section 5.5, minor path deviations do not cause an increase in motor energy consumption or hamper the ability to localise. We thus consider the optimisation of E to require only the minimisation of E while E_M is independent of ς. Moreover, as described later, to limit the magnitude of potential path deviations and further constraints we consider a set of schedules $\mathcal{F}$, to be *feasible*. Therefore, the task of perception scheduling is to find an optimal, feasible plan ς^* that minimises the energy consumption attributed to the localisation system:

$$\varsigma^* = \arg\min_{\varsigma \in \mathcal{F}} E_\varsigma. \tag{5.3}$$

5.2 The Model

In this section we formalise the notion of *perception schedule* energy E_ς and it's feasibility. Specifically, we formalise the problem of perception scheduling in the context of belief Markov decision processes [6, 93] from Chapter 2. Specifically, we describe the problem as a set of belief states B, a set of actions $\mathcal{A}$, a transition function $\mathcal{T}$ and a cost function E_a equivalent to negative reward.

5.2.1 Perception Plan Energy

We model the perception schedule as a sequence of actions, a_t such that

$$\varsigma = \{a_1, a_2, ..., a_N\}, \tag{5.4}$$

where t denotes the time at which an action is taken. These actions correspond to decisions the robot can make with respect to its navigation system, and each action incurs some energy cost E_a. We model E simply as a sum of energy demands for individual actions (see Figure 5.2),

$$E_\varsigma = \sum_{a \in \varsigma} E_a. \tag{5.5}$$

When the localisation system is enabled, the robot has the choice to keep it turned on, a_{on}, incurring an energy cost $P_{on} \cdot \Delta t$ where P_{on} is power demand of localisation when active, or to turn it off at zero cost. Similarly, when localisation is disabled, the robot can keep it turned off, a_{off}, at no cost or boot the localisation system at a cost $E_{start} \cdot \frac{\Delta t}{T_{start}}$ equivalent to an average energy demand over time T_{start} required to turn localisation on.

5.2.2 Perception Plan Feasibility

We consider a set of feasible schedules $\mathcal{F}$, such that

$$\mathcal{F} = \mathcal{F}_1 \cap \mathcal{F}_2, \tag{5.6}$$

where $\mathcal{F}_1$ and $\mathcal{F}_2$ separately satisfy different feasibility constraints.

5.2.2.1 Switching Frequency Constraint

First we consider that localisation can not be turned on instantaneously but requires time T_{start} to initialise (turning on the sensor, launching an executable, etc). This means a

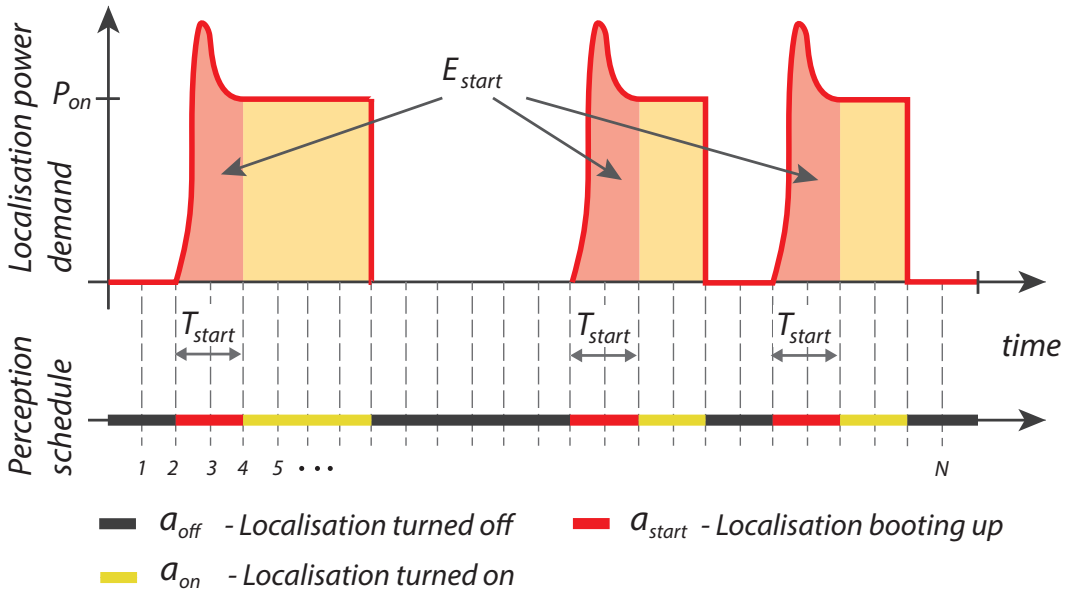


Figure 5.2: The energy E_ς required for localisation along the trajectory as given by the perception schedule ς. We assume that the localisation system needs time T_{start} and energy E_{start} to boot and then has a constant power demand P_{on} while operational.

sufficient number of a_{start} actions need to precede the first action a_{on}:

$$\mathcal{F}_1 = \left\{ \varsigma \;\middle|\; \begin{array}{c} a_{t+1} = a_{on} \wedge a_t \neq a_{on} \rightarrow a_{t-t'} = a_{start} \\ \forall t, t' : 0 \leq t' \leq T_{start} \end{array} \right\} \tag{5.7}$$

5.2.2.2 Belief Constraint

Secondly, we want the robot to remain (with a probabilistic guarantee) within a safe corridor $\mathcal{C}_t : \mathcal{S} \rightarrow [0, 1]$ of the planned trajectory even when localisation is turned off. In this case the real pose s_t is not known. Instead we can compute a belief state $\mathcal{B}_t$, which is a distribution over the possible robot poses at time t:

$$Bel(s_t | z_{1:t}, a_{1:t}) = p(s_t | z_{1:t}, a_{1:t}). \tag{5.8}$$

This belief evolves based on the actions taken and can be encoded in the form of a transition function $\mathcal{T}$. When the localisation is turned on we approximate this belief by a Dirac

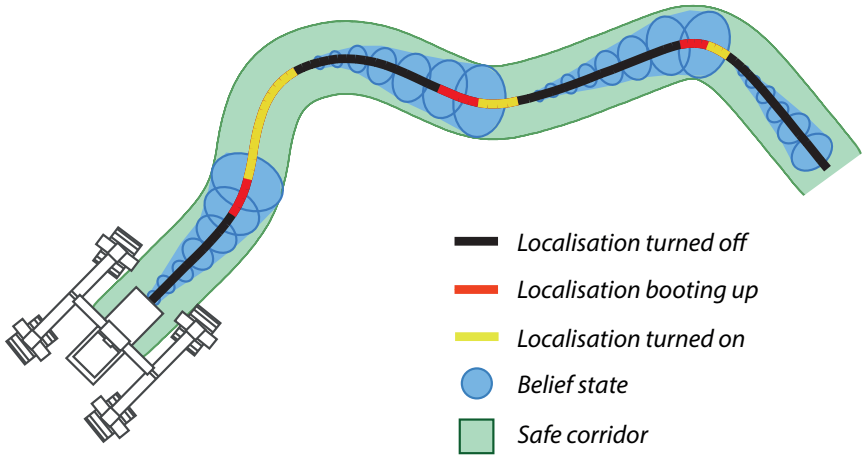


Figure 5.3: The evolution of the belief over the robot's pose s_t around the trajectory φ based on the perception schedule, ς. When localisation is turned on, the position is known and the belief $\mathcal{B}_t$ is given by a Dirac delta function, $\delta(s_t - x_t)$. When localisation is turned off, the belief evolves based on the robot's motion model. The perception schedule is conceived such that the robot always remains in the designated safe corridor area, $\mathcal{C}t$.

delta function centred on the intended pose. This reflects the assumption that localisation is exact. As the camera is turned off, the belief over poses evolves based on the command input u_t used to drive the robot from pose x_t to x_{t+1} along the trajectory, as well as on the odometry noise model (Figure 5.3),

$$\mathcal{T}(\mathcal{B}_t, a_t) = \begin{cases} \delta(s_t - x_t) & \text{if } a_t = a_{on} \\ p(s_t|\mathcal{B}_{t-1}, u_t) & \text{otherwise.} \end{cases} \tag{5.9}$$

The specific motion model encoding $p(s_t|\mathcal{B}_{t-1}, a_t)$ we consider here is described in Section 5.5.

As illustrated in Figure 5.4, every perception schedule $\{a_1, a_2, ..., a_N\}$ corresponds to a sequence of belief states $\{\mathcal{B}_1, \mathcal{B}_2, ..., \mathcal{B}_N\}$. We define the safe obstacle-free area around the robot's intended pose s_t at time t as $\mathcal{C}t$. The desired behaviour is that the robot will not exit this zone. Due to the randomness of the motion process when localisation is turned off, the risk of the robot leaving this corridor can not be completely avoided. Instead, we

leverage a probabilistic guarantee that the true robot pose is contained within this area with probability greater than or equal to p_{conf}. We are therefore interested in the set of solutions satisfying,

$$\mathcal{F}_2 = \{\varsigma \mid p(s_t \in \mathcal{C}_t|\mathcal{B}_t) \geq p_{conf} \ \forall t\}, \tag{5.10}$$

where

$$p(s_t \in \mathcal{C}_t|\mathcal{B}_t) = \int_{s_t} \mathcal{C}_t(s_t) Bel(s_t|z_{1:t}, a_{1:t}). \tag{5.11}$$

5.3 Solution Method

In this section, we describe the two algorithms used for computing feasible perception schedules. The first one, the *greedy* algorithm, is simple to implement: it does not require knowledge of the full trajectory in advance and, in practice, can save a substantial amount of perception-related energy. However, this algorithm may produce suboptimal behaviour in some situations as a result of taking decisions which are only locally optimal. The second algorithm overcomes this limitation by considering the entire trajectory (or up to a given planning horizon) and presents an efficient belief planning implementation based on dynamic programming [9]. This algorithm is able to find the *optimal* trajectory through the belief space attainable by the robot which, in turn, correspond to the optimal perception schedule, ς^* (see Figure 5.4).

5.3.1 Greedy Algorithm

A simple procedure to obtain a valid plan is to keep the sensor off as long as possible while simultaneously modelling the robot's belief state $\mathcal{B}_t$. When the uncertainty on the state becomes too high, i.e. $p(s_t \in \mathcal{C}_t) < p_{conf}$, we turn the sensor on, localise and turn the sensor off again. The only constraint is related to the boot time of the sensor, T_{start}.

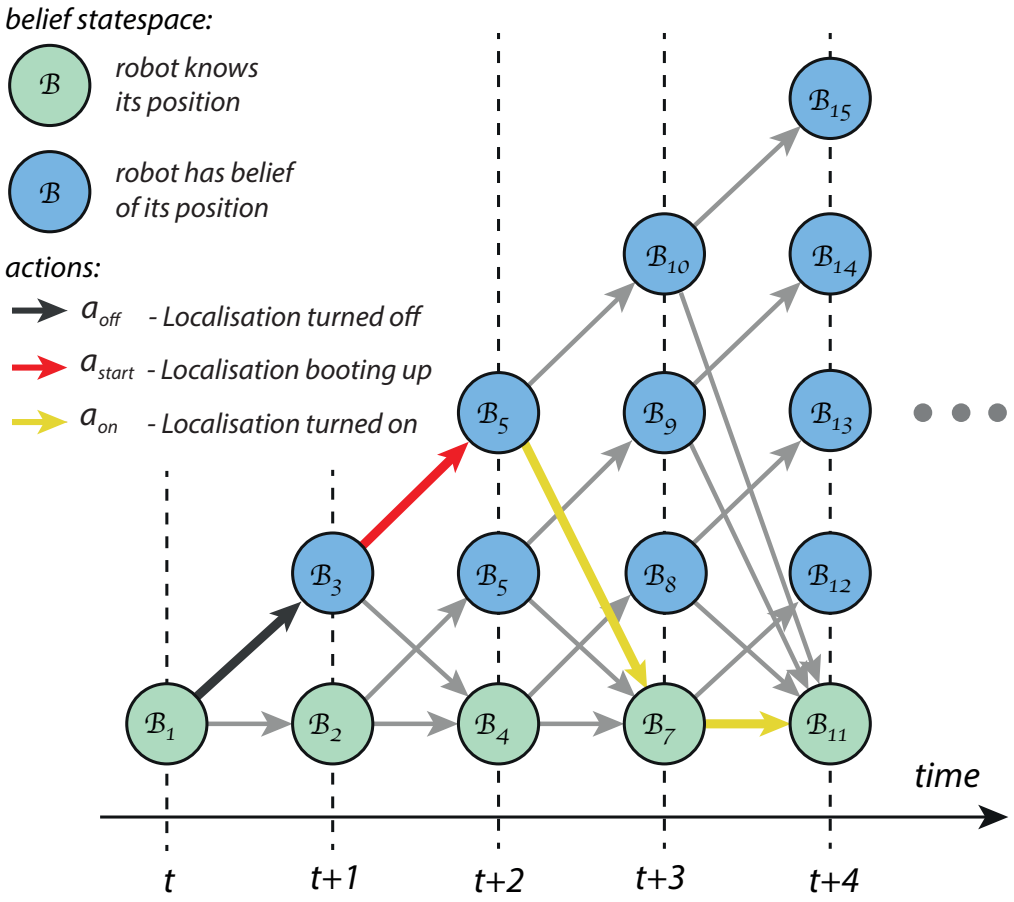


Figure 5.4: The belief space attainable by the robot starting at time t and a possible perception schedule indicated by the actions performed (coloured arrows). At every time step the robot has a choice either to localise or not. The latter results in more uncertainty over its current pose. Localisation has an energy cost associated with it, but remains the only mechanism by which pose uncertainty can be limited.

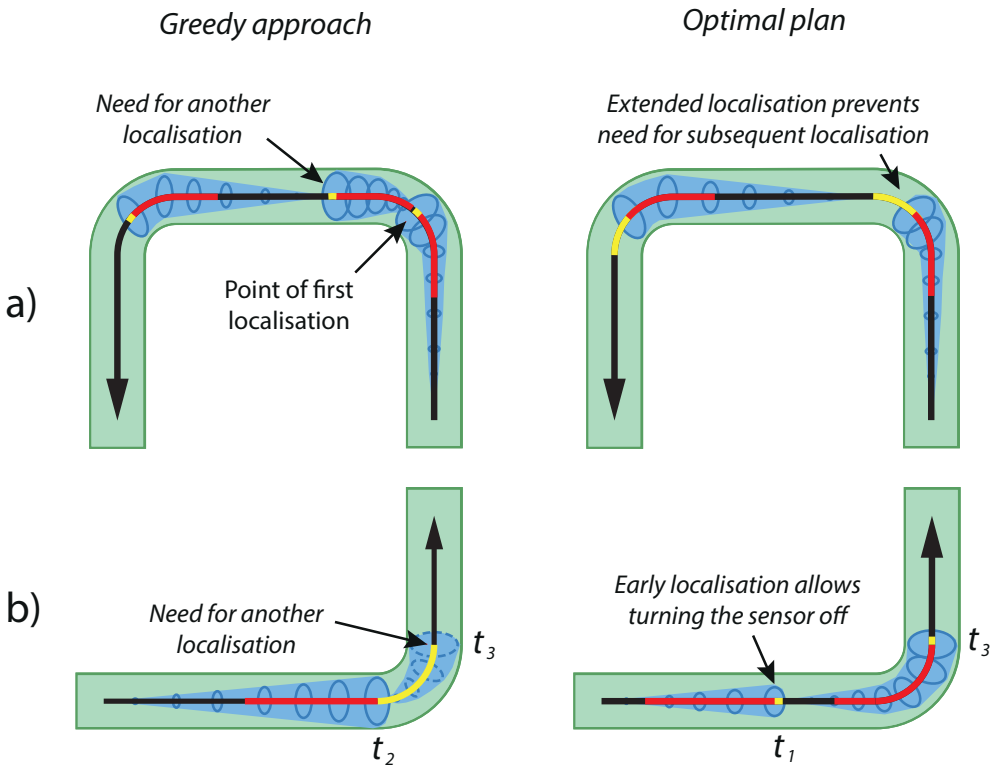


Figure 5.5: Example scenarios when a locally optimal decision results in a globally sub-optimal solution. A detailed description is provided in the text.

This means that we need to simulate the evolution of the belief state in advance for a time horizon T_{start} in order to be able to start the sensor in time. It also means that we always need to know the trajectory shape only for the next T_{start} seconds. The principle can be efficiently implemented as an online perception controller as shown in Algorithm 5. In such cases, the perception schedule ς is not created explicitly but is generated on-the-fly as a result of the controller's decisions. This simple procedure is surprisingly effective. As described in section 5.5, it can save over 40% of the perception-related energy consumption on the robotic platform used in this work.

Algorithm 5 Greedy Planning Strategy

Input:	$\mathcal{B}_1$	*current belief state - see Equ. 5.9.*
	T_{start}	*time required to localise*
	$x_{1:T_{start}}$	*trajectory for T_{start} seconds*
	$\mathcal{C}_{1:T_{start}}$	*safe corridor for T_{start} seconds*
	p_{conf}	*confidence of staying in the corridor*
Output:	a	*perception action taken*

Forward simulate belief state evolution.

1. **for** $t = 2, 3, ..., T_{start}$ **do**
2. $\quad \mathcal{B}_t = p(s_t|\mathcal{B}_{t-1}, u_t)$
3. $\quad$ **if** $p(s_t \in \mathcal{C}_t|\mathcal{B}_t) < p_{conf}$ **then**
4. $\quad\quad$ **return** $a = \begin{cases} a_{on} & \text{if } \mathcal{B}_1 = \delta(s_1 - x_1) \\ a_{start} & \text{otherwise} \end{cases}$
5. **return** $a = a_{off}$

However, if more complex trajectories are followed, this behaviour can be suboptimal. Consider the scenario shown in Figure 5.5a. When performing a motion resulting in high pose uncertainty (e.g. turning), keeping the sensor on until the motion is finished results in higher energy savings, since it prevents the later need for relocalisation. Similarly, in Figure 5.5b, if the energy to localise (E_{start}) is low but the interval T_{start} is long, it would be optimal to localise early, at t_1, and then turn the sensor off. Waiting until t_2 is suboptimal and requires more energy, since the interval $t_3 - t_2$ is shorter than the time required to boot the system (T_{start}) and the sensor would need to be kept on.

5.3.2 Belief MDP Algorithm

Given knowledge of the robot's initial position as well as of the entire trajectory, the optimal perception schedule can be computed as the energetically cheapest path to any belief state, $\mathcal{B}_N$, through all attainable belief states, $\mathcal{B}_1, \mathcal{B}_2, ..., \mathcal{B}_N$. It is important to observe that even though there are infinitely many possible belief states, our robot can visit at most $\frac{N(N+1)}{2}$ different ones. These states are produced by turning off the sensor at time t, and continuing blind for the remainder of the trajectory. This bound is important because it indicates that all possible belief states can be precomputed which in turn allows the deployment of deterministic planning algorithms commonly used for *finite* Markov decision processes [5]. This results in the optimal schedule $\varsigma^*_{\mathcal{B}}$ yielding minimal energy usage, $E^*_{\mathcal{B}}$ for any starting state, $\mathcal{B}$. The optimal schedule for the whole trajectory, $\varsigma^*_{\mathcal{B}_1}$, is equal to $\varsigma^*_{\mathcal{B}_1}$.

However, computing all possible belief states in advance is inefficient. Instead, we employ a dynamic programming approach to compute optimal plans ς^*_t at each time step, as well as the associated energy consumption E^*_t, only for belief states $\mathcal{B}_t = \delta(s_t - x_t)$ in the order $t = N, N-1, ..., 1$. This results in an efficient solution as detailed in Algorithm 6. Reachable belief states are computed only when required and previously determined optimal values are reused. The algorithm has time complexity $\mathcal{O}(N^2 \cdot T(\mathcal{B}))$ and memory complexity $\mathcal{O}(N + M(\mathcal{B}))$, where $T(\mathcal{B})$ is the time and $M(\mathcal{B})$ is memory complexity corresponding to the belief state evolution (see lines 2-3 of Algorithm 5 and 7-8 of Algorithm 6). Our implementation uses a particle filter to represent the belief states, which results in time complexity $\mathcal{O}(N^2 L)$ and memory complexity $\mathcal{O}(N + L)$, where L is the number of particles.

Algorithm 6 Optimal plan computation.

Input: $x_{1:K}$ *trajectory*
$\mathcal{C}_{1:K}$ *safe corridor*
P_{on} *power demand of perception*
T_{start} *time required to localise*
E_{start} *energy required to localise*
p_{conf} *confidence of not leaving the corridor*
Output: ς^* *perception plan*

Backward pass - compute E_t^* and ς_t^* for each t
1. **for** $t = N-1, N-2, ..., 1$ **do**
Value of keeping the sensor on
2. $E_t^* = \Delta t \cdot P_{on} + E_{t+1}^*$
3. $\varsigma_t^* = \{a_{on}, \varsigma_{t+1}^*\}$
Turn the sensor off and turn it on at time t'
4. $\varsigma = \{\}$
5. $\mathcal{B}_t = \delta(s_t - x_t)$
6. **for** $t' = t+1, t+2, ..., N$ **do**
Simulate belief state
7. $\mathcal{B}_{t'} = p(s_{t'}, \mathcal{B}_{t'-1}, u_{t'})$
8. **if** $p(s_t \in \mathcal{C}_t | \mathcal{B}_t) < p_{conf}$ **then**
9. **break**
10. **if** $t' < t + T_{start}$ **then**
11. $\varsigma = \{a_{start}, \varsigma\}$
12. **else**
Vale of turning the sensor on at time t'
13. **if** $E_{start} + E_{t'}^* < E_t^*$ **then**
14. $E_t^* = E_{start} + E_{t'}^*$
15. $\varsigma_t^* = \{\varsigma, a_{on}, \varsigma_{t'}^*\}$
16. $\varsigma = \{a_{off}, \varsigma\}$
17. **return** $\varsigma^* = \varsigma_1^*$



Figure 5.6: The robotic platform used for experiments. The robot is a commercially-available version of a planetary rover. It is equipped with low-power motors, a stereo camera pair for visual localisation and a CPU to carry out the computation.

5.4 Implementation

The robotic platform used in these experiments is the Q14 planetary rover from Advanced Robotics Concepts shown in Figure 5.6. It is designed to mimic the configurations and specifications found on rovers deployed for planetary exploration. In particular, robot locomotion is provided by eight low-power motors, independently articulating steering and four rotating wheels, travelling at a maximum speed of $0.5m/s$. The robot is equipped with a Point Grey Bumblebee XB3 stereo camera for visual localisation and wheel odometry sensors aiding motor controllers. The computation is carried out on an on-board MicroSVR computer containing an Intel Core i7 processor and 16GB of RAM. The robot is equipped with current sensors, which allow the measurement of the power consumption of individual motors. This was found to be of the order of $30W$. Power consumption values specified by the manufacturers are used for the CPU ($25W$) and the camera ($5W$). This amounts to an average overall power consumption of $60W$ while moving and localising. We estimate energy savings when the localisation system is turned off to be of the

order of $10W$. This comprises $5W$ for the camera as well as an additional $5W$ estimated reduction in CPU power.

The navigation system is based on visual teach and repeat (T&R) [34, 59], which is often employed in the space domain and a similar approach can used in sample and return missions [23, 4]. T&R enables a mobile robot to autonomously follow a previously driven path with high accuracy. First, a trajectory φ is driven by a human operator. During this teach run the system builds a feature map used later for localisation. Next, the perception schedule ς is computed using the presented algorithms. Finally, the robot attempts to follow the path autonomously, localising in the map according to the perception schedule. The entire system architecture is illustrated in Figure 5.7. In order to evaluate the robot's actual trajectory when adhering to perception schedules, the camera was not switched off in reality. Sensor deactivation is instead simulated by cutting off the image stream in software and subtracting the appropriate energy values from the total for the entire system to obtain the reduced energy demand. Note that, although we tested the behaviour and the performance of our methods in a T&R context, this is not the only scenario in which it is possible to use these methods. The utility of perception schedules during path following is general and does not depend on the source of the trajectory.

5.4.1 Odometry Model

We use a standard odometry motion model [57, 93] to approximate the robot's drift when the localisation system is not active. In this model, let $(\Delta x, \Delta y, \Delta\theta)_t$ be the command input fed to the controller required to move the robot from state s_t to s_{t+1}. The robot's motion during this interval is approximated as a composition of an initial rotation, ϕ_1, an intermediate translation, τ, and a second rotation ϕ_2, as shown in Figure 5.8. These

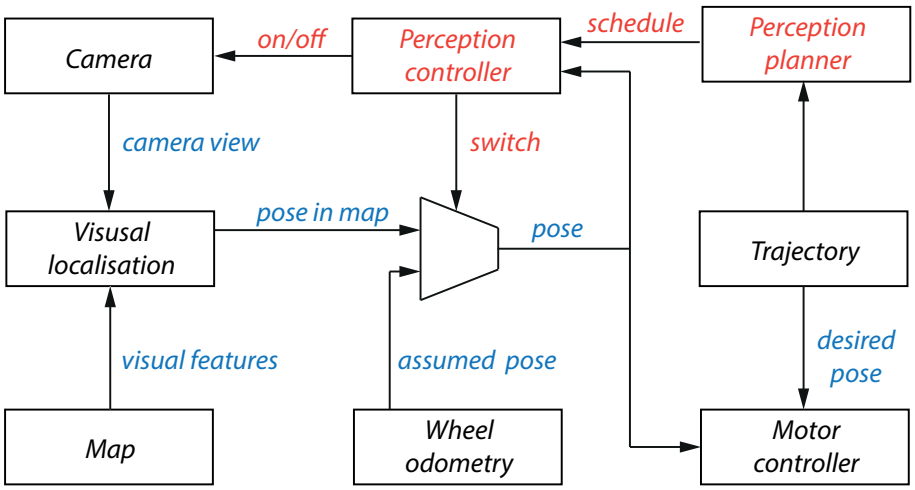


Figure 5.7: The system diagram enabling perception scheduling. The localisation system feeds pose information into a standard trajectory-control pipeline. The perception controller turns the sensor on or off based on the pre-computed perception schedule.

motions are described by

$$\phi_1 = \tan^{-1}(\Delta y, \Delta x), \tag{5.12a}$$

$$\tau = \sqrt{\Delta x^2 + \Delta y^2}, \tag{5.12b}$$

$$\phi_2 = \Delta\theta - \phi_1. \tag{5.12c}$$

We assume that the real motion is affected by additive zero-mean Gaussian noise with standard deviation proportional to the motion magnitude:

$$\tilde{\phi}_1 \sim \mathcal{N}(\phi_1, \alpha_1|\phi_1| + \alpha_2\tau), \tag{5.13a}$$

$$\tilde{\tau} \sim \mathcal{N}(\tau, \alpha_3\tau + \alpha_4(|\phi_1| + |\phi_2|)), \tag{5.13b}$$

$$\tilde{\phi}_2 \sim \mathcal{N}(\phi_2, \alpha_1|\phi_2| + \alpha_2\tau). \tag{5.13c}$$

where the tuning parameters $\alpha = (\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ govern the influence of each of the

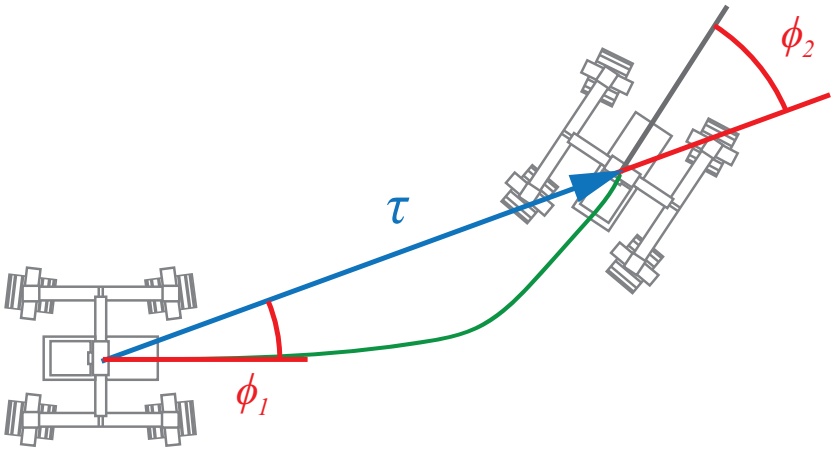


Figure 5.8: The plant model used to describe rover motion [57, 93]. The robot motion (green) in a time interval is approximated by a rotation, ϕ_1, followed by a translation, τ, and a second rotation, ϕ_2. The rotations and the translation are affected by noise.

motion components. Using these definitions, the real robot motion is given by

$$\Delta\tilde{x} = \tilde{\tau}\ \cos(\tilde{\phi}_1), \tag{5.14a}$$

$$\Delta\tilde{y} = \tilde{\tau}\ \sin(\tilde{\phi}_1), \tag{5.14b}$$

$$\Delta\tilde{\theta} = \tilde{\phi}_1 + \tilde{\phi}_2. \tag{5.14c}$$

In our experiments we obtain the parameter vector α by maximising the log likelihood of the robot motion, as reported by the visual localisation system, given the commands used to control the robot on a calibration trajectory of length 85m. The resulting values are given in Table 5.1.

In the implementation of the perception scheduling algorithms, we use the particle filter described in [26, 93] to represent the belief state, with $L = 10000$ particles. The propagation of the belief and the computation of the probability of the robot being in the safe area $\mathcal{C}_t$ (lines 2–3 of Algorithm 5 and 7–8 of Algorithm 6) is then obtained by sampling the described motion model and computing the number of particles in $\mathcal{C}_t$. For a typical trajectory of length $N = 500$ time steps, the generation of the perception schedule

α	*Odometry noise model*	$[0.428, 0.100, 0.054, 0.150]$
P_{on}	*Localisation power demand*	$10W$
T_{start}	*Localisation boot time*	$4s$
E_{start}	*Localisation boot energy*	$0.0111Wh$
p_{conf}	*Confidence threshold*	90%
γ	*Corridor width*	0.9 *m*

Table 5.1: The parameters used to generate the perception schedules in the experiments.

is typically obtained within 0.1s in the case of the greedy algorithm, and within 5s in the case of the belief planning algorithm.

To generate the perception schedules, we consider localisation to be reliable if the distance from the robot's real pose s_t to the expected one x_t is less than $\gamma = 0.9$m, and if the angular error is less than $20°$. This defines the scope of the safe area $\mathcal{C}_t$. Similarly, we assume that the robot can reliably localise after time $T_{start} = 4$s and that the energy consumption required to start up the localisation system is $E_{start} = T_{start} \cdot P_{on} = 0.011$ Wh.

5.5 Experiments

We evaluate the performance of the algorithms presented using a commercially available planetary rover prototype in a path-following task with a total driving distance of 1.8km. We demonstrate that the methods described are able to save over 50% of the sensing-related energy, which constitutes approximately 12% of the overall energy usage of the rover.

5.5.1 Trials

We evaluate the performance of the algorithms on three different trajectories of length 61m, 54m and 53m, as depicted in Figure 5.9. These trajectories were chosen to consider different terrain conditions. The first trajectory is driven on asphalt in an urban area, the second on gravel paths and the third on grass.

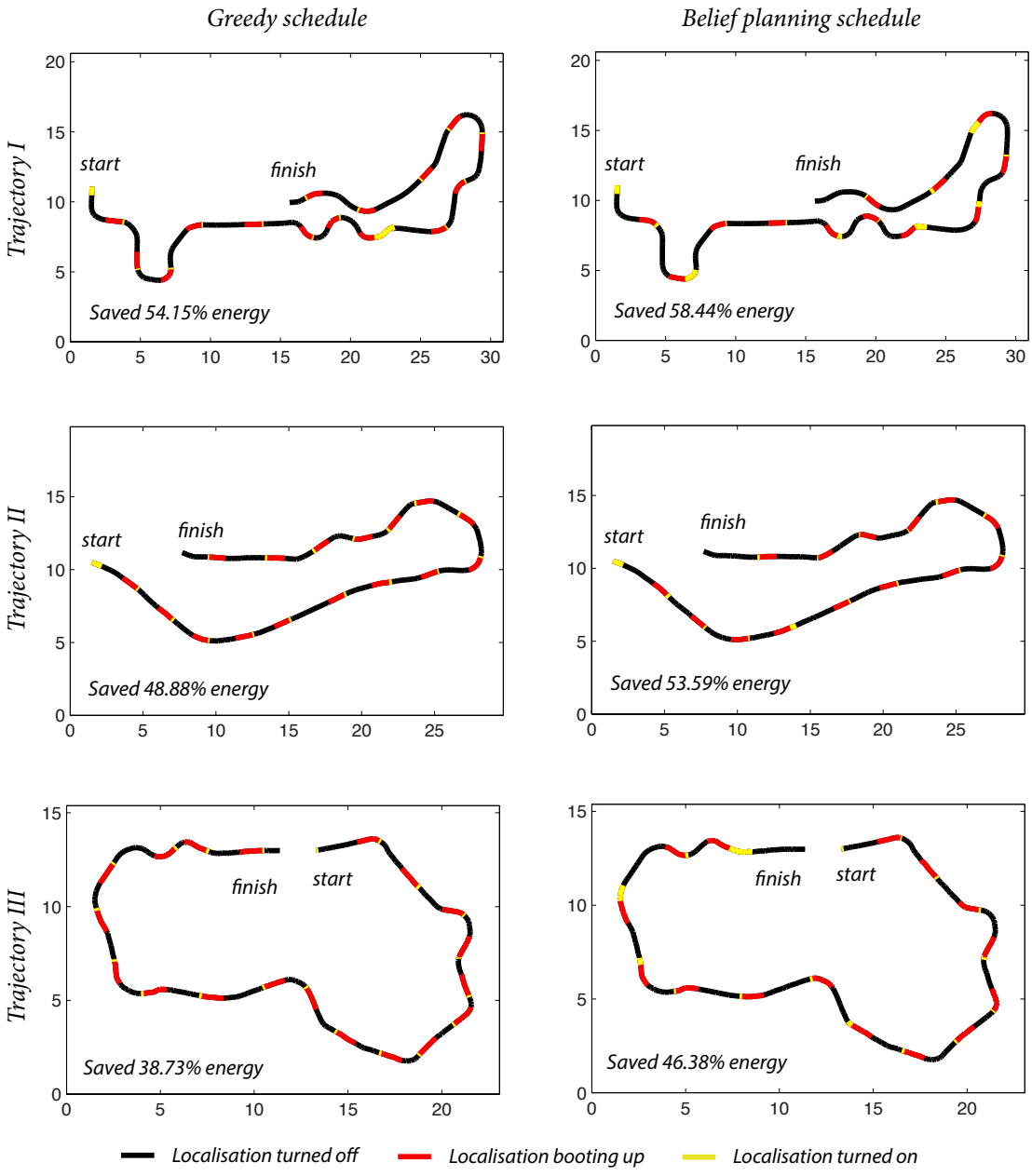


Figure 5.9: Perception schedules for three different trajectories and respective perception energy savings ΔE_P, generated by the *Greedy algorithm* (top row) and the *Belief planning algorithm* (bottom row).

Trajectory I

Perception schedule	K	E_M [Wh]	E_P [Wh]	ΔE_P [%]	ΔE [%]
All-time	3	3.10	0.51	0	0
Greedy	3	3.02	0.24	54.15	9.75
Belief planning	3	2.98	0.21	58.44	11.52

Trajectory II

Perception schedule	K	E_M [Wh]	E_P [Wh]	ΔE_P [%]	ΔE [%]
All-time	3	2.33	0.42	0	0
Greedy	3	2.22	0.21	48.88	9.42
Belief planning	3	2.21	0.19	53.59	12.80

Trajectory III

Perception schedule	K	E_M [Wh]	E_P [Wh]	ΔE_P [%]	ΔE [%]
All-time	5	2.94	0.45	0	0
Greedy	5	2.78	0.27	38.73	9.92
Belief planning	5	2.81	0.24	46.38	10.10

Table 5.2: Energy usage along the three test trajectories measured by K independent trials. Each table corresponds to one trajectory and shows the energy used by the motors E_M, the camera E_P, the proportion of saved perception related energy ΔE_P and total energy ΔE.

To establish a baseline for the energy consumption, the robot follows all trajectories K times with the localisation system active for the entire traversal. Then, for each trajectory, perception schedules are generated using both the *greedy* and the *belief planning* algorithms. Each perception schedule is executed autonomously by the robot K times. The schedules are shown in Figure 5.9. The first two trajectories are travelled $K = 3$ times each, and the last trajectory $K = 5$ times.

5.5.2 Evaluation

All the trajectories driven when executing the perception schedules remained within the safe area as required by the feasibility constraint. For each trial, the perception-related energy E_P, the motor power consumption E_M, the proportion of the saved perception related energy ΔE_P, and the total saved energy ΔE are recorded and compared to the power consumption when localising throughout. The mean values over the K trials for

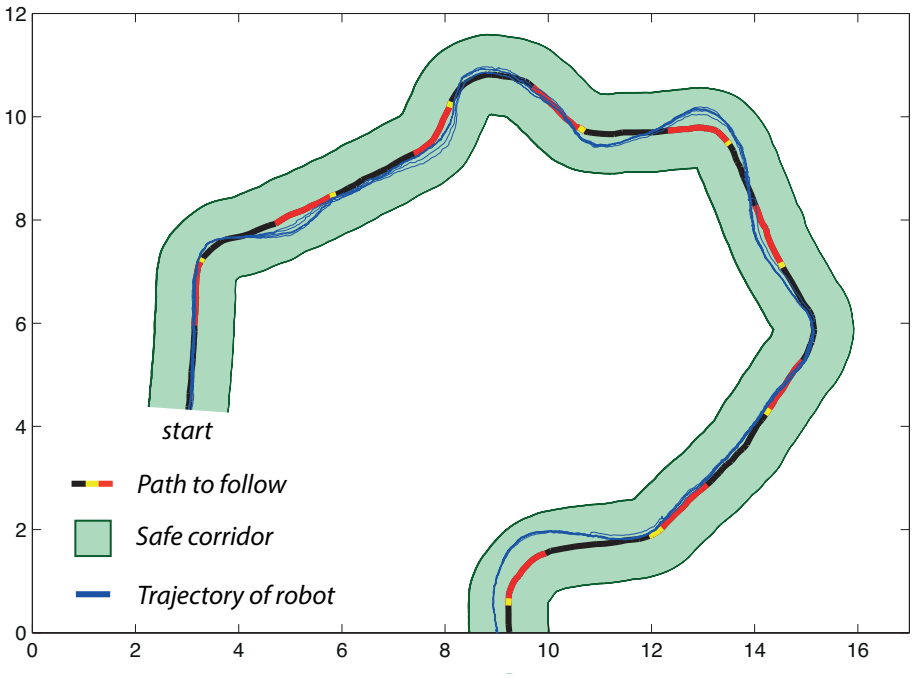


Figure 5.10: Part of the second teach run and the actual trajectories taken by the robot while following the perception schedule. While the robot does not follow the trajectory exactly, it never leaves the safe area [green].

each trajectory are summarised in Table 5.2. In Figure 5.10 we provide details for part of the third trajectory, illustrating the actual path traversed by the robot following the schedule produced by the *belief planning* algorithm. On average we can see that the *greedy* algorithm is able to save around 47.3% of the energy associated with sensing (9.7% of the total energy usage) while employing the *belief planning* algorithm results in a reduction in perception energy of 52.8% (11.5% of the total).

If we examine Figure 5.9 in further detail, we can observe that the shape of the trajectory plays an important role in the energy savings. In particular, the straight segments of the trajectories require localisation less often than the ones characterised by turns. We also notice that the *belief planning* method exhibits the optimised behaviour described in Figure 5.5. Our assumption that the execution of the perception schedule did not cause an increase in motor energy consumption has been validated empirically during these trials. Surprisingly, we observe the opposite in our experimental results. The motor energy is

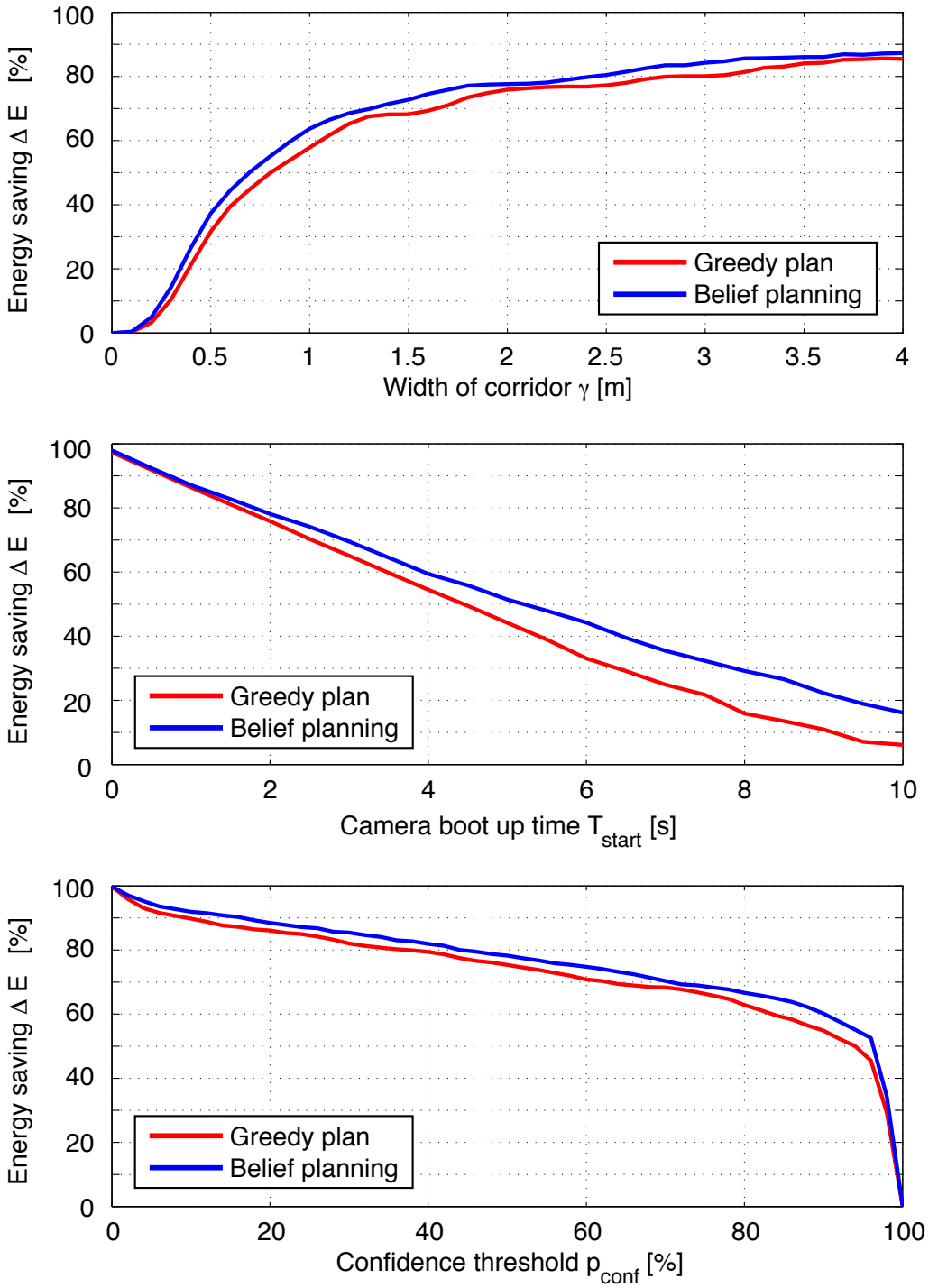


Figure 5.11: Perception energy savings ΔE_P as a function of the corridor width γ, the camera booting up time T_{start} and the confidence threshold p_{conf}. Increasing the confidence threshold as well as tightening the corridor or decreasing the camera booting time, produces a noticeable decrease of the energy savings for both algorithms.

slightly lower when following the perception schedules. We explain this by the fact that the robot performed fewer feedback corrections when the localisation system was turned off.

It is also interesting to observe that even though the second and the third trajectory have similar length, the energy consumed by the motors is higher for the second trajectory. We attribute this to the difference in terrain (grass vs. gravel paths), and the more complex trajectory. This requires the camera to be switched on for longer for both algorithms.

Finally, we performed a simulation-based analysis of the behaviour of the planning algorithms when varying key system parameters. Figure 5.11 shows the proportion of the perception-related energy saved, ΔE_P, after changing the allowed corridor width γ, the time necessary for the localisation system to boot T_{start}, and the confidence threshold p_{conf} for the first trajectory. Increasing the confidence threshold as well as tightening the corridor or decreasing the boot time produces a noticeable decrease in energy savings for both algorithms. When increasing the camera boot time, the *greedy* algorithm performs progressively worse than the optimal *belief planning* algorithm. This is attributed to the fact that longer boot times have a bigger impact on the schedule for even slightly complex trajectories.

5.6 Related Works

Optimising robotics energy consumption was explored in a number of different scenarios. The bulk of this prior art addresses the energetic cost of robot locomotion. A number of works in this vein consider path length, trajectory shape as well as the properties of different type of terrain to predict expected energy use (e.g. [87, 42, 14, 58]). Notably, in a simulation study, the authors of [29] do so in the context of planetary rovers. Mei et al. [61, 63] analyse the impact of different velocities, providing a control algorithm aimed at maximising the distance travelled. Motor dynamics and control input are considered in

[48].

The problem of planning and acting in partially observable stochastic domains was previously explored in [44] and various approximate solutions exists, such as [79]. To the best of our knowledge, the idea of scheduling a robot's perception system in order to save energy in practical terms is largely unexplored. The notion that motor energy – next to sensing and computation – may only account for a fraction of the energy requirements of a platform is demonstrated in [62]. This study also notes that the energy consumption of the sensors is a function of the frequency with which they are deployed. Brateman et al. [13] build on these findings and note that energy can be saved by regulating the clock speed of the processor used for perception computation. Based on this insight, in a simulation study they develop an algorithm which trades off vehicle velocity against processor clock speed.

Our work is closely related in spirit to both [62] and [13]. In contrast to these works we explicitly develop trajectory-dependent activity schedules for the perception subsystem. In addition, our methods are evaluated using real-world experiments. In order to provide optimal schedules, we frame this problem as a belief space planning task, which has a successful track record in mobile robotics in dealing with uncertainty in robot state (e.g. [47, 76, 95]).

5.7 Conclusions

This chapter explores the problem of planning in a partially observable environment using the framework of partially observable Markov decision process. Specifically, we focus on the efficacy of scheduling robot perception in order to save energy when following a planned path, i.e. trading information for energy. This is fundamentally different to the more common approach of minimising motor energy by altering the robot's trajectory. Our perception scheduling problem is framed as a belief space planning task which ex-

plicitly accounts for localisation start-up cost as well as pose uncertainty while the main localisation system is turned off. This allows for both optimal schedules to be computed and performance guarantees to be provided regarding the allowable deviation from the intended trajectory. Using 1.8km of autonomous traversals, our perception schedules are demonstrated to save a maximum of 58% of the perception energy, which accounts for 11.5% of the total robot energy expenditure. This is in comparison to a simpler greedy approach, which saves 5.5% less energy on perception on average than the optimal solution based on dynamic programming. This is a function of trajectory shape as well as the robotic platform used. A more in-depth characterisation of this dependency, together with intelligent detection and handling of situations if the robot exits the safe corridor, is subject to further work. In all scenarios greater accuracy can be obtained with more precise models of robot drift and energy usage. Other avenues for further investigation include an intelligent joint optimisation of robot trajectory and perception schedule in order to achieve even higher energy savings. This will need to be balanced with our surprising experimental observation that the reduced feedback from when the perception system is disabled can result in lower motor energy consumption.

Chapter 6

Conclusions

In this chapter we summarise the work described in the thesis and discuss some possible avenues for future work.

6.1 Summary of Contributions

In this thesis we have considered the limitations of applying Markov models to challenging robotics problems. In particular we have looked at three problem domains where application of this framework is considered difficult, and advanced the state-of-the-art by designing novel and practical solutions to them:

1. **Learning the observation and transition models from scratch in partially observable environments** (Chapter 3). We have described a new method based on using recurrent neural networks for training an internal model of the world in a partially observable environments, using only raw sensor observations $\mathcal{Z}$ in a form of a Hidden Markov Model (HMM). We have demonstrated that this method leads to a completely new paradigm for object tracking and data-efficient learning from supervised and unsupervised data.

2. **Evaluating inferred behaviours in large state-spaces** (Chapter 4). We have pre-

sented a new method for learning and evaluating user-specific driving behaviour using a Markov Decision Process (MDP) to predict the driver-specific range. This opens up new possibilities to achieve highly accurate range predictions, mitigating *range anxiety*, where drivers worry about the battery in their electric vehicle running out. We have formulated this task in terms of inferring an unknown driver reward function $\mathcal{R}$, vehicle energy consumption E and we have provided a new probabilistic inference method which is both fast and accurate.

3. **Planning to minimise perception** (Chapter 5). We have designed a fast belief-planning algorithm to compute the optimal sensor schedule for a robot traversing any trajectory while saving the maximum amount of sensing energy in a Partially Observable Markov Decision Process (POMDP). Simply turning off sensors at well-calculated moments results in significant energy savings in energy-critical applications such as planetary rover missions.

6.2 Future Work

There are a wide variety of avenues for developing the work described in this thesis; some particular areas of interest include:

6.2.1 Tracking in Partially Observable Environments

As discussed in Chapter 3, there are many ways to extend the presented DeepTracking method:

- **POMDP extension.** DeepTracking is formulated in the context of a hidden Markov model where the neural network learns the dynamics $\mathcal{T} : \mathcal{S} \times \mathcal{Z} \rightarrow \mathcal{S}$ of a system from raw observations. A simple extension is to consider the robot's decisions and learn the model $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{Z} \rightarrow \mathcal{S}$. This would allow us to learn the impact of

robot's actions on the environment.

- **Different sensor modalities.** DeepTracking has been demonstrated only on scenarios involving a planar 1D laser sensor. We believe that similar techniques could be used in 3D with depth cameras or multiple sensors.

- **Visibility ambiguity.** We believe that the presented method can be used in a wide variety of problems in which information about the visible/invisible portions of the space are explicitly given. More work is required to address problems where this information is not readily available, such as the case of RGB cameras.

- **Knowledge transfer.** We have demonstrated that a principle of knowledge transfer can be used to make the trained system perform well in semantic segmentation, which is a different but related scenario. This approach could be extended to produce different kinds of information, such as bounding box positions, or in a more general setting and in combination with deep inverse reinforcement learning, the value of the state as described below in Section 6.2.4.

- **New network architectures.** The presented neural network performs well for the considered scenario with a static sensor and moving objects. A more appropriate network is desirable to address scenarios in which robot moving through environment.

6.2.2 Learning and Evaluating the Policy

The topic presented in Chapter 4 can be extended in a number of ways:

- **Neural network to represent state reward**. We have considered only simple linear models for learning both the reward function $\mathcal{R}_{s,a}$ and consumed energy $E_{s,a}$. A more robust model in the form of a neural network could be used to learn complex non-linear approximations directly from the data, achieving higher complexity.

- **Transfer of knowledge.** As described in Section 6.2.4, the reward function can be learned on top of a pre-trained model similarly to learning semantic segmentation on top of a tracking network. This results in a higher data-efficiency in learning. It is as well a safer learning way to, for example, learning from scratch using reinforcement learning as there learning is achieved through continuous process of trials and errors.

6.2.3 Planning in Partially Observable Environments

The method of planning in partially observable environments presented in Chapter 5 offers several possibilities for further work:

- **Different application domains.** We believe that the presented method can be used for another trajectory following scenario, for example those involving robotic manipulation, or optimising a trade-off between mobile and cloud processing as demonstrated in [56].

- **Uncertainty in neural networks**. We have used a simple particle filter to represent the state of the system and corresponding uncertainty. A neural network could be used instead similarly to Chapter 3. To do this a mechanism for estimating neural network uncertainty would need to be developed first. The work of [35] provides inspiration for a possible approach.

6.2.4 A Full Neural Pipeline

In this final section we briefly outline the possibility of combining topics discussed in the thesis to construct a new kind of robotics pipeline, leading to potentially better and more powerful robots. The basic idea is to combine DeepTracking (Chapter 3), inverse reinforcement learning from (Chapter 4) and the plan-generation algorithm into a single pipeline which learns its functionality directly from the data (as shown in Figure 6.1).

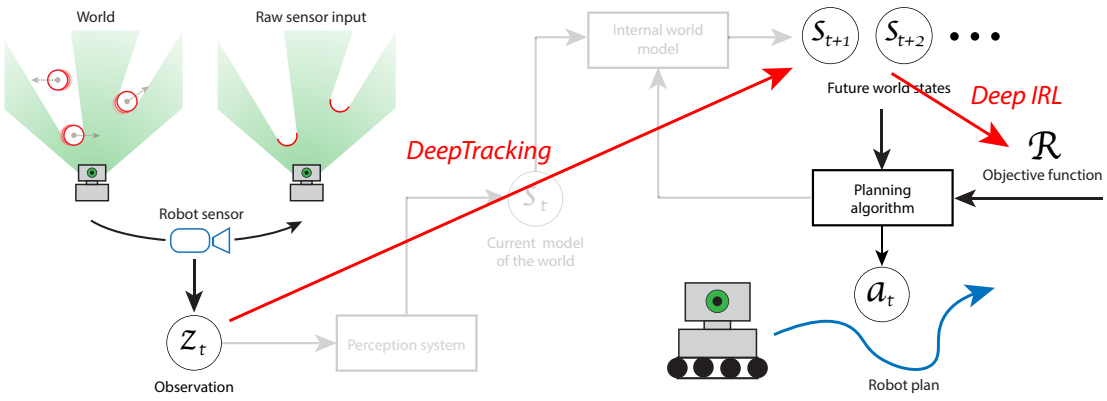


Figure 6.1: The proposed neural robotics pipeline compared to the classical robotics pipeline. DeepTracking is used to map raw sensor observations directly to un-occluded future world states. Deep Inverse Reinforcement Learning (IRL) is then used to distinguish desirable ones.

In this paradigm, unlike in reinforcement learning approaches to robotics, the training of such a pipeline is safe as it does not involve a robot repetitively trying to succeed in a particular task, but rather it observes world and an expert behaviour.

We propose to use the DeepTracking algorithm described in Chapter 3 as the world model simulator. DeepTracking learns to act as an oracle that can predict the future world states. Such predictions effectively act as a world simulator capable of evaluating any possible robot plan, and thus providing the robot with a simple yet effective planning paradigm.

Next, to derive robot actions, we aim to generate a large number of feasible kinematic plans for a robot to perform. This can be done by either an exhaustive generation of all possible sequences of control signals, or by using more elaborate methods. These plans would in turn be evaluated using the DeepTracking network, whereby the network would generate a sequence of future world states $s_{t+1}, s_{t+2}, s_{t+3}, ...$ when following a given plan of actions. Finally, the cost of each sequence would be computed, thus capturing the feasibility and value of the plan, and determining its ranking. The entire system is shown in Figure 6.2.

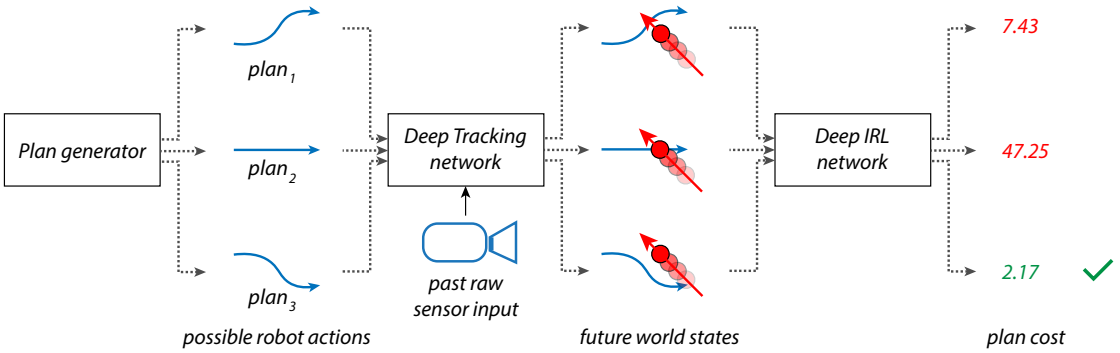


Figure 6.2: The proposed neural robotics pipeline. The plan generator synthesises a set of possible plans of actions. The DeepTracking network acts as an oracle capable of predicting the future world states in response to any given plan of actions. The plan with the lowest cost as evaluated by the deep Inverse Reinforcement Learning (IRL) network is then selected for execution.

Finally, we propose using *deep Inverse Reinforcement Learning* (IRL) [102] for evaluating the different plans of actions as described above. Using deep IRL would avoid the need for designing such an evaluation function by hand. Instead, a separate neural network would be specifically trained to infer the desired objective function using a supervised data set of demonstrations, where a human in the charge of the control signal.

The advantage of our proposed robotics pipeline design is that the *DeepTracking* network can be trained using large amounts of purely unsupervised data to learn *how the world works* and how to remove occlusions. Then, the network for *deep IRL* can exploit the outputs of the DeepTracking framework to be trained only how to identify *which world states are desirable*, without considering occlusions. This is a much simpler task which can be learnt using relatively little supervised data, as we have demonstrated in Chapter 3.

We believe that this approach would allow the robot to infer optimal procedures for sensing, internal world model updates and planning directly from the sensory data. The main benefit of our approach is that it reduces the amount of hand-engineering and prior knowledge injected into the pipeline, thus removing the constraints that come with them.

6.3 Concluding Remark

In this thesis we have considered Markov models in the context of robotics problems. Markov models offer a sound theoretical framework applicable to a range of robotics problems, but lack efficiency in many cases, some of which we explore in this thesis. For those, we have provided novel solutions that overcome these limitations. We hope that this work will be valuable for future research and development in the fields of robotics, applied machine learning and artificial intelligence, in the quest to develop capable, intelligent and useful robots in our future.

Bibliography

[1] Full electric vehicles. Technical Report AN-1564, Allied Business Intelligence (ABI) Research, Intelligent Transportation Systems Technology Analysis, 2013.

[2] Pieter Abbeel, Dmitri Dolgov, Andrew Y Ng, and Sebastian Thrun. Apprenticeship learning for motion planning with application to parking lot navigation. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1083–1090. IEEE, 2008.

[3] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

[4] Raymond E. Arvidson. Planetary science decadal survey: Mars 2018 MAX-C caching rover. Technical report, National Aeronautics and Space Administration (NASA), March 2010.

[5] Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

[6] Richard Bellman. A markovian decision process. Technical report, DTIC Document, 1957.

[7] Yoshua Bengio. Learning deep architectures for AI. *Foundations and trends in Machine Learning*, 2(1):1–127, 2009.

[8] Niclas Bergman. Recursive bayesian estimation. *Department of Electrical Engineering, Linköping University, Linköping Studies in Science and Technology. Doctoral dissertation*, 579, 1999.

[9] Dimitri P Bertsekas. *Dynamic programming and optimal control*. Athena Scientific Belmont, MA, 2012.

[10] Jeffrey J Biesiadecki, P Chris Leger, and Mark W Maimone. Tradeoffs between directed and autonomous driving on the mars exploration rovers. *The International Journal of Robotics Research*, 26(1):91–104, 2007.

[11] Byron Boots, Arunkumar Byravan, and Dieter Fox. Learning predictive models of a depth camera & manipulator from raw execution traces. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4021–4028. IEEE, 2014.

[12] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[13] Jeff Brateman, Changjiu Xian, and Yung-Hsiang Lu. Energy-effcient scheduling for autonomous mobile robots. In *Very Large Scale Integration, 2006 IFIP International Conference on*, pages 361–366. IEEE, 2006.

[14] John A Broderick, Dawn M Tilbury, and Ella M Atkins. Characterizing energy usage of a commercially available ground robot: Method and results. *Journal of Field Robotics*, 31(3):441–454, 2014.

[15] Richard H Byrd, Jean Charles Gilbert, and Jorge Nocedal. A trust region method based on interior point techniques for nonlinear programming. *Mathematical Programming*, 89(1):149–185, 2000.

[16] Rich Caruana. Learning many related tasks at the same time with backpropagation. *Advances in neural information processing systems*, pages 657–664, 1995.

[17] M. Ceraolo and G. Pede. Techniques for estimating the residual range of an electric vehicle. *IEEE Transactions on Vehicular Technology*, 50(1), 2001.

[18] Mohammad Charkhgard and Mohammad Farrokhi. State-of-charge estimation for lithium-ion batteries using neural networks and ekf. *Industrial Electronics, IEEE Transactions on*, 57(12):4178–4187, 2010.

[19] Y. Chen, C. Huang, Y. Kuo, and S. Wang. Artificial neural network for predictions of vehicle drivable range and period. In *Proceedings of the IEEE International Conference on Vehicular Electronics and Safety*, 2012.

[20] Zhe Chen. Bayesian filtering: From Kalman filters to particle filters, and beyond. *Statistics*, 182(1):1–69, 2003.

[21] Sonia Chernova and Manuela Veloso. Confidence-based policy learning from demonstration using gaussian mixture models. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 233. ACM, 2007.

[22] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[23] National Research Council Committee on the Planetary Science Decadal Survey. *Vision and Voyages for Planetary Science in the Decade 2013-2022*. National Academies Press, Washington, D.C., 2011.

[24] George E Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):30–42, 2012.

[25] Frank Deinzer, Joachim Denzler, and Heinrich Niemann. Viewpoint selection–planning optimal sequences of views for object recognition. In *International Conference on Computer Analysis of Images and Patterns*, pages 65–73. Springer, 2003.

[26] Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Monte carlo localization for mobile robots. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 2, pages 1322–1328. IEEE, 1999.

[27] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[28] Alberto Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[29] Saber Fallah, Bonnie Yue, Orang Vahid-Araghi, and Amir Khajepour. Energy management of planetary rovers using a fast feature-based path planning and hardware-in-the-loop experiments. *Vehicular Technology, IEEE Transactions on*, 62(6):2389–2401, 2013.

[30] Jialue Fan, Wei Xu, Ying Wu, and Yihong Gong. Human tracking using convolutional neural networks. *Neural Networks, IEEE Transactions on*, 21(10):1610–1623, 2010.

[31] João C Ferreira, Vítor Monteiro, and João L Afonso. Data mining approach for range prediction of electric vehicle. 2012.

[32] Katerina Fragkiadaki, Sergey Levine, Panna Felsen, and Jitendra Malik. Recurrent network models for human dynamics. ICCV, 2015.

[33] J. Froehlich and J. Krumm. Route prediction from trip observations. *Society of Automotive Engieers Special Publication (SAE SP)*, 2193:53, 2008.

[34] Paul T. Furgale and Timothy D. Barfoot. Visual teach and repeat for long-range rover autonomy. *Journal of Field Robotics, Special issue on "Visual mapping and navigation outdoors"*, 27(5):534–560, May 2010.

[35] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *arXiv preprint arXiv:1506.02142*, 2015.

[36] Jean Dickinson Gibbons and Subhabrata Chakraborti. *Nonparametric statistical inference*. Springer, 2011.

[37] H. Gonzalez, J. Han, X. Li, M. Myslinska, and J. P. Sondag. Adaptive fastest path computation on a road network: a traffic mining approach. In *Proceedings of the 33rd international conference on Very large data bases*, pages 794–805. VLDB Endowment, 2007.

[38] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[39] L. Guzzella and A. Sciarretta. *Vehicle propulsion systems*. Springer, 2007.

[40] M. Haklay and P. Weber. Openstreetmap: User-generated street maps. *Pervasive Computing, IEEE*, 7(4):12–18, 2008.

[41] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[42] Genya Ishigami, Keiji Nagatani, and Kazuya Yoshida. Path planning and evaluation for planetary rovers based on dynamic mobility index. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 601–606. IEEE, 2011.

[43] Uwe Jänen, Christian Paul, Michael Wittke, and Jörg Hähner. Multi-object tracking using feed-forward neural networks. In *Soft Computing and Pattern Recognition (SoCPaR), 2010 International Conference of*, pages 176–181. IEEE, 2010.

[44] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1):99–134, 1998.

[45] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of Fluids Engineering*, 82(1):35–45, 1960.

[46] D. Karbowski, S. Pagerit, and A. Calkins. Energy consumption prediction of a vehicle along a user-specified real-world trip. In *Proceedings from the Electric Vehicle Symposium (EVS)*, 2012.

[47] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, 1996.

[48] Chong Hui Kim and Byung Kook Kim. Energy-saving 3-step velocity control algorithm for battery-powered wheeled mobile robots. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 2375–2380. IEEE, 2005.

[49] E. Kim, J. Lee, and K. G. Shin. Real-time prediction of battery power requirements for electric vehicles. In *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems (ICCPS)*, 2013.

[50] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 2013.

[51] T. Kono, T. Fushiki, K. Asada, and K. Nakano. Fuel consumption analysis and prediction model for eco route search. In *Proceedings of the 15th World Congress on Intelligent Transport Systems and ITS America's 2008 Annual Meeting*, 2008.

[52] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[53] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[54] Quoc V Le. Building high-level features using large scale unsupervised learning. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8595–8598. IEEE, 2013.

[55] Seong Jae Lee and Zoran Popović. Learning behavior styles with inverse reinforcement learning. In *ACM Transactions on Graphics (TOG)*, volume 29, page 122. ACM, 2010.

[56] Alex Locher, Michael Perdoch, Hayko Riemenschneider, and Luc Van Gool. Mobile phone and cloud - a dream team for 3d reconstruction. In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–8. IEEE, 2016.

[57] Tomás Lozano-Perez, Ingemar J Cox, and Gordon T Wilfong. *Autonomous robot vehicles*. Springer Science & Business Media, 2012.

[58] Steven Martin and Peter Corke. Long-term exploration and tours for energy constrained robots with online proprioceptive traversability estimation. In *Proceedings*

of the 2014 IEEE International Conference on Robotics and Automation (ICRA), pages 5778–5785. IEEE, 2014.

[59] C McManus, P T Furgale, B Stenning, and T D Barfoot. Lighting-invariant visual teach and repeat using appearance-based lidar. *Journal of Field Robotics*, 30(2):254–287, 2013.

[60] LR Medsker and LC Jain. Recurrent neural networks. *Design and Applications*, 2001.

[61] Yongguo Mei, Yung-Hsiang Lu, Y Charlie Hu, and CS George Lee. Energy-efficient motion planning for mobile robots. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 5, pages 4344–4349. IEEE, 2004.

[62] Yongguo Mei, Yung-Hsiang Lu, Y Charlie Hu, and CS George Lee. A case study of mobile robot's energy consumption and conservation techniques. In *Advanced Robotics, 2005. ICAR'05. Proceedings., 12th International Conference on*, pages 492–497. IEEE, 2005.

[63] Yongguo Mei, Yung-Hsiang Lu, Yu Charlie Hu, and CS George Lee. Deployment of mobile robots with energy and timing constraints. *Robotics, IEEE Transactions on*, 22(3):507–522, 2006.

[64] Christoph Mertz, Luis E Navarro-Serment, Robert MacLachlan, Paul Rybski, Aaron Steinfeld, Arne Suppe, Christopher Urmson, Nicolas Vandapel, Martial Hebert, Chuck Thorpe, et al. Moving object detection with laser scanners. *Journal of Field Robotics*, 30(1):17–43, 2013.

[65] Claire F Minett, AM Salomons, Winnie Daamen, Bart Van Arem, and Sjon Kuijpers. Eco-routing: comparing the fuel consumption of different routes between an

origin and destination using field test speed profiles and synthetic speed profiles. In *Integrated and Sustainable Transportation System (FISTS), 2011 IEEE Forum on*, pages 32–39. IEEE, 2011.

[66] Tom M Mitchell, Sebastian B Thrun, et al. Explanation-based neural network learning for robot control. *Advances in neural information processing systems*, pages 287–287, 1993.

[67] Roni Mittelman, Benjamin Kuipers, Silvio Savarese, and Honglak Lee. Structured recurrent temporal restricted boltzmann machines. In *Proceedings of the 31st International Conference on Machine Learning*, pages 1647–1655, 2014.

[68] George E Monahan. State of the art—a survey of partially observable markov decision processes: theory, models, and algorithms. *Management Science*, 28(1):1–16, 1982.

[69] P. Newson and J. Krumm. Hidden markov map matching through noise and sparseness. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS)*, pages 336–343, 2009.

[70] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, pages 663–670, 2000.

[71] M. Nilsson. Electric vehicles: The phenomenon of range anxiety. In *Report for the ELVIRE Project (FP7 PROJECT ID: ICT-2009.6.1)*, 2011.

[72] Javier A Oliva, Christoph Weihrauch, and Torsten Bertram. A model-based approach for predicting the remaining driving range in electric vehicles. In *Annual conference of the prognostics and health management society*, pages 438–448, 2013.

[73] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *Knowledge and Data Engineering, IEEE Transactions on*, 22(10):1345–1359, 2010.

[74] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *arXiv preprint arXiv:1211.5063*, 2012.

[75] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, 12:1532–1543, 2014.

[76] Samuel Prentice and Nicholas Roy. The belief roadmap: Efficient planning in belief space by factoring the covariance. *The International Journal of Robotics Research*, 2009.

[77] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[78] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[79] Stéphane Ross, Joelle Pineau, Sébastien Paquet, and Brahim Chaib-Draa. Online planning algorithms for pomdps. *Journal of Artificial Intelligence Research*, 32:663–704, 2008.

[80] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[81] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5:3, 1988.

[82] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach (3rd ed.)*. Prentice Hall, 2009.

[83] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. *arXiv preprint arXiv:1506.04214*, 2015.

[84] Patrice Y Simard, David Steinkraus, and John C Platt. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, volume 3, pages 958–962, 2003.

[85] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[86] Le Song, Byron Boots, Sajid M. Siddiqi, Goeffrey J. Gordon, and Alex Smola. Hilbert space embeddings of hidden markov models. In *Proceedings of the 26 International Conference on Machine Learning*. IEEE, 2010.

[87] Zheng Sun and John Reif. On energy-minimizing paths on terrains for a mobile robot. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 3, pages 3782–3788. IEEE, 2003.

[88] Ilya Sutskever and Geoffrey E Hinton. Learning multilevel distributed representations for high-dimensional sequences. In *International Conference on Artificial Intelligence and Statistics*, pages 548–555, 2007.

[89] Ilya Sutskever, Geoffrey E Hinton, and Graham W Taylor. The recurrent temporal restricted boltzmann machine. In *Advances in Neural Information Processing Systems*, pages 1601–1608, 2009.

[90] R. Sutton and A. Barto. *Reinforcement learning: An introduction*, volume 1. Cambridge Univ Press, 1998.

[91] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep neural networks for object detection. In *Advances in Neural Information Processing Systems*, pages 2553–2561, 2013.

[92] Sebastian Thrun and Arno Bücken. Integrating grid-based and topological maps for mobile robot navigation. In *Proceedings of the National Conference on Artificial Intelligence*, pages 944–951, 1996.

[93] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[94] M. Treiber, A. Hennecke, and D. Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical Review E 62 (2)*, pages 1805–1824, 2000.

[95] Jur Van Den Berg, Sachin Patil, and Ron Alterovitz. Motion planning under uncertainty using iterative local optimization in belief space. *The International Journal of Robotics Research*, 31(11):1263–1278, 2012.

[96] Javier Velez, Garrett Hemann, Albert S Huang, Ingmar Posner, and Nicholas Roy. Planning to perceive: Exploiting mobility for robust object detection. In *ICAPS*, 2011.

[97] Javier Velez, Garrett Hemann, Albert S Huang, Ingmar Posner, and Nicholas Roy. Modelling observation correlations for active exploration and robust object detection. *Journal of Artificial Intelligence Research*, 2012.

[98] A. Vogel, D. Ramachandran, R. Gupta, and A. Raux. Improving hybrid vehicle fuel efficiency using inverse reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2012.

[99] Dominic Zeng Wang, Ingmar Posner, and Paul Newman. Model-free detection and tracking of dynamic objects with 2D lidar. *The International Journal of Robotics Research*, 34(7):1039–1063, 2015.

[100] Tao Wang, David J Wu, Andrew Coates, and Andrew Y Ng. End-to-end text recognition with convolutional neural networks. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 3304–3308. IEEE, 2012.

[101] Douglas J White. Real applications of Markov decision processes. *Interfaces*, 15(6):73–83, 1985.

[102] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*, 2015.

[103] SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun WOO. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in Neural Information Processing Systems*, pages 802–810, 2015.

[104] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *Acm computing surveys (CSUR)*, 38(4):13, 2006.

[105] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[106] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W. Ma. Understanding mobility based on GPS data. In *Proceedings of the 10th International Conference on Ubiquitous Computing*, pages 312–321, 2008.

[107] Y. Zheng, X. Xie, and W. Ma. Geolife: A collaborative social networking service among user, location and trajectory. *IEEE Data Engineering Bulletin*, 33(2):32–39, 2010.

[108] Y. Zheng, L. Zhang, X. Xie, and W. Ma. Mining interesting locations and travel sequences from GPS trajectories. In *Proceedings of the 18th International Conference on World Wide Web*, pages 791–800, 2009.

[109] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 1433–1438, 2008.