Predictive Sensing for Field Robotics

Julie Dequaire

Pembroke College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Hilary 2019

Abstract

The global autonomous robot market is expected to be worth more than eleven billion US dollars by 2024, with a need - across industries - for autonomous robots able to operate safely in complex and dynamic real-world environments. In this thesis, we argue that this will require developing robots able to develop situational awareness by constructing a higher level understanding of the world beyond the instantaneous and limited data provided by their sensors. We suggest this can be achieved by enabling robots to *learn* directly from their environment to predict the future evolution of the world and performance of their *systems*.

Robust mobile autonomy is developed around the three pillars of navigation, perception, and planning. For an agent to effectively plan a route through its environment, it needs to know where it is in its environment and what is going on around it. To do so, it is equipped with localisation and perception systems which can interpret incoming data from onboard sensors. Localisation systems typically provide instantaneous information with regards to whether an agent is localised or lost. Perception systems typically tracks objects in the environment using multi-stage pipelines which operate independently, require much hand-engineering, and show little robustness to natural occlusions. These low-level interpretations of incoming data often contribute to poor situational awareness and make it difficult to plan ahead with far-sightedness.

In this thesis we address both shortcomings and propose data-driven approaches to increase situational awareness for a localisation and perception system operating in the field. In the area of navigation, we propose a novel framework for predicting ahead of time how well a robot will be able to localise in a given environment given an appearance model of the scene. In the area of perception, we extend an end-to-end deep-learning framework for predicting near-future scene occupancy beyond natural occlusions, to operate in the real world, from a moving platform, and taking into consideration the scene context.

In doing so, we contribute to developing greater situational awareness for robotic systems operating in real world environments. We argue that robotic systems can make greater sense of what their are perceiving by moving away from *instant sensing* to *predictive sensing*.

Predictive Sensing for Field Robotics

Julie Dequaire

Pembroke College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Supervisor: Prof. Ingmar Posner

Hilary 2019

To mum and dad,

Per ardua ad astra

"Humans are not disabled. A person can never be broken. Our built environment, our technologies are broken and disabled."

— Prof. Hugh Herr, MIT Media Lab.

Acknowledgements

First and foremost I would like to thank my supervisor Prof. Ingmar Posner for his endless enthusiasm for robot learning and great patience in guiding my wondering mind throughout these years. Thank you for leading me through peaks of research joys and supporting me through unexpected troughs of the journey.

Many thanks to Prof. Paul Newman for bringing me onboard the MRG adventure and along with Dr. Nicola Bellotto for interesting reflections at the viva. To my many ORI colleagues and friends for enlightening discussions, white board inspirations, late night coding, and greats times in and out of the lab. Particular thoughts go to Corina Gurau, Hillary Shakespeare, Matt Gadd, Ștefan Săftescu, Dan Barnes, Hugo Grimmett and Rosemary Cameron.

I am grateful to have worked with, co-authored and learned much from Peter Ondrúška; "what is it that you are trying to achieve?" has avoided me many pitfalls and remains at the forefront of my research mind today. Thanks to my coauthors Dushyant Rao, Chi Hay Tong, Dominic Zeng Wang, and Winston Churchill for their breadth of knowledge and thought-provoking suggestions.

This thesis would not have happened had I not been inspired to pursue research by my previous supervisors at NASA Ames, Dr. Melinda Kahre and Dr. Robert Haberle. Many have kept me sane along the way: Christelle Alvarez, Valeria Rueda and Sarah Houillon, the Foxes in and out of the pitch, members of Bicester and Turweston flying clubs for new horizons, and Delta Notch's 8 hour study mix when writing up.

To Emmanuel, Myriam, mamie Simone and papi Henri.

To the Dequaires, Sug, Lou, Adri and mamie chouchou for supporting me throughout and papounet and mamounette for their love, guidance, resilience and courage. Rovers navigate Mars, yet individuals are bound to century-old wheelchairs. Technology must also be developed to empower the individual.

Lastly, to Tom, my co-pilot, with whom everything

Abstract

The global autonomous robot market is expected to be worth more than eleven billion US dollars by 2024, with a need - across industries - for autonomous robots able to operate safely in complex and dynamic real-world environments. In this thesis, we argue that this will require developing robots able to develop situational awareness by constructing a higher level understanding of the world beyond the instantaneous and limited data provided by their sensors. We suggest this can be achieved by enabling robots to *learn* directly from their environment to predict the future evolution of the world and performance of their *systems*.

Robust mobile autonomy is developed around the three pillars of navigation, perception, and planning. For an agent to effectively plan a route through its environment, it needs to know where it is in its environment and what is going on around it. To do so, it is equipped with localisation and perception systems which can interpret incoming data from onboard sensors. Localisation systems typically provide instantaneous information with regards to whether an agent is localised or lost. Perception systems typically tracks objects in the environment using multi-stage pipelines which operate independently, require much hand-engineering, and show little robustness to natural occlusions. These low-level interpretations of incoming data often contribute to poor situational awareness and make it difficult to plan ahead with far-sightedness.

In this thesis we address both shortcomings and propose data-driven approaches to increase situational awareness for a localisation and perception system operating in the field. In the area of navigation, we propose a novel framework for predicting ahead of time how well a robot will be able to localise in a given environment given an appearance model of the scene. In the area of perception, we extend an end-to-end deep-learning framework for predicting near-future scene occupancy beyond natural occlusions, to operate in the real world, from a moving platform, and taking into consideration the scene context.

In doing so, we contribute to developing greater situational awareness for robotic systems operating in real world environments. We argue that robotic systems can make greater sense of what their are perceiving by moving away from *instant sensing* to *predictive sensing*.

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.

Julie Dequaire, Pembroke College

Funding

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

Contents

List of Figures

List of Tables

1 Introduction

The global autonomous robot market is expected to nearly triple in value between 2016 and 2024 to reach more than 11,920 million US dollars, with a demand for autonomous robots expected to grow at around 13% during that period[1]. Amongst the leading drivers of this increasing market are the automotive, mining, maritime, and aerospace and defence industries which will all require mobile robots able to autonomously and safely navigate dynamic, complex, and real-world environments. Pivotal to mobile autonomy is the ability for robots to predict the future evolution of the world and performance of their systems, as reflected by the wide body of work relating to object tracking and planning under system constraints. To meet the challenge, we argue that robots will need to be able to *learn* from their environment and construct a higher-level understanding of the world beyond the instantaneous and limited data provided by their sensors. This can be achieved by learning to predict what systems are about to sense or estimate, rather than rely on instantaneous sensor information and system interpretations. If robots are to develop true *situational awareness*, there is a need for robotic systems to make greater sense of what they are perceiving, and move from *instant sensing* to *predictive sensing*.

[1] Zion Market Research estimate from March 2018

To address the need for increasingly capable robots, the field robotics community will have to develop robust *mobile autonomy* constructed around the three pillars of *navigation*, *perception*, and *planning*. For an agent to effectively navigate its environment autonomously towards a goal, it needs to continuously know *where it is* in its environment, *what is going on* around it, and *what to do next*. Notably, if the agent is lost or cannot perceive, it will be difficult if not impossible to plan further.

To navigate and perceive the world, robots are equipped with localisation and perception *systems* which themselves rely on information provided by sensors such as cameras and range scanners. Though elaborate algorithms can be devised to interpret the incoming data, these systems are often limited by the instantaneous and local information provided by their sensors. This often results in low-level interpretations of their surroundings, and poor *situational awareness*. A localisation system typically provides instantaneous information with regards to its ability or failure to localise in the environment. At every iteration, it detects features or patches in the incoming data stream which it attempts to match against a memory. If the matching fails, the robot is lost. This can be very costly and limiting if a robot is to explore the surface of Mars or underwater environments on Earth. A perception system typically tracks objects in the environment using elaborate multi-stage pipelines consisting in object detection, data association, forward prediction and filtering over time. These pipeline modules tend to process data on a *local* temporal level and are developed independently which results in little robustness to natural object occlusions in the data, and much hand-engineering.

Low-level interpretations of sensor data renders navigating real world environments particularly challenging for robots. The world state is dynamic, changing, diverse, and cannot be fully observed. On a local level particularly, the world never looks exactly the same twice. The appearance of objects and places changes with weather, season, time of day, and motion of both objects and robot. Dynamic objects can exhibit an array of non-linear dynamics resulting from their *state*, e.g what and where they are, what their intentions are, and what is going on around them. At a higher level however, the world exhibits a lot of structure. Buildings

and trees do not tend to move very often, and urban areas are governed by very context-dependant motion patterns.

This thesis argues that truly capable robots in the field will need to be equipped with systems able to capture this higher level structure if they are to autonomously navigate real-world environments in a safe and reliable way. They need to be able to build *situational awareness* by constructing long-sighted interpretations of the data provided by their sensors. Rather than limiting themselves to providing instant information that is disconnected from a greater understanding of the environment, we suggest they can learn a higher level model of their sensing systems directly from experience, by learning to *infer* future observations or predictions of these systems.

We advocate moving away from instant sensing, to *predictive* sensing, and suggest that localisation and perception systems need to be able to learn *from experience* and estimate *a priori* the evolution of the state of the environment and their ability to operate in it. In the context of field robotics, we are interested in data driven approaches for modelling real world unknown processes and performing inference. We turn towards Gaussian Processes as a non-parametric solution, as well as neural networks for their capacity to approximate complex functions.

In the context of navigation, we first investigate how a vision-based localisation system can learn to estimate *ahead of time* how it is going to perform in the world, rather than provide instantaneous localisation information. We use Gaussian Processes and learn a latent distribution over possible functions to predict future localisation performance based on the appearance of the scene we wish to localise against. In the context of perception, we investigate how a tracking system can estimate the underlying state of the world and predict future unoccluded scene occupancy. We leverage the Dynamic Bayes framework, and use recurrent neural networks to approximate a distribution over belief states in real world environments. This estimate can be used to infer future observations, handle vehicle ego-motion, and learn to track objects according to scene layout.

We detail our contributions and thesis structure in the following sections.

1.1 Contributions

Improving a Localisation System's Situational Awareness by Predicting Localisation Performance

In Chapter 3, we first address limitations of a *localisation* module and motivate the importance of developing a robot's capacity to predict *a priori* its ability to localise in a given environment. We consider the visual *Teach and Repeat* [Furgale and Barfoot, 2010] paradigm and develop a framework for a robot to estimate ahead of time how well it will be able to localise along the path it wishes to repeat, depending on the surrounding scene appearance. We use Gaussian Processes to predict the number of features likely to be matched by the localisation system against a particular map keyframe based on an appearance model of the keyframe as well as on factors such as intended path deviation and trajectory shape. The resulting localisation envelope is produced as a 2D cost map for use by a planner.

This contribution is important because in many areas of robotics, particularly where human-robot communication is delayed or absent, getting lost can incur a huge cost. This can be the case for a robot exploring the surface of Mars, Earth's oceans, or deep tunnels of a mine. Having a framework able to predict ahead of time where it is most likely to get lost therefore has value. From a different angle, a robot may want to explore areas where it expects its localisation ability to be poor, in order to gather more data and increase its knowledge of this area. Depending on the robot's goals, the planner could either decide to remain within high confidence localisation ability zones, or decide to explore outside of its comfort zone. Finally, for energy considerations, some routes may be preferred to others depending on localisation ability. The proposed approach can also readily be extended beyond the *Teach and Repeat* paradigm to other visual localisation frameworks such as localising in global overhead maps.

This body of work constitutes the introductory research chapter of this thesis and has featured in a conference publication [Dequaire et al., 2016]. At time of publication, to the best of our knowledge, our proposed system is the first able to predict localisation performance based on an appearance model of the scene. Since

publication, it has found to be relevant in work on system introspection for robot decision making, [Hu and Kantor, 2017], [Gurău et al., 2017], user trust [Israelsen and Ahmed, 2017], and improving map quality for localisation [Vysotska and Stachniss, 2017], [Martins, 2017].

We provide the following contributions:

- Development of an *appearance-based* approach to estimating the likely *localisation envelope* around a taught route in the context of Visual Teach and Repeat
- Demonstration on real-world offroad data that the extended system is able to *predict* this *localisation envelope* for trajectories situated in similar, yet geographically distant locations where no repeat runs have yet been performed
- Demonstration that the proposed solution performs as well as prior art when it comes to interpolating localisation performance based on a number of repeat runs.

.

Improving A Perception System's Situational Awareness by Developing Richer State Estimators

In Chapter 4, we address limitations of a *perception system*, and motivate the importance of developing frameworks able to learn rich state representations of real world environments beyond the limitation of a robot's sensors, and directly from data. We particularly focus on predicting future unoccluded scene occupancy from a time series of raw range sensor measurements. We build on *Deep Tracking* [Ondruska and Posner, 2016], a recurrent neural network adaptation of the Dynamic Bayes framework which aims to learn state representation, state transition and observation functions for Hidden Markov Models. Originally demonstrated on simple *synthetic* data, we motivate and investigate a number of architectural changes and extend the system to perform tracking on *real world* data collected at a busy urban intersection. We show increased ability to track into the future and through occlusions compared

to a state-of-the are tracking pipeline, with the additional benefit of offering an end-to-end self-supervised solution which can be trained in the absence of ground truth.

This is an interesting contribution as for systems to work in the real world, they need to be developed and tested outside of synthetic environments. By demonstrating improved tracking ability on a real world urban dataset we contribute a system that can be used and built upon in field robotics. Secondly, we find that this tracking framework could provide a useful and practical alternative to traditional planning cost maps. Occupancy grids often form a ready input for planning algorithms, who need to be able to see into the future and separate dynamic from static obstacles. Alternatively, the *Deep Tracking* framework produces and predicts unoccluded occupancy grids into the future, is learned end-to-end directly from data and can, by construct, be readily adapted and extended to various needs as we explore in further chapters. This makes it a very practical and versatile system for field robotics.

The work presented in this chapter was published in a journal issue [Dequaire et al., 2018] following a workshop publication where it received an RSS Best Workshop Paper award [Ondruska et al., 2016]. Since publication, it has found relevance in intuiting physics [Ehrhardt et al., 2017], multi-object tracking [Agarwal and Suryavanshi, 2017, Milan et al., 2017], and traffic prediction [Djuric et al., 2018].

We provide the following contributions:

- Development of the *Deep Tracking* framework to complex, dynamic and partially observable real-world scenarios. We evaluate a number of architectural changes and demonstrate the effectiveness of the system in recovering the unoccluded scene occupancy on a hour-long dataset collected from a static range scanner located at a busy urban intersection in the Oxford city centre, UK

- Demonstration that our system compares favourably to a state-of-the-art model-free tracking solution in terms of tracking through occlusion, and simplicity of implementation and deployment

- Elaborate architectural ablation studies to investigate the added value of the proposed model extensions. We demonstrate improved tracking ability through occlusion with a more elaborate memory state module as well as efficient increase of receptive field with dilated convolutions for capturing a range of object sizes and dynamics

- Analysis of the system's ability to use its rich state representation for learning further tasks in a data-efficient manner, such as producing scene semantics using Transfer of Knowledge. We introduce a static memory able to learn place specific information and demonstrate improved performance in the task of multi-class object semantic prediction as well as scene segmentation in terms of static and dynamic objects.

Extending State Estimation to a Moving Vehicle for Practical Real-World Deployment

In Chapter 5 we address the need to extend our framework to a moving platform. Our tracking system relies on a robot-centric occupancy grid representation which is spatially coherent with the world. Rather than maintaining a global grid representation in which the robot navigates, we adopt a local grid view that moves along with the robot. We extend our formulation to reflect the full Dynamic Bayes framework and incorporate vehicle ego-motion in our belief state update. We decouple ego-motion from world dynamics, and coherently transform the state representation to perform tracking from a moving platform inspired by the work of Jaderberg et al. [2015] on *Spatial Transformer Networks*. We demonstrate the effectiveness of our solution on synthetic data, and show improved tracking ability on urban real-world data collected from a vehicle.

We find this is an interesting development as first and foremost, systems that are to be used in real-world field robotics need to be deployable on moving vehicles. By extending our state estimator to function on a moving platform, we contribute to developing end-to-end, learnable, deep state estimators for use in real world settings. Part of the work presented in this chapter was published in a journal issue [Dequaire et al., 2018]. We provide the following contributions:

- Development of the deep tracking framework to perform on a moving platform by harnessing ego-motion estimates to update the belief state in a principled way
- Demonstration of the effectiveness of our proposed solution on synthetic data featuring dynamic objects and a moving sensor
- Demonstration that our proposed solution improves tracking ability on 28km of real-world urban data collected from a vehicle in the Oxford city centre, UK
- Demonstration that the system's state representation learned on the task of tracking can be used to semantically label the scene in terms of static versus dynamic objects, on synthetic data.

Improving Place-Specific State Estimation by Incorporating Scene Context

In Chapter 6 we address the importance of enabling a tracking system to adapt its motion models to the scene context. We consider road patterns as a proxy for scene layouts, and inspired by the work of De Brabandere et al. [2016] on *Dynamic Filter Networks* we propose conditioning network dynamics on the scene layout. We first observe the limitations of our baseline model in predicting pertinent dynamics at road intersections and roundabouts, and investigate two mechanisms for adapting network dynamics appropriately. We show on synthetic scenarios that the extended architecture is able to incorporate scene context that generalises to unseen layouts. We then extend the analysis to real-world layouts and link usability to real-world scenarios in a setting where GPS positioning is available.

This is an interesting development as context is an important component to scene understanding and object motion prediction. Cars tend to follow the curve of the road and stop at red lights in urban environments. If a tracker is not able to perceive an upcoming roundabout or traffic light, it may well predict trajectories that are neither relevant nor desirable in view of the scene context. By introducing mechanisms for incorporating the underlying road layout, we contribute to efforts

for adapting a tracker's motion models to the live scene at hand in a self-supervised way. In doing so we make the following contributions:

- Development of the deep tracking framework to predict motion according to an input scene layout, without the need for annotated ground truth or pre-processing
- Demonstration of the effectiveness of dynamic filters for tayloring motion models to the scene at hand on synthetic layouts
- Presentation of preliminary results linking usability to real world applications using GPS localisation in overhead maps.

1.2 Publications

Part of the work presented in this thesis has appeared in the following publications:

- **J. Dequaire**, C.H. Tong, W. Churchill, and I, Posner, *Off the beaten track: Predicting localisation performance in visual teach and repeat*, The International Conference on Robotics and Automation, 2015 [Dequaire et al., 2016] - Chapter 3
- **J. Dequaire**, P.Ondrúška, D. Rao, D. Wang, and I. Posner, *Deep tracking in the wild: End-to-end tracking using recurrent neural networks*, The International Journal of Robotics Research, 2018 [Dequaire et al., 2018] - Chapters 4 and 5
- P. Ondrúška, **J. Dequaire**, D. Zeng Wang, and I, Posner, *End-to-End Tracking and Semantic Segmentation Using Recurrent Neural Networks*, Robotics: Science and Systems, Workshop on Limits and Potentials of Deep Learning in Robotics, 2016, **[Best Workshop Paper Award]** [Ondruska et al., 2016] - Chapter 4

1.3 Thesis Structure

In Chapter 2 we consider related works and collate relevant background material to the work presented in this thesis. We provide an overview of learning latent models for inference in the context of field robotics and present the two methods used in this thesis namely Gaussian Processes and Recurrent Neural Networks for Bayesian Filtering. We also clarify the general systems and methods used in our experiments We refer to sections of this chapter accordingly throughout the thesis.

In Chapter 3 we address the limitation of a localisation system and present an appearance-based approach to predicting *a priori* how well a localiser is expected to perform along a given route of interest.

In Chapter 4 we address the limitation of a perception system and extend the *Deep Tracking* [Ondruska and Posner, 2016] framework to dynamic and complex real-world scenarios.

In Chapter 5, we address the limitation of a static tracking system and extend our framework to handle vehicle ego-motion in dynamic real world environments.

In Chapter 6, we address the limitation of a tracker agnostic to scene context and introduce an extended architecture for adapting tracking predictions to the underlying road layout.

We summarise our contributions in Chapter 7 and suggest areas of future work building on our findings.

2

Related works, Tools and Methods

This thesis extends the capacity for robots to build a high level understanding of their environment directly from data, for far-sighted perception and localisation. We first frame our research in light of related works in Section 2.1. In Section 2.2 we present the methods used for performing inference in the field. We then present the frameworks and conventions adopted to enable visual localisation and ego-motion estimation in Section 2.3. We refer the reader to the very clearly presented material found in [Thrun et al., 2005], [Bishop, 2006], [Rasmussen and Williams, 2012] for more in depth reading relating to the learning-based methods presented.

2.1 Related Works

In this section we present work related to the various aspects of localisation and perception addressed in this thesis.

Predicting Localisation Performance

Vision-based localisation is an active area of research due to its reliance on ubiquitous and comparatively low-cost sensors (see, for example, Dayoub and Duckett [2008], Konolige et al. [2010], Lee et al. [2014], Linegar et al. [2015], Muehlfellner et al. [2013]). From amongst several approaches, *Teach and Repeat* has distinguished itself as being an effective solution in a number of domains such as space exploration

[Furgale and Barfoot, 2010] and life-long autonomous driving [Arroyo et al., 2015, Churchill and Newman, 2013, Linegar et al., 2016a]. In this framework, a robot is driven along a desired path in a *teach* phase, during which it builds a topometric map of the world [Sibley et al., 2010, Strasdat et al., 2011] by estimating its relative pose between consecutive keyframes. During a *repeat* run, localisation is performed by matching features between the live feed and those of the map, estimating the position of the robot relative to the route previously traversed as we further detail in Section 2.3.3. In doing so, it is commonly observed that some environments are less lenient than others to even small lateral or rotational deviations from the known trajectory, so that localisation can be lost if the taught trajectory is not followed within some tolerance dictated by appearance.

Localisation corridor Despite this strong coupling between navigation, planning and control of a vehicle, teach and repeat systems are often developed in isolation from planning and control. A notable exception here is Ostafew et al. [2013] which aims to learn controllers which reduce path-tracking errors over time. However, situations commonly arise where environment conditions or vehicle characteristics may not allow for an exact path repetition. In addition, path deviations may be tolerated in order to pursue alternative mission goals such as energy savings [Martin and Corke, 2014, Ondruska et al., 2015] or more accurate situational awareness [Velez et al., 2011]. In these cases, *a-priori* knowledge of the degree to which deviations from a taught trajectory can be tolerated *as soon as a route is acquired* is pivotal for planning and control. Yet this problem has thus far received little attention.

Krüsi et al. [2014] compared the accuracy of a laser based Teach and Repeat system to a vision based one. They find the laser based system is more robust to path deviations and ambient conditions. However they do not model or predict localiser performance. Furgale and Barfoot [2010]'s vision based Teach and Repeat system demonstrated autonomous retro-traverse over 32 km in an outdoor setting. As part of their experiments, they calculated how robust the localisation was to lateral deviations. The camera was manually placed at offsets from the original

teach path until the failure point was found. By testing at several locations along the route, they computed a single, constant lateral offset for the map.

Predicting Localisation Envelopes Churchill et al. [2015] extended the work of Furgale and Barfoot [2010] by aiming to capture the local variation of the localiser performance as a function of position in the map. This was modelled using a Gaussian Process, with one of the input variables being position. In this context as elsewhere, Gaussian Processes provide a framework amenable to incorporating new measurements as they become available while providing uncertainty estimates in a principled way. Churchill et al. [2015] demonstrated that trajectory shape can be used to accurately predict the performance of the localisation system once a number of repeat passes have been performed. However, the limitation of this approach lies in its use of position as an input variable, which means that the model is specific to a particular route, and cannot be generalised to other maps or places.

In Chapter 3 we advocate an approach for mapping directly from a teach trajectory to likely localisation performance in the surrounding area – its *localisation envelope*. We extend the work of Churchill et al. [2015] by proposing an appearance-based approach to *predict* localisation performance, in an environment where only a single teach path has been recorded. Crucially, unlike Churchill et al. [2015], we are able to *generalise* predictions to freshly taught trajectories.

Since publication of our work [Dequaire et al., 2016], Merzić et al. [2017] addressed the challenge of estimating whether a given map is of sufficient quality for localisation. They propose a framework for estimating localisation performance given a 6 DOF pose in a map, in the context of visual landmark-based mapping.

Introspection Our work is also an early instantiation of *robot instrospection*, whereby the localisation system learns to estimate its own performance in a given environment. System and model introspection is a growing field that explores ways of providing system performance guarantees, reasoning, and limitations, which constitute valuable grounds for developing user *trust* [Israelsen and Ahmed, 2017]. In the context of perception for visual teach and repeat, Gurău et al. [2016] explore

the idea of predicting the likely performance of a robot's perception system based on past experience. Predictions can be leveraged to provide autonomous decision-making processes with valuable time to failure estimates of the perception module [Gurău et al., 2017]. Additionally, Hu and Kantor [2017] learn a full posterior predictive distribution of performance estimates of a perception module.

Perception for State Estimation

It is well recognised that prediction is a major component of intelligence in humans [Hawkins and Blakeslee, 2007] and the wider animal kingdom [Zentall, 2005]. It requires and reflects a deep understanding of the world or system considered. We believe in the paradigm that perception is guided by the anticipation of future events. A recent study imaging the visual cortex of human volunteers found evidence of preplay of anticipated events [Ekman et al., 2017]. After familiarizing subjects with a spatial sequence, flashing only the start of the sequence triggered an activity wave in V1 resembling the full sequence.

Despite this growing evidence, robotic perception systems often detect, track, and classify the environment by relying on instantaneous, occluded data provided by their sensors. In this thesis we are therefore interested in exploring how a perception system can be enhanced with the ability to estimate the state of its environment in order to predict future observations, and frame the learning problem as one of laser-based end-to-end tracking.

Object Tracking Pipelines Object detection and tracking has posed a long-standing challenge of robotics. Commonly however, tracking frameworks employ multiple hand-engineered stages developed independently, such as object detection, semantic classification, data association, state estimation, motion modeling, and occupancy grid generation. Several such approaches perform *model-free* tracking [Vu et al., 2007], [Yang and Wang, 2011], and assume a general motion model for objects, but suffer in terms of robustness. In contrast, *model-based* techniques [Zhao and Thorpe, 1998], [Arras et al., 2007], [Petrovskaya and Thrun, 2009] explicitly consider the type of object being tracked, which limits their generality but improves tracking

performance for the object classes considered. In all of these cases, the use of multiple hand-engineered stages in the framework is cumbersome, requires manual tuning of parameters, and introduces unnecessary additional failure modes to the tracking process. As robots begin to operate in increasingly complex environments, this approach becomes more and more infeasible.

Deep Methods for Tracking Recent research in machine learning has shown the ability of deep neural networks to capture complex structure, achieving state-of-the-art performance in numerous computer vision and natural language processing tasks (see early breakthrough results on ImageNet of Krizhevsky et al. [2012], Dahl et al. [2012], Wang et al. [2012]). Such models usually require large, task-specific corpora of annotated ground-truth labels to master the desired task. However, such a data-intensive supervised learning paradigm is infeasible for object tracking in crowded urban environments, as we are required to learn a model of the environment without access to corresponding ground truth.

Addressing both the challenge of multi-stage tracking pipelines and data-intensive supervision, [Ondruska and Posner, 2016] propose to replace these multiple stages with a single end-to-end learning framework known as *Deep Tracking* which directly learns the mapping from raw laser input to an unoccluded occupancy grid. They leverage recurrent neural networks (RNN, [Medsker and Jain, 2001]) to capture the state and evolution of the world in a sequence of laser frames, in a fully self-supervised manner thus removing the need for expensive labelled data. Here, the end-to-end model implicitly performs the individual steps implemented in the traditional multi-stage pipeline, without need for hand engineering or tuning of parameters for each component. Choi et al. [2016] follow a different approach using *recurrent flow networks*. The model explicitly encodes a range of velocities in the hidden layers of an RNN, and uses Bayesian optimisation to learn the network parameters responsible for updating velocity estimation and occupancy prediction. However, the model is not trained to explicitly track objects through occlusion. Interesting recent work by [Milan et al., 2017] casts the classical Bayesian

state estimation, data association, and particle birth/date estimation tasks as an end-to-end learning framework using recurrent neural networks.

Deep Methods for State Estimation Our work relates to deep learning approaches applied to predictive video modelling [Patraucean et al., 2015], [Lotter et al., 2016], in that it is trained to predict the future state of the world based on current input data. This is of vital importance to achieving accurate prediction and tracking, as to successfully predict the future location of dynamic objects in the scene, the model must implicitly store the position and velocity of each object in its internal memory state. Our work is also is an early instantiation of (and indeed in some cases inspiration for) approaches to learning or intuiting physics based directly on sequences of sensor observations [Watters et al., 2017], [Ehrhardt et al., 2017].

Transfer of Knowledge It is also possible to exploit the fact that the learned representation captures latent higher-order information in the data, such as the location, shape, and velocity of an object, which is also pertinent to semantic classification tasks. This is a form of *inductive transfer* of knowledge between machine learning tasks [Pan and Yang, 2010], [Weiss et al., 2016]. In the context of neural networks it has been successfully applied to a range of tasks, in the areas of multi-task learning [Mitchell and Thrun, 1993], [Caruana, 1995] and in the form of unsupervised pre-training and supervised fine-tuning [Pennington et al., 2014], [Le, 2013].

Predicting World Dynamics from a Moving Platform

In Chapter 5 we extend our tracking framework to a moving platform. Our work relates to learning in motion and deep learning approaches to addressing robot motion.

Motion Estimation for Learning Motion is a crucial component to learning about the world, and there is extensive evidence that motion plays a key role in the development of the human visual system. From their experiments on kittens, experimental psychologists Held and Hein suggested that self-generated movement in

concert with visual feedback was mandatory for proper perceptual development [Held and Hein, 1963]. In their 1963 experiment, they compared the visual perception development of two kittens exposed to the same visual experience, though one kitten was able to move about freely of its own accord, whilst the other one was carried in a basket and unable to self-generate movement. They observed that the kitten unable to move in its environment suffered fundamental perception problems. In the machine learning community, Jayaraman and Grauman [2015] explicitly link the task of scene recognition to observer ego-motion by enforcing a neural network to learn equivariant visual representations from egocentric video. They demonstrate improved performance on visual recognition and next-best view prediction tasks. In a similar direction, Pathak et al. [2017] explores a novel approach to unsupervised feature learning based on building visual representations from low-level motion-based grouping cues in images. Leveraging transfer learning they demonstrate improved ability capacity for image classification and segmentation under limited computational or data resources. In a different area, Wang et al. [2017] introduce a novel end-to-end framework for estimating monocular visual odometry with recurrent neural networks.

Deep methods for Tracking in Motion Closest to our work is Luo et al. [2018] who propose an end-to-end deep learning framework for jointly reasoning about detection, tracking, and motion forecasting given 3D point clouds. They aggregate the last n laser frames and perform 3D convolutions across space and time using a bird's eye view representation of the world. They produce bounding boxes for detected objects forward into the future. Their approach requires labelled data and cannot account for future ego-robot motion. Building on their work, Casas et al. [2018] add maps for traffic constraint and a discounted temporal factor for accounting for the inherent ambiguity of the future.

Spatial Transformer Networks As we set our work in a robotic system, we do not need to learn the ego-motion estimate, but rather use readily available ego-motion information to update our belief state. In Chapter 5, we consider

the *Spatial Transformer* network module developed by Jaderberg et al. [2015] to decouple sensor ego-motion from world dynamics. In their original work, the spatial transformer is a learnable module which explicitly allows the spatial manipulation of data within a network. Inserted into existing convolutional architectures and fully differentiable, it can learn to actively spatially transform feature maps by conditioning the transformation on the feature maps themselves. The network is split in three parts which we illustrate in Figure 2.1.

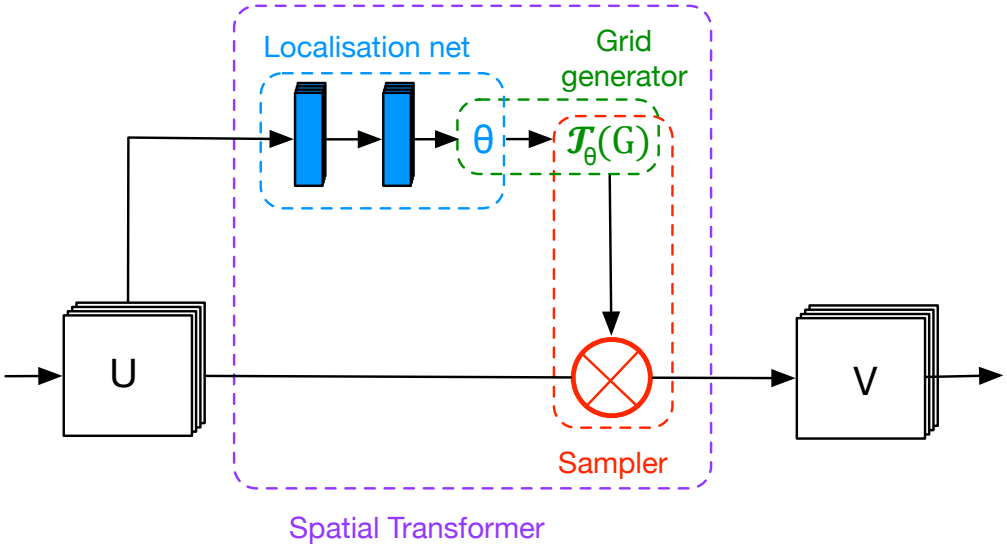


Figure 2.1: The architecture of the Spatial Transformer network of Jaderberg et al. [2015]. An input feature map U is passed through a localisation net to regress to an array of transform parameters θ. The coordinate grid G of V is transformed by the Grid generator to the sampling grid $\mathcal{T}_\theta(G)$. A sampler applies the sampling grid to U to produce V. All input feature maps of U are transformed in the same way. Figure inspired from Jaderberg et al. [2015].

A *localisation net* first takes an input feature map U and regresses to parameters θ of the transformation that should be applied to U. If we define G as the regular grid of pixels $G = \{(x_i^t, y_i^t)\}$ forming the output feature map V, a *grid generator* then converts the predicted transform parameters to a sampling grid $\mathcal{T}_\theta(G)$. This grid corresponds to the set of points where the input map U should be sampled to produce output V. If $\mathcal{T}_\theta$ is a 2D affine transformation with parameters $\theta = \{\theta_{i,j}\}$, the sampling grid indicates the following pointwise transformation:

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \mathcal{T}_\theta(G_i) = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}, \tag{2.1}$$

where s and t superscripts correspond to the input source and target output feature map coordinates.

Finally, a *sampler* module samples U according to $\mathcal{T}_\theta(G)$ and produces V. In theory, any sampling kernel can be used so long as gradients can be defined with respect to the source coordinates x_i^s and y_i^s. In Chapter 5, we use the bilinear sampling kernel to update the hidden state according to vehicle ego-motion:

$$V_i = \sum_n^H \sum_m^W U_{nm} \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|),$$

where U_{nm} corresponds to the value at location (n, m) in the input and V_i is the output value for pixel i at location (x_i^t, y_i^t).

In their work, Jaderberg et al. [2015] demonstrate that a spatial transformer module can learn to undistort, crop, and rescale input maps of distorted digits and coloured bird images, with improved classification compared to more traditional architectures. We do not know of any other work that utilises Spatial Transformers to transform maps according to sensor ego-motion.

Predicting World Dynamics in Context

In Chapter 6 we extend the tracking framework for learning to predict scene dynamics according to the local scene context. Our work relates to context-tracking, activity and trajectory forecasting and object-scene interaction.

Tracking in Context Classical approaches to tracking are agnostic of the local scene context. Whether the framework be a linear/non linear Kalman Filter or particle filter approach, model-based or model-free, they traditionally focus on the tracked objects and their possible interactions with other objects in the scene. Motion models are class or environment specific (e.g particular motion models for cars in urban environments rather than off-road) but are not adapted to the instantaneous

and local scene. To the extreme, this can lead to incoherent track predictions such as pedestrians walking into walls or cars driving off the road. [Yang et al., 2009] improve visual tracking accuracy by learning to track auxiliary objects to the target that present strong motion correlation to the tracked target. In the case of tracking a person's face, this can consist in tracking other persons in the scene that are persistently co-occurring with the target and easy to track. The method is automatic and learned online. Recent work suggests that humans estimate scene motion in relation to the scene perceived, and that primate brain combines static form cues with motion cues when predicting motion [Krekelberg et al., 2003]. Our work aims to move in this direction by combining static scene cues with object dynamics.

Activity and Trajectory Forecasting Seminal work by Kitani et al. [2012] models the effects of the physical environment on the future actions of people using visual input. They learn pedestrian preferences in terms of where they prefer walking (e.g sidewalk) given an initial and destination point. [Ballan et al., 2016] look at predicting human trajectory based on the interplay between functional properties of the scene and the prior knowledge of moving agents in that scene. Leveraging deep learning they explicitly learn patch descriptors encoding the probability of an object class moving from one patch to the other, being in that particular patch, or changing behaviour, which they then use to predict plausible trajectories. Their system requires explicit data association in the form of tracks and knowledge of object class. Similar work by [Walker et al., 2014] extracts mid-level representations and learns spatial behaviours of individual patches in static images. They learn patch transitions from similar feature patches in temporally close frames which are paired up using a tracker. They additionally learn a reward function for the interaction of each element within the scene using a pre-built training set. At deployment time they first estimate the elements in the scene that are likely to be active, before applying their learned transition and reward functions to predict both possible motion and future visual appearance in the scene. Non parametric data-driven approaches to temporal motion prediction such as [Yuen and Torralba, 2010] rely on comparing a query input scene to a bank of videos for which expected

motion models have been learned. This relies on collecting and learning from an extensively large amount of training data to learn all the possible spatial and temporal configurations of objects observed in the world.

Predicting Traffic Patterns More specifically focused on traffic pattern prediction, [Zhang et al., 2013], [Geiger et al., 2011] propose a generative model of 3D urban scenes which reasons about the geometry and objects present in the scene. From a small number of patterns they are able to model the vast majority of traffic scenes and estimate vehicle-to-lane association. They consider tracklets and a Kalman filter to learn traffic patterns for a restricted number of possible intersections given a static image.

Scene Layout Estimation Ahead of predicting trajectories, we are interested in detecting the scene layout. Closest to our desired layout input is the work of [Ballardini et al., 2017] who propose an online probabilistic road intersection detector and classifier. From an overhead projection of a forward facing camera depth estimate they predict one of K intersection types. Their estimate remains a generic model of an intersection and it is not estimated in the robot frame.

Dynamic Filter Networks We are interested in learning methods for state estimation in context. We are therefore drawn to learnable modules - similar to the Spatial Transformer - and are inspired by Dynamic Filter Networks introduced by [De Brabandere et al., 2016]. In their work, they generate convolutional filters dynamically conditioned on the input at hand. This for example enables to generate future frames with parameters adapted to the motion pattern within a particular video. Their approach consists of a *filter-generating network*, and a *dynamic filtering layer* which we illustrate in Figure 2.2 on the following page.

In their local filtering approach, the filter-generating network produces filters conditioned on an input A that are applied locally to an input B. Different filters can thus be applied to different positions of input B. The filtering operation is

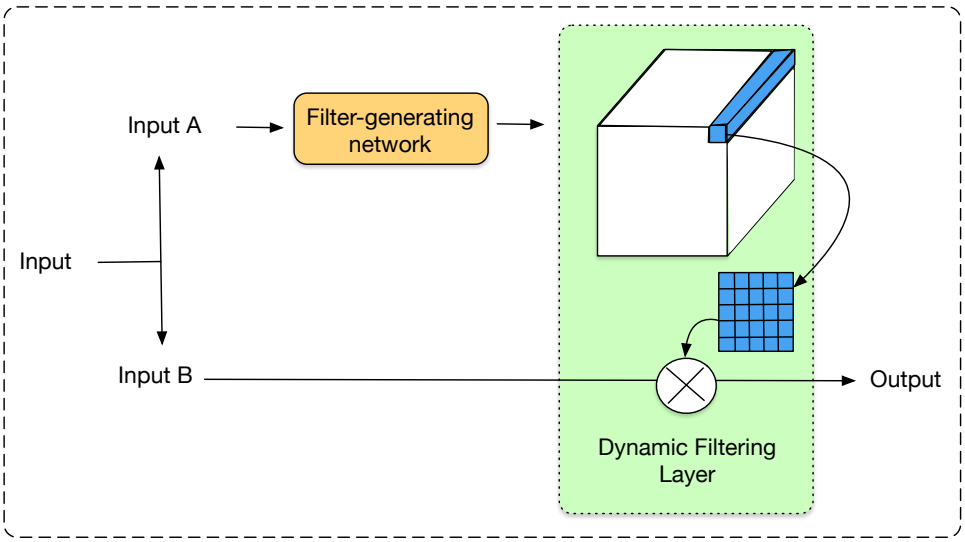


Figure 2.2: The Dynamic Filter Network introduced by De Brabandere et al. [2016] consists in a filter generating network conditioned on an input A, and a dynamic filtering layer which applies the dynamically produced filters to the input B. In this representation (inspired from De Brabandere et al. [2016]), filters are dynamically produced for every location of Input B and referred to as *local filtering*, similar to locally connected networks. An alternative option would be to produce a bank of filters conditioned on input A, applied to all input B similarly to traditional translation invariance convolutional filters.

no longer translation invariant but rather position specific. For each position (i, j) of the conditioning input I, a specific local filter ${W_\theta}^{(i,j)}$ is applied to the region centered on position (i, j) of I:

$$Y(i, j) = {W_\theta}^{(i,j)}(I(i, j))$$

Their module naturally inserts in a neural network architecture and can be learned in consort with the task at hand. We clarify this module further in Chapter 6 where we use Dynamic Filters for conditioning motion models on the scene layout.

2.2 Inference for Predictive Sensing

In this thesis, we are interested in learning models of sensing systems so as to make predictions of future observations of these systems. In the context of navigation, we wish to predict the expected performance of a localisation system given the appearance of a scene. In the context of perception, we wish to predict future unoccluded scene occupancy given a sequence of observations. We are therefore interested in making inferences about the relationship between input vectors $\mathbf{x}$ and output targets y. For both tasks we have access to a collection of N example pairs of observations which we note $\mathcal{D} = (\mathbf{X}, \mathbf{y}) = \{(\mathbf{x_i}, y_i)|i = 1, ..., N\}$. Our inference objective can be framed as a *supervised learning* problem, which aims to learn a function f that maps from input vector $\mathbf{x}$ to target output y:

$$f : \mathbf{X} \longrightarrow \mathbf{y}.$$

Observations are typically corrupted with noise which is commonly modelled as a normal distribution of mean zero and variance σ^2, resulting in the updated mapping:

$$y = f(\mathbf{x}) + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, \sigma^2).$$

The objective then becomes one of modelling the predictive distribution $p(\mathbf{y}|\mathbf{X})$ in order to make predictions y^* for new input $\mathbf{x}^*$: $p(y^*|\mathbf{x}^*, \mathcal{D})$.

In this thesis, we are particularly interested in learning mappings between variables related to *systems*. We are therefore inclined to turn towards latent models for capturing f, that is we assume there is a latent "hidden" process emanating from these systems that explains the observed data, but that the process itself is not directly observable. We are also interested in framing our work in a Bayesian setting, where we aim to learn a *distribution* over possible models, rather than make point estimates. This enables making predictions weighted by the model uncertainty which is a useful feature for decision making in robotics. Inference for

new datapoints $\mathbf{x}^*$ consists in marginalising over the posterior distribution over models after seeing some data $p(f|\mathcal{D})$:

$$p(y^*|\mathbf{x}^*, \mathcal{D}) = \int p(y^*|\mathbf{x}^*, f)p(f|\mathcal{D})df. \tag{2.2}$$

This being said, we also require *practical* methods that can learn complex real world processes. This requires methods that can learn directly from data, as we have little prior idea what these complex processes "look like". This may require approximating the distributions of Equation 2.2 if they are intractable. Combining all these requirements with the problems at hand, we make the following two model choices:

Predicting Localisation Performance with Gaussian Processes For modelling localisation prediction, we consider a Gaussian Process which directly models a distribution over functions $p(f|\mathcal{D})$. Figure 2.3 (left) illustrates the graphical model associated with the latent process f. Any collection of output variables $\mathbf{y}$ are noisy observations of function values $\mathbf{f} = \{f(\mathbf{x})\}$ which form a multi-variate Gaussian distribution whose covariance matrix is defined by a similarity metric over input variables. The latent nodes are fully connected and defined by the appropriate entry in the covariance matrix. If we also model the observation $p(y^*|\mathbf{x}^*, f)$ as Gaussian, the predictive distribution of Equation 2.2 can be readily evaluated:

$$p(y^*|\mathbf{x}^*, \mathcal{D}) \sim \mathcal{N}(\bar{f}^*, cov(f^*)). \tag{2.3}$$

As the distribution is shaped by the data itself, it provides a learnable flexible Bayesian model well suited for real world data modelling.

Predicting Future State Occupancy with Recurrent Neural Networks For modelling future unoccluded scene occupancy, we consider the formulation of *Deep Tracking* [Ondruska and Posner, 2016] and frame the problem as one of Dynamic Bayes filtering. We illustrate this with a graphical model in Figure 2.3, right. Under this formulation, we consider a temporal sequence of sensor observations

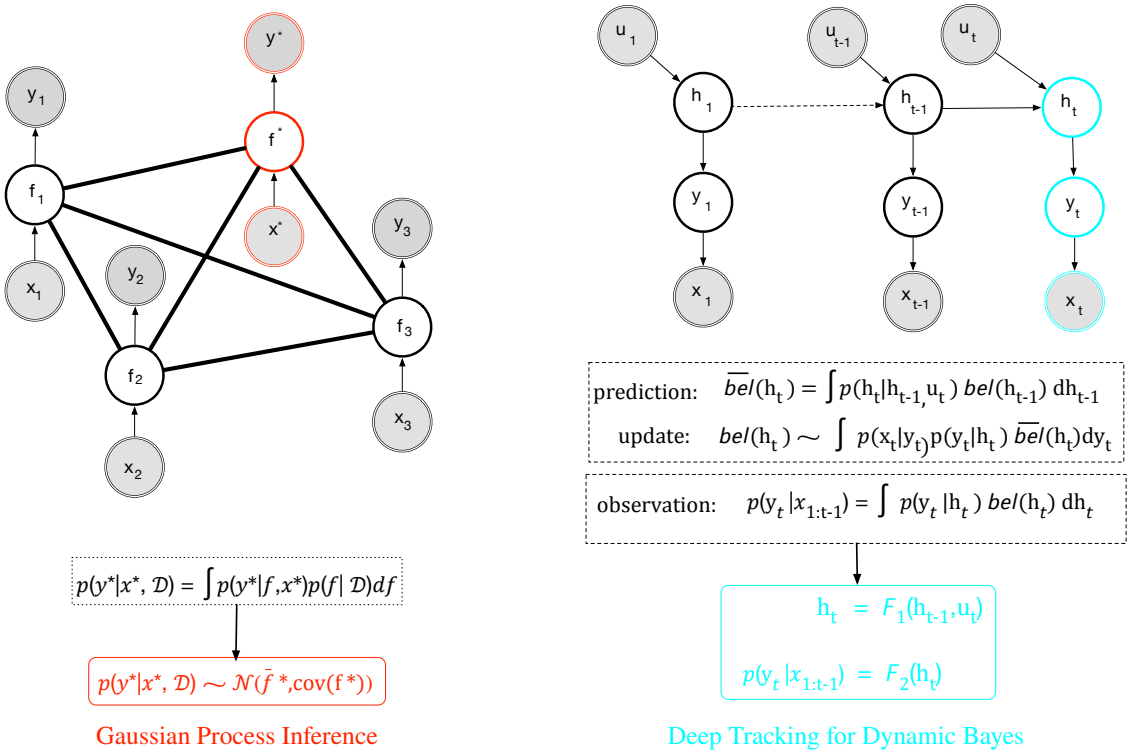


Figure 2.3: Graphical models of the assumed processes underlying the observed variables of our localisation and perception systems. Gray variables are observed, white variables are latent or "hidden". Left: localisation performance outputs $\mathbf{y}$ are noisy observations of an underlying Gaussian Process which defines a Gaussian distribution over latent variables $\mathbf{f}$. Edges in the fully connected graph correspond to entries in the covariance matrix, which is defined as a kernel function over the inputs $\mathbf{x}$. Predictive inference for a new input $\mathbf{x}^*$ is achieved by directly marginalising over the latent distribution. Right: sensor observations $\mathbf{x}$ are partial observations of the true unoccluded state occupancy $\mathbf{y}_t$ which is explained by an underlying dynamic process $\mathbf{h}_t$ with control input $\mathbf{u}_t$. Both the observation model $p(\mathbf{y}_t|\mathbf{x}_{1:t})$ and belief state distribution $bel(\mathbf{h}_t)$ are approximated by functions and $\mathbf{h}_t = \mathcal{F}_1(\mathbf{h}_{t-1}, \mathbf{u}_t)$, and $p(\mathbf{y}_t|\mathbf{x}_{1:t}) = \mathcal{F}_2(\mathbf{h}_t)$.

$\mathbf{x}_{1:t}$ as corresponding to partial observations of the underlying true unoccluded occupancy state of the world $\mathbf{y}_{1:t}$. State occupancy $\mathbf{y}_t$ can be explained by an underlying Markov process $\mathbf{h}_t$ which evolves over time due to world dynamics and input control $\mathbf{u}_t$ representing the robot motion. The Bayes filter algorithm carries a distribution over possible states for $\mathbf{h}_t$ as a belief $bel(\mathbf{h}_t)$. This belief is updated recursively at every timestep according to the system dynamics and observed measurement. Modelling $p(\mathbf{y}_t|\mathbf{x}_{1:t})$ therefore requires specifying a suitable belief state representation, system transition probability, and observation models. This is a difficult task for real world applications. We avoid the need to specify

these distributions and adopt the approach of [Ondruska and Posner, 2016] who use expressive neural networks to learn an approximation to the state transition and observation models:

$$\begin{aligned} \mathbf{h}_t &= \mathcal{F}_1(\mathbf{h}_{t-1}, \mathbf{x}_t, \mathbf{u}_t) \\ p(\mathbf{y}_t|\mathbf{x}_{1:t}) &= \mathcal{F}_2(\mathbf{h}_t). \end{aligned} \tag{2.4}$$

Both functions are learned simultaneously end-to-end as part of a recurrent neural network.

In the next sections, we give a brief overview of the general approach followed to define a suitable objective function and model form for approaching a supervised learning task. We then further detail the two models chosen in our work.

2.2.1 Supervised Learning

Supervised learning requires choosing appropriate characteristics for f ("what shape and complexity do we expect the function to have?") as well as an appropriate *objective function* to assess how well f explains the data ("what is my metric of success?").

Objective Function One choice of objective function can take the form of estimating f^* such that it *maximises the likelihood* of the data:

$$f^* = \arg_f \max p(\mathbf{y}|\mathbf{X}, f).$$

This produces a point estimate of f, resulting in deterministic predictions without any measure of (un)certainty in the model choice. This approach is often plagued by the well known problem of overfitting. This arises when the learned model fits the training data very well but does not generalise well to new data points of a given test set. Maximum likelihood by construction is prone to overfitting as model parameters are learned to precisely fit the training data well.

To alleviate the problem of overfitting, a more Bayesian approach can introduce an additional prior distribution over models, $p(f)$, which will balance out the

influence of the observed data on the learned model. The objective becomes one of learning a *posterior distribution* over models after seeing the data:

$$p(f|\mathcal{D}) \propto p(\mathbf{y}|\mathbf{X}, f)p(f)$$

$$posterior \quad \propto likelihood \times prior.$$

One can either maximise the posterior distribution to provide a *maximum a posteriori estimate*:

$$f^* = \mathbf{f}_{MAP} = \arg_{\mathbf{w}}\max\, p(\mathbf{f}|\mathcal{D}),$$

or keep the full posterior distribution of Equation 2.5 to perform Bayesian inference:

$$p(y^*|\mathbf{x}^*, \mathcal{D}) = \int p(y^*|\mathbf{x}^*, f)p(f|\mathcal{D})df. \tag{2.5}$$

Output predictions now consist in a weighted combination of individual model predictions $p(y^*|\mathbf{x}^*, f)$, weighted by the corresponding probability of the model $p(f|\mathcal{D})$ under the posterior distribution.

Model choice Choosing an appropriate form for f has no absolute solution, though one can make assumptions as to which characteristics the function is required (or desired) to have depending on the task at hand. Two common approaches to choosing characteristics for f are known as *parametric* and *non-parametric* models with the following general characteristics.

Parametric approaches assume a functional form for f, and parameters are learned from the training data. A simple form for f consists in a linear combination of the input variables:

$$y = f_{\mathbf{w}}(\mathbf{x}) + \epsilon = \mathbf{w}^T\mathbf{x} + \epsilon$$

This is a form of *linear regression*, as f is linear with respect to the parameters. It is also linear with respect to the input $\mathbf{x}$, which is a very limiting assumption for

real world settings. A possible adjustment can project every input $\mathbf{x}$ onto a higher dimensional space by considering non linear basis functions $\Phi(\mathbf{x})$. If we consider the example of a polynomial basis function, function f assumes the following form:

$$f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \phi(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2^2 + ...w_k x^k, \tag{2.6}$$

where k is a positive integer greater than 1. The function remains parametric and linear in its parameters $\mathbf{w} = \{w_0, ..w_k\}$. Further complexity can be modelled by passing the linear combination of basis functions through a nonlinear activation function $h()$:

$$f(\mathbf{x}, \mathbf{w}) = h\Big(\mathbf{w}^T \phi(\mathbf{x})\Big). \tag{2.7}$$

Neural networks extend this modelling approach by making the basis functions no longer fixed, but dependant on parameters that can be adjusted along with weights during training. The full network is built as a modular composition of layers of nonlinear activations such as 2.7. In theory, they can approximate any arbitrary function provided they are given an appropriate number of processing units and training data. This can make them a powerful modelling choice when it is difficult to specify or integrate over distributions. We further discuss neural networks in Section 2.2.4.

Non-parametric on the other hand make few assumptions about the form of the mapping function. This can be interesting if there is little a priori knowledge of the functional form which is rather learned from the training data itself. This can also lead to more interpretable models, where we do not have to choose a level of complexity and shape for f. At training time, the search space is shrunk for the function to fit the observations. As the data determines the shape of the function, they are kept at test time to make new predictions for new data points. These models typically require a metric or kernel function to be defined that measures the similarity of any two vectors in input space. The training consists in learning the hyperparameters of this similarity measure that best explains the data. Prediction

for a new input $\mathbf{x}^*$ consists in applying the kernel between the new input and each of the training inputs. This results in a weighted sum of similarities:

$$f(\mathbf{x}^*) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x}^*). \tag{2.8}$$

A simple example of a non-parametric model is k-nearest neighbours where the algorithm clusters the data around k centres that best represent the distribution. A new test data point is assigned the same label as that of the most similar cluster centre. The only assumption made is that data points that are similar (according to some chosen metric) will yield similar outputs. A popular probabilistic choice is the Gaussian Process, which considers the collection of function outputs $\{f(\mathbf{x})\}$ as random variables, any finite number of which have a joint Gaussian distribution. This means that any test output vector $\mathbf{f}^*$ along with the training set outputs $\mathbf{f} = \{f(\mathbf{x}_1), ...f(\mathbf{x}_d)\}$ have a joint Gaussian distribution whose covariance function is defined by a kernel on the inputs $k(\mathbf{x}, \mathbf{x}')$:

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}^* \end{bmatrix} \sim \mathcal{N}\left(\begin{matrix} m(\mathbf{X}) \\ m(\mathbf{X}^*) \end{matrix} \quad , \quad \begin{bmatrix} K(\mathbf{X}, \mathbf{X}) & K(\mathbf{X}, \mathbf{X}^*) \\ K(\mathbf{X}^*, \mathbf{X}) & K(\mathbf{X}^*, \mathbf{X}^*) \end{bmatrix} \right) \tag{2.9}$$

The predictive distribution of Equation 2.5 can be easily evaluated by conditioning the joint prior distribution on the observations $p(\mathbf{f}^*|\mathbf{X}^*, \mathbf{X}, \mathbf{f})$, resulting in a Gaussian form. This approach does not require much data, provides notions of uncertainty on the predictions, and only requires a prior assumption over the space of functions considered, via the choice of kernel function which can be parametrised and fit to capture the relationships within the training data points. We choose a Gaussian Process for predictive localisation.

2.2.2 Gaussian Processes for Predicting Localisation Performance

In this thesis we are interested in developing a localisation system's situational awareness such that it can predict ahead of time how well it expects to be able to localise in a given environment. We consider localisation in the context of Teach and Repeat, a framework we present in Section 2.3.3, but which essentially enables a

robot to retrace a taught trajectory. During the teach phase, a chain of robot poses is saved to local memory as a topometric map, alongside with keyframe images from the camera. To repeat this route, the robot needs to be able to localise against it, and does so by trying to recognise local features in live images, from those saved in memory. Once features are matched, the localiser will try to estimate its pose with respect to the saved map, allowing it follow the desired route.

This localisation process is consequently intimately tied to characteristics such as the appearance of the saved keyframes and position of the robot at repeat time. To learn a predictive model of the localisation system, we therefore hypothesise that there may be a relationship f to be learned between input features characterising teach and repeat path characteristics, and output localisation performance:

$$f : \mathbf{x} = \{\text{path characteristics}\} \longrightarrow y = \text{localisation performance}.$$

We do not have any prior idea as to what shape or complexity f may have, but we can assume that similar inputs in feature space would yield similar localisation performance output values. Inversely, if input data points are very dissimilar, we don't really know how their outputs are going to differ. If a test data point is unlike any other training data point, we would like to know that the model is uncertain about its predictions. This could inform a planner to avoid an area to avoid getting lost, or on the contrary further explore to collect more data. Finally, we are interested in learning a model from one taught trajectory and a number of repeats, in order to make predictions for a second route in a similar area. We therefore need a model that can use the data itself, without needing too much of it. These considerations lead us towards using Gaussian Processes for modelling the localisation envelope.

Overview

We introduce the Gaussian Process from the weight-space view. In a simple Bayesian parametric approach to linear regression, we typically introduce a Gaussian prior over weights $p(\mathbf{w})$ and define f as a linear combination of basis functions $f(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x})$. We initially omit the observation noise ϵ for clarity of exposition. In this approach,

any given value of $\mathbf{w}$ defines a particular function f, so a prior distribution over weights represents a prior distribution over functions $f(\mathbf{x})$. As we are interested in evaluating the function f at a set of input $\mathbf{X}$, we are interested in the joint distribution of output values $\mathbf{y} = \{f(\mathbf{x}_1), .., f(\mathbf{x}_N)\}$:

$$\mathbf{y} = \mathbf{\Phi}\mathbf{w}, \tag{2.10}$$

where $\mathbf{\Phi}$ is the design matrix with elements $\Phi_{nk} = \phi_k(\mathbf{x_n})$. As $\mathbf{y}$ is a linear combination of Gaussian distributed variables $\mathbf{w}$, it itself follows a Gaussian distribution. It is therefore entirely defined by its mean $m[\mathbf{y}]$ and covariance $cov[\mathbf{y}]$:

$$\begin{aligned} m[\mathbf{y}] &= \mathbf{\Phi} m[\mathbf{w}] = \mathbf{0} \\ cov[\mathbf{y}] &= m[\mathbf{y}\mathbf{y}^T] = \mathbf{\Phi} m[\mathbf{w}\mathbf{w}^T]\mathbf{\Phi}^T = \mathbf{K}, \end{aligned} \tag{2.11}$$

where $\mathbf{K}$ is the Gram matrix with elements:

$$\mathbf{K}_{nm} = k(\mathbf{x}_n, \mathbf{x}_m) = \sigma^2 \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m), \tag{2.12}$$

where we define $k(\mathbf{x}, \mathbf{x}')$ as the kernel function between two inputs $\mathbf{x}$ and $\mathbf{x}'$, and σ^2 is the variance of the Gaussian prior on the weights.

In our example, the collection of function values $\mathbf{y}$ constitute a collection of random variables which follow a Gaussian distribution of mean 0 and for which the covariance matrix is defined by a particular *kernel* function (Equations 2.11 and 2.12). The marginalization property of Gaussian distributions implies that any subset of $\mathbf{y} = \{y_1, ..y_N\}$ is also described as a Gaussian of mean 0 and relevant sub covariance matrix. This model hence defines a probability distribution over functions $f(\mathbf{x})$ evaluated at an arbitrary N number of points $\mathbf{y}$ in a consistent manner. We write the Gaussian Process as:

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), f(\mathbf{x}, \mathbf{x}')).$$

With the GP modelling approach, we no longer need to define a parametric form for f, but rather we need to choose a prior probability distribution over

functions directly. This distribution reflects our prior expectation of the value of $f(\mathbf{x})$ as defined by the *mean function*, and our prior expectation of how "smooth" the functions are expected to be as defined by the *covariance function*.

Mean Function

In many applications we do not have any prior knowledge of what the mean of $f(\mathbf{x})$ is expected to be and choose it to be 0. This is similar to the choice of a zero-mean Gaussian for the prior over weights in the parametric basis function approach. In our work of Chapter 3, we set the mean function to be zero, to capture the assumption that we cannot localise in the absence of information. As a result, the behaviour of the GP is defined entirely by the covariance function, which encodes how two input feature vectors, $\mathbf{x}$ and $\mathbf{x}'$, relate to each other.

Covariance Function

The covariance function of a GP is evaluated at any two values of $\mathbf{x}$ as given by a *kernel function*:

$$m(f(\mathbf{x}), f(\mathbf{x}')) = k(\mathbf{x}, \mathbf{x}')$$

In the case where the mean is 0, the Gaussian Process is entirely determined by this kernel function. In the GP example we introduced as defined by the linear regression model, the kernel function is given by Equation 2.12.

Rather than choosing a basis function ϕ, one can directly define a kernel function as long as the corresponding Gram matrix be positive semidefinite to correspond to a valid positive definite covariance function (i.e symmetric with positive eigenvalues). The kernel function defines a similarity metric between two points in the input space and reflects how similar two outputs should be depending on how close they are in input space. The art of modeling with GPs resides in choosing an appropriate covariance function, and developing more expressive covariance functions. The choice of covariance depends on the prior assumption one has about the problem at hand, related to smoothness, non-stationarity, periodicity etc. As sums and

products of GPs are also GPs, more elaborate covariance functions can be built by combining different simpler covariance functions.

Rather than choosing a fixed covariance function, we can consider a parametric family of functions and infer parameter values from the data. The most widely-used kernel within the kernel learning community is the squared exponential kernel:

$$k(x, x') = exp(\frac{-|\mathbf{x} - \mathbf{x}'|^2}{2\mathbf{l}^2}),$$

where parameter $\mathbf{l}$ defines the *characteristic length scales* of the input feature dimensions. In loose terms, the length scales specify how far one needs to move along a feature dimension in input space for the function values to become uncorrelated. In this way, the inverse of the length-scale determines how relevant an input is. The squared exponential kernel is however infinitely differentiable, which results in a very smooth function. This may be an unreasonable modelling assumption for many physical processes [Stein, 2012] and the Matérn kernel is often a preferred choice:

$$k(x, x') = \frac{1}{\Gamma(\nu)2^{\nu-1}}\Big[\frac{\sqrt{2\nu}}{l}|\mathbf{x} - \mathbf{x}'|\Big]^{\nu} K_{\nu}\Big(\frac{\sqrt{2\nu}}{l}|\mathbf{x} - \mathbf{x}'|\Big),$$

where K_{ν} is the modified Bessel function of second kind of order ν, and l is the characteristic length scale. The hyperparameter ν controls the degree of smoothness and Matérn functions are $[\nu - 1]$ times differentiable. The function becomes particularly simple when ν is half integer, and $\nu = 3/2$ or $\nu = 5/2$ represent interesting options for more realistic smoothness characteristics of physical processes:

$$k_{\nu=3/2}(\mathbf{x}, \mathbf{x}') = (1 + \frac{\sqrt{3}|\mathbf{x} - \mathbf{x}'|}{l})exp(-\frac{\sqrt{3}|\mathbf{x} - \mathbf{x}'|}{l}),$$
$$k_{\nu=5/2}(\mathbf{x}, \mathbf{x}') = (1 + \frac{\sqrt{5}|\mathbf{x} - \mathbf{x}'|}{l} + \frac{5|\mathbf{x} - \mathbf{x}'|^2}{3l^2})exp(-\frac{\sqrt{5}|\mathbf{x} - \mathbf{x}'|}{l}).$$

Learning in Gaussian Processes concerns finding the structural form of the covariance function and secondly its parameters.

Learning Hyperparameters

Though choosing properties of the covariance function can be done from the context of the task at hand, it is typically more difficult to determine what constitutes "sound" values for the model hyperparameters θ such as length scale. Values of hyperparameters (which constitutes a form of model selection) can be learned by considering the marginal likelihood of the training data:

$$p(\mathbf{y}|\mathbf{X},\theta) = \int p(\mathbf{y}|\mathbf{f},\mathbf{X})p(\mathbf{f}|\mathbf{X})d\mathbf{f}.$$

This marginal likelihood integrates the likelihood times the prior over all possible function values $\mathbf{f}$. Under the Gaussian process model the prior is Gaussian $\mathbf{f}|\mathbf{X} \sim \mathcal{N}(\mathbf{0}, K)$:

$$\log p(\mathbf{f}|\mathbf{X},\theta) = -\frac{1}{2}\mathbf{f}^T K^{-1}\mathbf{f} - \frac{1}{2}\log |K| - \frac{N}{2}\log 2\pi,$$

and the likelihood is a factorized Gaussian $\mathbf{y}|\mathbf{f} \sim \mathcal{N}(\mathbf{f}, \sigma^2 I)$ leading to the integrated log marginal likelihood:

$$\log p(\mathbf{y}|\mathbf{X},\theta) = -\frac{1}{2}\mathbf{y}^T(K+\sigma^2 I)^{-1})\mathbf{y} - \frac{1}{2}\log |K+\sigma^2 I| - \frac{N}{2}\log 2\pi. \tag{2.13}$$

Equation 3.3 can be maximised with respect to individual hyperparameters using gradient based optimizers. Once hyperparameters are learned, we can make predictions for a new incoming datapoint $\mathbf{x}^*$.

Inference

We wish to use a GP to make predictions $\mathbf{f}^*$ for new test datapoints $\mathbf{X}^*$. From the definition of the Gaussian process, the test output vector $\mathbf{f}^*$ along with the training set outputs $\mathbf{f}$ have a joint Gaussian distribution:

$$\begin{bmatrix}\mathbf{f}\\ \mathbf{f}^*\end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}\ ,\ \begin{bmatrix}K(\mathbf{X},\mathbf{X}) & K(\mathbf{X},\mathbf{X}^*)\\ K(\mathbf{X}^*,\mathbf{X}) & K(\mathbf{X}^*,\mathbf{X}^*)\end{bmatrix}\right). \tag{2.14}$$

In a more realistic situation we observe noisy versions of the function output:

$y = f(\mathbf{x}) + \epsilon$. Assuming additive noise, we obtain the following joint distribution between the noisy observations $\mathbf{y}$ and function values at test point $\mathbf{f}^*$ under the prior:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}^* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0} \ , \ \begin{bmatrix} K(\mathbf{X}, \mathbf{X}) + \sigma^2 I & K(\mathbf{X}, \mathbf{X}^*) \\ K(\mathbf{X}^*, \mathbf{X}) & K(\mathbf{X}^*, \mathbf{X}^*) \end{bmatrix} \right). \tag{2.15}$$

From Equation 2.15 we obtain the mean and covariance of the conditional distribution $\mathbf{f}^*|\mathbf{X}, \mathbf{y}, \mathbf{X}^*$ which are the key results of the Gaussian process regression:

$$\begin{aligned} &\mathbf{f}^*|\mathbf{X}, \mathbf{y}, \mathbf{X}^* \sim \mathcal{N}(\overline{\mathbf{f}^*}, \text{cov}(\mathbf{f}^*)), \text{where} \\ &\qquad \overline{\mathbf{f}^*} = K(\mathbf{X}^*, \mathbf{X})[K(\mathbf{X}, \mathbf{X}) + \sigma^2 I]^{-1}\mathbf{y}, \\ &\text{cov}(\mathbf{f}^*) = K(\mathbf{X}^*, \mathbf{X}^*) - K(\mathbf{X}^*, \mathbf{X})[K(\mathbf{X}, \mathbf{X}) + \sigma^2 I]^{-1}K(\mathbf{X}, \mathbf{X}^*). \end{aligned} \tag{2.16}$$

2.2.3 Dynamic Bayes for State Estimation

In this thesis, we are interested in developing a perception system's situational awareness beyond the instantaneous and occluded information provided by its sensors. Real world environments can be viewed as stochastic dynamical systems which possess an internal state that describes all characteristics of that world. One aspect of the state of the world that is of particular interest to a robot that needs to navigate through it, is the location and movement through time of objects present in the local environment.

Unfortunately, not all such characteristics of interest relating to scene understanding are directly measurable by onboard sensors. Cameras, LIDARs or radars cannot directly provide the intentions of cars, pedestrians, cyclists and their interactions with the environment. Furthermore, not all objects are observable at every given time, as they can temporarily become occluded from the sensor view in busy dynamic environments such as urban areas.

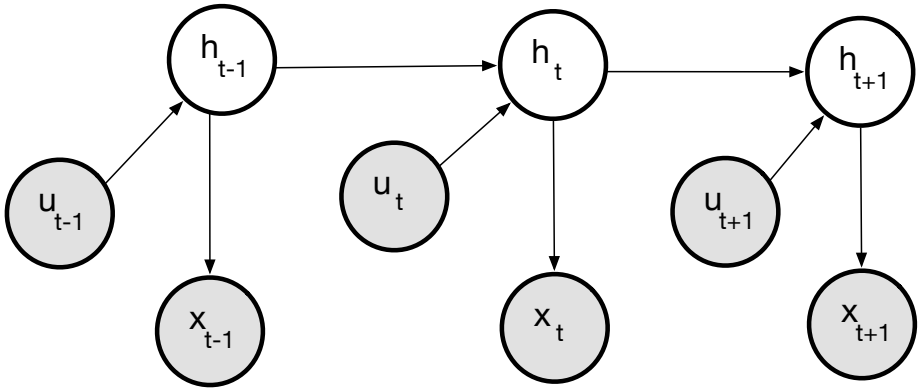


Figure 2.4: The graphical model of a dynamic Bayes network charaterizes the evolution of state (h), control (u) and observation (x) variables in a dynamical system [Thrun et al., 2005].

If a perception system were able to estimate the aspect of the world state relevant to predicting scene dynamics, it could forward propagate this model through time in order to infer future unoccluded scene occupancy. This would be valuable information to a planning system. A common challenge for robotics therefore consists in estimating these world state characteristics from a sequence of observations collected by their onbard sensors [Thrun et al., 2005]. To model the dynamic state of the world, we turn to the Dynamic Bayes network formulation.

Dynamic Bayes network

The challenge of estimating the state of the environment is commonly framed as a *dynamic Bayes network* (DBN) represented in Figure 2.4.

In this framework, at any given timestep t, we consider the world to possess an internal state $\mathbf{h}_t$ which evolves probabilistically over time. The notation $\mathbf{h}$ refers to the fact that it is only partially observable by a sensor or "hidden". The system is stochastic as measurements are noisy and only provide partial observation of the state. The robot therefore can only maintain a *belief* of what the state of the environment is, which takes the form of a probability distribution over possible values. A robot can acquire information about the environment using its sensors, $\mathbf{x}_t$, and it can influence the environment through its actuators, by moving in the environment, $\mathbf{u}_t$. All three variables are modelled as random variables, and the

main objective is to be able to predict the next state of the world $p(\mathbf{h}_t)$ given previous observations and estimates of the state:

$$p(\mathbf{h}_t|\mathbf{h}_{0:t-1}, \mathbf{x}_{1:t-1}, \mathbf{u}_{1:t}).$$

For practical reasons, we commonly assume that $\mathbf{h}_t$ satisfies the first order *Markov* property, that is to say that $\mathbf{h}_t$ is a sufficient summary of all observation and control up to time t. Under this assumption, the *state transition probability* simplifies to:

$$p(\mathbf{h}_t|\mathbf{h}_{0:t-1}, \mathbf{x}_{1:t-1}, \mathbf{u}_{1:t}) = p(\mathbf{h}_t|\mathbf{h}_{t-1}, \mathbf{u}_t).$$

To help us estimate this process, we can model the observation $\mathbf{x}_t$, which given the Markov property can be entirely explained by the state h_t at time t:

$$p(\mathbf{x}_t|\mathbf{h}_{0:t}, \mathbf{x}_{1:t-1}, \mathbf{u}_{1:t}) = p(\mathbf{x}_t|\mathbf{h}_t).$$

The *state transition* probability and *measurement* probability together describe the dynamical stochastic system of the robot and its environment.
As mentioned, because the state of the world cannot be directly measured, the robot can only develop an internal *belief* of what the underlying true state of the world is. This belief $bel(\mathbf{h}_t)$ represents a probability distribution over every possible hypothesis with regards to the true state $\mathbf{h}_t$. Estimating the state of the world becomes one of predicting the robot's belief of what the true state of the world is after taking action $\mathbf{u}_t$:

$$\overline{bel}(\mathbf{h}_t) = p(\mathbf{h}_t|\mathbf{x}_{1:t-1}, \mathbf{u}_{1:t})$$

and updating that belief after incorporating measurement $\mathbf{x}_t$:

$$bel(\mathbf{h}_t) = p(\mathbf{h}_t|\mathbf{x}_{1:t}, \mathbf{u}_{1:t})$$

Belief estimation A common approach to estimating the robot's belief is the *Bayes* filter which is a 2-step recursive algorithm to update the belief $bel(\mathbf{h}_t)$. In a first *prediction* step, the previous belief $\mathbf{h}_{t-1}$ is updated considering the robot's control $\mathbf{u}_t$:

$$\overline{bel}(\mathbf{h}_t) = \int p(\mathbf{h}_t|\mathbf{h}_{t-1}, \mathbf{u}_t) bel(\mathbf{h}_{t-1}) d\mathbf{h}_{t-1}.$$

This requires summing over the product of two distributions: the prior belief at $t-1$, and the probability that control u_t induces a transition from h_{t-1} and h_t. In a second *measurement update* step, the belief is multiplied by the probability that the measurement may have been observed:

$$bel(\mathbf{h}_t) = \eta p(\mathbf{x}_t|\mathbf{h}_t)\overline{bel}(\mathbf{h}_t)$$

where η is a normalizing constant for the distribution to integrate to 1.

Successful recursive state estimation based on the Bayes filter requires choosing a suitable state representation and suitable and tractable distributions for the state transition and observation models [Särkkä, 2013]. Common effective tractable choices for representing these processes assuming continuous latent variables are Gaussian filters (e.g linear processes with the Kalman filter [Kalman, 1960], and their numerous extensions to non-linear processes [Musoff and Zarchan, 2009], [Julier et al., 1995]) or non-parametric approaches such as *particle filters* [Gordon et al., 1993]. Alternatively the *hidden Markov model* provides an important model for processes where latent variables are discrete.

We are interested in predicting the unoccluded occupancy scene $\mathbf{y}_t$ around a robot given a sequence of observations $\mathbf{x}_{t=1:t}$ provided by a range sensor: $p(\mathbf{y}_t|\mathbf{x}_{1:t})$.

We formulate this problem using the *Deep Tracking* framework introduced in [Ondruska and Posner, 2016] which models the underlying world state as a hidden Markov model, and approximates the transition and observation processes as an end-to-end learning task using highly expressive recurrent neural networks.

We next present neural networks as powerful tools for approximating complex distributions, and then detail the implementation of the deep tracking solution of Ondruska and Posner [2016] to Bayesian filtering.

2.2.4 Neural Networks for Function Approximation

A neural network is a non-linear *parametric* function of the input variables $\mathbf{x}$ parametrised by a vector of adjustable weights $\mathbf{w}$:

$$y = \mathcal{F}_{\mathbf{w}}(\mathbf{x})$$

In a probabilistic interpretation, the output of the neural network can be seen as approximating the underlying predictive distribution $p(y|\mathbf{x})$:

$$p(y|\mathbf{x}, \mathbf{w}) = \mathcal{F}_{\mathbf{w}}(\mathbf{x})$$

If the N observed data points in the training set are considered independent and identically distributed, we obtain the following likelihood function:

$$p(\mathbf{Y}|\mathcal{D}) = \prod_{i=1}^{N} p(y_i|\mathbf{x_i}, \mathbf{w}) = \prod_{i=1}^{N} \mathcal{F}_{\mathbf{w}}(\mathbf{x_i}) \tag{2.17}$$

Objective Function

How well $\mathcal{F}$ explains the data is assessed by choosing an appropriate *objective function* to which is associated a loss function $\mathcal{L}_{\mathcal{D}}$. One can perform *maximum likelihood estimation* or MLE, and find a set of parameters $\mathbf{w}^*$ that maximizes the likelihood function in Equation 2.17. Equivalently, as optimisers generally seek a minimum rather than maximum, we can consider the corresponding *loss function* $\mathcal{L}_{\mathcal{D}}(\mathbf{w})$ corresponding to the negative log-likelihood of the data $\mathcal{D}$ given the model:

$$\mathcal{L}_D(\mathbf{w}) = -log\ \ p(\mathbf{Y}|\mathbf{X}, \mathbf{w}) = -\sum_{i=1}^{N} log(p(y_i|\mathbf{x_i}, \mathbf{w})),$$

with the corresponding *maximum likelihood estimate*:

$$\mathbf{w}^* = \arg_w \ \min \mathcal{L}_D(\mathbf{w}).$$

Once parameters are learned, the training data is discarded and new predictions are made using the learned parameters $\mathbf{w}^*$.

Regularisation More complex functions can potentially better fit the data, but also risk *overfitting* the training set. To prevent this, the *objective function* can be modified to incorporate a regularisation term over the weights $\mathcal{L}_W(\mathbf{w})$:

$$\mathcal{L}(\mathbf{w}) = \mathcal{L}_{\mathcal{D}}(\mathbf{w}) + \lambda \mathcal{L}_W(\mathbf{w}). \tag{2.18}$$

In a more Bayesian view, this corresponds to considering a prior distribution over the weights parametrised by parameter α, $p(\mathbf{w}|\alpha)$, and considering the posterior distribution over weights $\mathcal{L}(\mathbf{w})$:

$$posterior \ \propto likelihood \times prior$$

$$p(\mathbf{w}|\mathbf{X}, \mathbf{Y}, \alpha) \propto p(\mathbf{Y}|\mathbf{X}, \mathbf{w}) p(\mathbf{w}|\alpha).$$

The learning objective then becomes one of learning the *maximum a posteriori* distribution over weights or MAP. Taking the negative logarithm of the objective and considering a Gaussian prior of form $\mathcal{N}(0, \alpha^{-1}\mathbf{I})$, the objective boils down to minimising the following loss:

$$\mathcal{L}(\mathbf{w}) = -log \ p(\mathbf{Y}|\mathbf{X}, \mathbf{w}) + \frac{\alpha}{2}\mathbf{w}^T\mathbf{w}.$$

We recognise the updated objective of Equation 2.18 with regularization parameter $\lambda = \frac{\alpha}{2}$ reflecting the balance between finding a model that explains the data well ($\mathcal{L}_{\mathcal{D}}(\mathbf{w})$) whilst preventing overfitting ($\mathcal{L}_W(\mathbf{w})$). This type of regulariser is also referred to as an L_2 loss related to the Gaussian squared exponential, or as *weight decay* in the particular case of neural networks.

Another way of preventing neural network overfitting is to monitor the data fit

loss function $\mathcal{L}_{\mathcal{D}}(\mathbf{w})$ measured on a *test* set during training. As described below, the learning process in a neural network is an iterative process reducing the error function on the training data. It is often observed that whilst the error function slowly decreases on the training data, if evaluated on an independent *test* set it initially decreases before starting to increase. Increasing errors on a separate dataset (also referred to as a *validation set*) can reflect overfitting on the training set. To prevent this, training can be stopped when the error on the test set is at its lowest point. This is known as *early stopping*. We choose early stopping as a method to prevent overfitting in our experiments of Chapters 4 through 6.

Network Architecture

Typical parametric models for classification consist in a linear combination of fixed nonlinear basis functions ϕ_j:

$$y(x, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j\right), \tag{2.19}$$

where $f()$ is a nonlinear activation function. A neural network allows to extend this model by making the basis functions no longer fixed, but dependant on parameters that can be adjusted along with the weights $\{w_j\}$. A neural network is built around activation functions that take the form of Equation 2.19, and where the basis functions are parametric non-linear combinations of input activations which themselves follow the same form as 2.19. Essentially, the model takes the generic compositional form:

$$y(x, \mathbf{w}) = \mathcal{F}^l_{w_l}(\mathcal{F}^{l-1}_{w_{l-1}}(...\mathcal{F}^1_{w_1}(\mathbf{x}))),$$

where $\mathcal{F}^l_{wl}$ takes the form of 2.19.

Figure 2.5 on the following page represents a simple 2-layer feed-forward network whose overall network function can be written:

$$y(x, \mathbf{w}) = f\left(\sum_{j=1}^{j=K} w_j^{(2)}\left(\sum_{i=1}^{D} w_{j,i}^{(1)} x_i + w_{j,0}^{(1)}\right) + w_0^{(2)}\right)$$

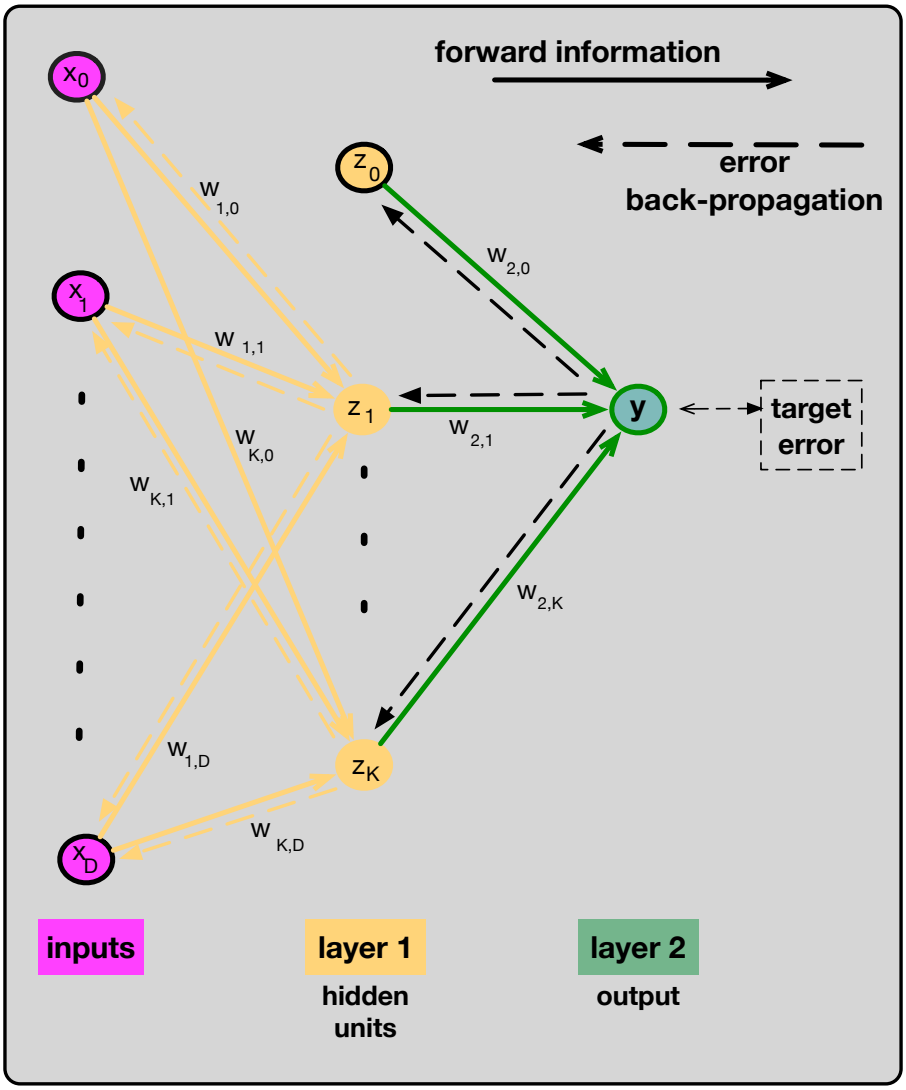


Figure 2.5: Example of a 2-layer feed-forward neural network graph. Each unit is a non-linear combination of units in lower layers. The gradient of the error function between output and target is backpropagated through the network.

Every stage of functional transformation is referred to as a *layer* and layers that are between the input layer and output layer are referred to as *hidden*. The output is compared with the target value and an error computed. Gradients of the error function are propagated along the network connections back to the input during training as we describe later in this section. Every activation in one layer nominally has the same non linear activation function as every other unit in the same layer.

The network connections, number of layers and activation functions are examples of *architectural choices* that need to be made and that play an important role in

the success (or failure) of the learning process. Finding a good architecture (in terms of performance) generally requires trial and error, ablation studies, best practices from prior work, and intuition.

Though Figure 2.5 on the preceding page represents a fully-connected network, we opt for a *fully convolutional* network architecture where a unit activation in layer l only depends on spatially neighbouring units in layer $l-1$ (rather than all units of layer $l-1$ like in a fully-connected network). The graph network is said to be *sparse*, the *receptive field* of units is local and specified by the *kernel size* of the convolutional filters. This allows for a reduction in the overall number of parameters, and for detecting features regardless of their position in the layer unlike fully connected architectures.

We also make use of *recurrent neural networks* [Medsker and Jain, 2001] which are a way of adpating feed-forward networks to sequential data. In recurrent architectures, the hidden units are connected through time as illustrated in Figure 2.6.

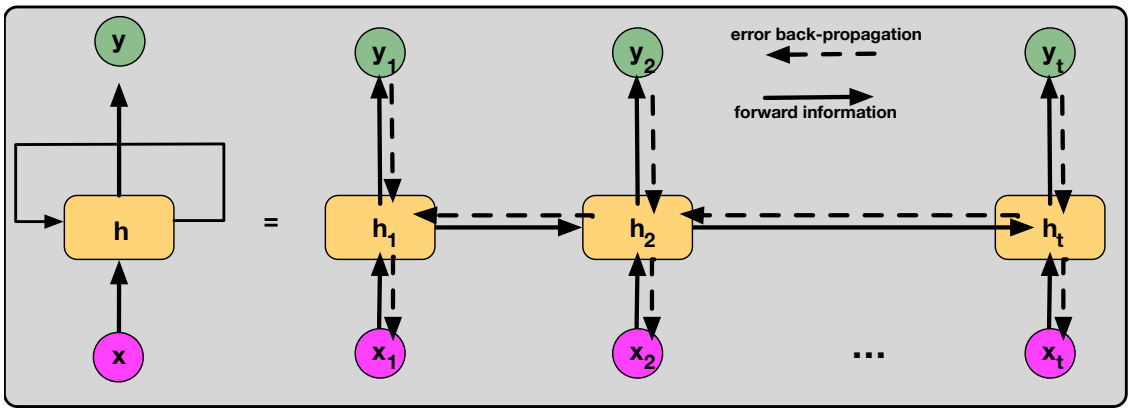


Figure 2.6: Schematic representation of a recurrent neural network for modelling sequential data. Information is propagated forward "through time" via the hidden layers h_t which get updated with every new input x_t and which act as a network memory.

Any graph structure can potentially be used provided there are no closed directed cycle and the architecture remains *feed-forward*. We refer the reader to [Bengio et al., 2009] for further architectural choices. The last layer of the designed architecture is then chosen to suit the learning problem. In a probabilistic interpretation we assume that the model output predictions $p = \mathcal{F}_{\mathbf{w}}(x)$ approximates the predictive

distribution $p(y|x, \mathbf{w})$. Depending on whether the variable to predict is continuous or categorical, the problem is framed as regression or classification.

Regression

If the output variable is continuous, the network can be trained to approximate the mean of the distribution:

$$\mu = \mathcal{F}_{\mathbf{w}}(x). \tag{2.20}$$

If the observed data is assumed to be noisy independent observations of this mean with Gaussian noise, the objective function results in maximising:

$$p(y|x, \mathbf{w}) = \prod_{\mathcal{D}} \mathcal{N}(y|\mu, 1), \tag{2.21}$$

for which the canonical loss is the L_2-loss:

$$\mathcal{L}_D(\mathbf{w}) = -\frac{1}{2} \sum_{\mathcal{D}} (y - \mathcal{F}_{\mathbf{w}}(x))^2. \tag{2.22}$$

The error term from this loss is backpropagated through the individual network weights at training time via the corresponding gradient:

$$\frac{\partial \mathcal{L}_{\mathcal{D}}}{\partial w} = \sum_{\mathcal{D}} (y - \mathcal{F}_{\mathbf{w}}(x)). \frac{\partial \mathcal{F}_{\mathbf{w}}}{\partial w}(x). \tag{2.23}$$

Classification

If the observed variable is a categorical 1 of K classes, the network needs to predict a normalised vector $\mathbf{p} \in [0, 1]^K$. This can be achieved by passing the last real-valued output $\mathbf{a} \in \mathbb{R}^K$ of the network into a normalising *softmax* activation function:

$$\begin{aligned} \mathbf{a} &= \mathcal{F}_{\mathbf{w}}(\mathbf{x}), \\ p_k &= \frac{\exp(a_k)}{\sum_{k=1}^{K} \exp(a_k)}, \end{aligned} \tag{2.24}$$

where index k corresponds to the k-th entry in the K-dimensional vector. If observations are considered independent, the learning objective takes the form of maximising the following predictive distribution:

$$p(\mathbf{y}_K|x,\mathbf{w}) = \prod_{\mathcal{D}}\prod_{k} p_k^{y_k}, \tag{2.25}$$

where the output is reshaped as a vector $\mathbf{y}_K$ of length K with entries $y_k \in [0,1]$, representing the one hot encoding of the desired class: $\mathbf{y}_K = [0,0,...,1,...,0] \in \mathbb{R}^K$. The corresponding loss function becomes:

$$\mathcal{L}_D(\mathbf{w}) = -\sum_{\mathcal{D}}\sum_{k} y_k \log p_k. \tag{2.26}$$

The corresponding gradient of the loss with respect to the weights takes the form:

$$\frac{\partial \mathcal{L}_\mathcal{D}}{\partial w} = -\sum_{\mathcal{D}}\sum_{k} \frac{y_k}{p_k}.\frac{\partial p_k}{\partial w}, \tag{2.27}$$

with corresponding backpropagation through the softmax function:

$$\frac{\partial p_k}{\partial w} = -p_k.(1-p_k), \frac{\partial a_k}{\partial w}. \tag{2.28}$$

Binary classification In the particular case where the $K = 2$, the problem is framed as one of *binary classification* where output y is commonly considered to be of value 1 or 0. The real-valued network output $\mathbf{a}$ is now of length $K = 2$ and can be normalised by the *sigmoid* activation function:

$$\begin{aligned} \mathbf{a} &= \mathcal{F}_{\mathbf{w},l}(\mathbf{x}), \\ p_k &= \frac{1}{1+e^{-a_k}}. \end{aligned} \tag{2.29}$$

if the probability of class 1 is p, the predictive distribution takes the form:

$$p(y|x,\mathbf{w}) = \prod_{D} p^y.(1-p)^y, \tag{2.30}$$

with corresponding canonical *binary cross-entropy* loss function:

$$\mathcal{L}_\mathcal{D}(\mathbf{w}) = -\sum_{\mathcal{D}} y\log(p) + (1-y)\log(1-p). \tag{2.31}$$

The gradients for backpropagtion take the form:

$$\frac{\partial \mathcal{L}_{\mathcal{D}}}{\partial w} = -\sum_{\mathcal{D}} \frac{y-p}{p.(1-p)}.\frac{\partial p}{\partial w}, \tag{2.32}$$

with gradient through the sigmoid:

$$\frac{\partial p}{\partial w} = p.(1-p).\frac{\partial a_k}{\partial w}. \tag{2.33}$$

We consider both binary and multi-class classification in our work.

Training

Finally, neural networks are trained using basic gradient descent where the weights are updated using the opposite of the gradient of the loss function $-\frac{\partial \mathcal{L}}{\partial \mathbf{w}}$ with respect to the weights, using the chain rule:

$$\frac{\mathcal{L}}{\partial w_l} = \frac{\mathcal{L}}{a_{l+n}}.\frac{\partial a_{l+n}}{\partial a_{l+n-1}}....\frac{a_l}{w_l},$$

where a_l represent the intermediate outputs and activation functions of the network layers.

As in practice the non-linearity of the network function causes the error to be non convex, weights are updated using an iterative process. The simplest approach consists in choosing some initial value for the weights $\mathbf{w}_0$ and moving through weight space along the negative of the gradient of the loss w.r.t to the parameters:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \frac{d\mathcal{L}_t}{d\mathbf{w}_t}, \tag{2.34}$$

where $\eta > 0$ is the *learning rate*. At each step t of the iteration, the gradient is evaluated for the updated weight values and the process re-iterated. In practice optimisation is performed using *stochastic gradient descent* [Bottou, 2010] on mini-batches of data. At each *training epoch*, the data is randomly distributed into N mini-batches, and the optimisation process run N times, once on each batch. This allows more efficient handling of redundancy in the data and potentially avoids local-minima compared to training on the entire dataset at once. We use the *Adagrad* [Duchi et al., 2011] optimiser which allows for more efficient learning, for all experiments in this thesis. The algorithm adapts the learning rate for each

parameter at every timestep, producing higher rates for rare occurring features, and low rates for high occurring features. The learning rate is adapted using the past gradients of the parameters:

$$\mathbf{w}_{t+1,j} = \mathbf{w}_{t,j} - \frac{\eta}{\sqrt{G_{t,j} + \epsilon}} \frac{d\mathcal{L}_t}{d\mathbf{w}_{t,j}},$$

where $\epsilon \approx 1.e^{-8}$ ensures numerical stability, and $G_{t,j}$ corresponds to the sum of squares of past gradients until t:

$$G_{t,j} = \sum_{s=1}^{t} (\frac{d\mathcal{L}_s}{d\mathbf{w}_{s,j}})^2.$$

2.2.5 Deep Tracking Formulation

We close this section on inference by presenting the *Deep Tracking* framework proposed by Ondruska and Posner [2016] as a learnable solution to the dynamic Bayes formulation of tracking.

We frame our state estimation problem as one of uncovering the true, *unoccluded* state of the world y, in terms of a 2D occupancy grid $\mathbf{y}_t$, given a sequence of *partially observed* states of the environment $\mathbf{x}_{1:t}$ computed directly from raw LIDAR measurements. We consider range sensors as they provide straightforward information with regards to scene occupancy, and they can be readily converted into a 2D occupancy and visibility grid centered on the robot for use by a planning system.

Deep tracking [Ondruska and Posner, 2016] is a framework initially introduced to model the conditional distribution $P(\mathbf{y}_t|\mathbf{x}_{1:t})$ using a recurrent neural network [Medsker and Jain, 2001]. Motivated by recursive state estimation as presented earlier in Section 2.2.3, the underlying process is assumed to exhibit a first-order Markov property. The graphical model for the state transition and observation processes is represented in Figure 2.7 on the next page.

That is, the latent state at time t, denoted by $\mathbf{h}_t$, captures the complete information required to predict the output $\mathbf{y}_t$, which may denote scene appearance and dynamics, locations of all objects, their shapes, and velocities.

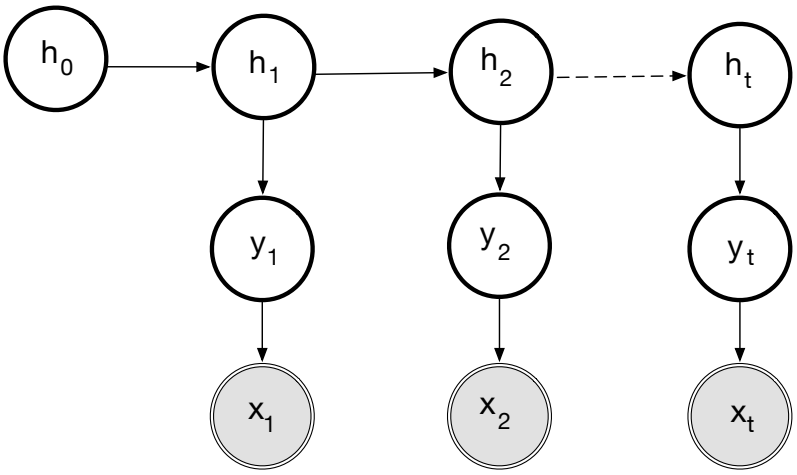


Figure 2.7: The *deep tracking* graphical model assumes an underlying Hidden Markov Process for the world state h_t. The world occupancy y_t resulting from the state is only partially observed by sensor measurement x_t [Ondruska and Posner, 2016].

Thus we have:

$$p(\mathbf{y}_t|\mathbf{x}_{1:t}) = p(\mathbf{y}_t|\mathbf{h}_t). \tag{2.35}$$

To avoid having to specify the complex and unknown distributions involved, they use an expressive neural network to approximate state transition and observation models:

$$\begin{aligned} \mathbf{h}_t &= \mathcal{F}_1(\mathbf{h}_{t-1}, \mathbf{x}_t) \\ p(\mathbf{y}_t|\mathbf{x}_{1:t}) &= \mathcal{F}_2(\mathbf{h}_t). \end{aligned} \tag{2.36}$$

Both these functions are trained jointly as parts of a single neural network. The prediction and update steps in Equations 2.35 and 2.36 can thus be performed repeatedly as part of a recurrent neural network that updates the current belief, or *network memory* state, $\mathbf{h}_t$, and uses it to predict the unoccluded output $\mathbf{y}_t$. In this way, the model can be used for online filtering of the sensor input.

Problem Formulation The input observation at each time step t, $\mathbf{x}_t \in \{0,1\}^{2\times M\times M}$, is represented as a pair of discretised 2D binary grids of size $M \times M$, parallel to the ground and centred on the sensor frame. The first of these matrices encodes *visibility*: whether a cell is directly observable by the sensor at time t; while the second matrix encodes *occupancy*: whether a cell is observed to be free (value of 0) or occupied (value of 1). These two matrices are denoted as $\mathbf{x}_{t,vis}$ and $\mathbf{x}_{t,occ}$ respectively as illustrated in Figure 2.8. The true unoccluded map is another grid $\mathbf{y}_t \in \{0,1\}^{M\times M}$ encoding the occlusion-free state of the world. Thus, the goal is to solve for $p(\mathbf{y}_t|\mathbf{x}_{1:t})$, the conditional probability of the complete, unoccluded state of the world at time t given a sequence of partial observations at all previous time steps.

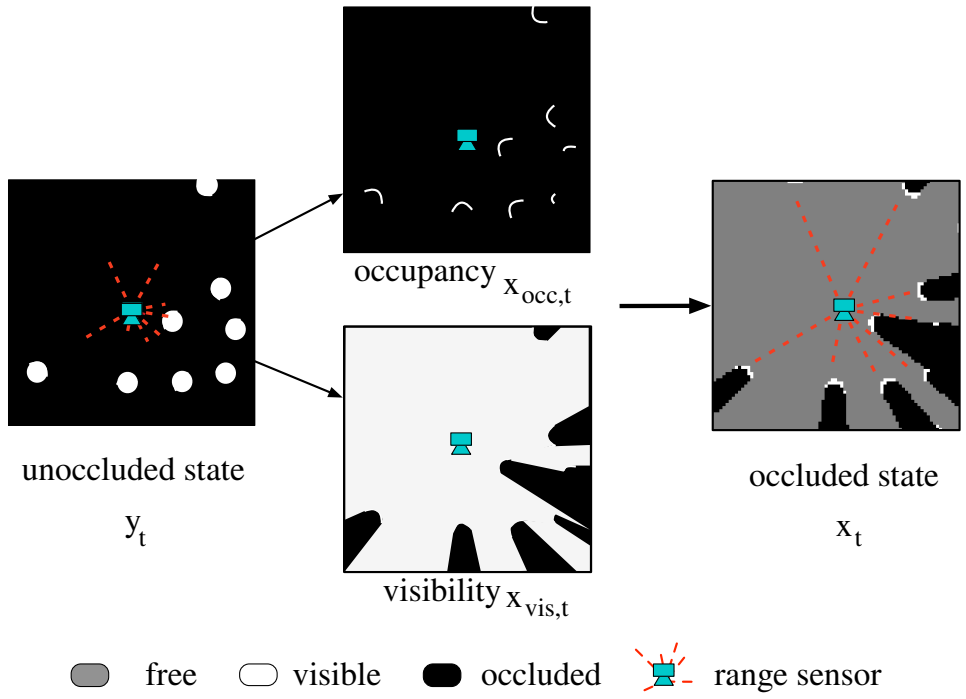


Figure 2.8: At time t, the true unoccluded state of the world y_t is partially observed by a range sensor. The scan can be transformed in a pair of 2D occupancy $x_{occ,t}$ and visibility $x_{vis,t}$ forming an occluded occupancy grid x_t.

Under the same formulation, future states can also be predicted by solving for $p(\mathbf{y}_{t+n}|\mathbf{x}_{1:t})$ after masking the future inputs, i.e. $\mathbf{x}_{t+1:t+n} = \mathbf{0}$. The tracking problem of solving for $p(\mathbf{y}_t|\mathbf{x}_{1:t})$ can also be further extended to predicting the scene semantics $P(\mathbf{c}_t|\mathbf{x}_{1:t})$. This takes the form of predicting one of K objects for each cell. The semantic map $\mathbf{c} \in \{1,...,K\}^{M\times M}$ can reflect object classes

such as *pedestrian, cyclist, car, static background*, or whether an occupied cell belongs to a *dynamic* or *static* object.

Training Since the true state of occluded objects is not known in real-world scenarios, the output ground-truth $\mathbf{y}_t$ is not readily available. To circumvent this issue, the network is trained in a self-supervised fashion. To do this, we consider that predicting the movement of objects under occlusion at time t is similar to predicting a *future* state $\mathbf{y}_{t+n}$ without any future input provided to the network, i.e $\mathbf{x}_{t+n} = \mathbf{0}$. The lack of input observations equates to complete occlusion of the scene. Given that the *observable* ground truth is available, we alter the problem of outputting $\mathbf{y}_{t+n}$ to that of predicting the parts of $\mathbf{y}_{t+n}$ that are directly observable, denoted $\mathbf{y}'_{t+n}$, which is equivalent to the observed input $\mathbf{x}_{t+n}$. Training the network to instead predict the probability of the observable ground truth $p(\mathbf{y}'_{t+n}|\mathbf{x}_{1:t})$ is equivalent to only backpropagating errors in the observable parts of the scene. In other words, the network is trained to correctly predict the subset of the ground-truth occupancy present in the *future input*. The training procedure is illustrated in Figure 2.9. As demonstrated in [Ondruska and Posner, 2016], an important consequence of this training strategy is that, at deployment, the trained network starts to correctly imagine objects and their movement in the occluded regions. This is because the situation with occluded input at deployment is similar to that at training when no input was provided at all for the future time $t + n$, and the network was trained to predict the observable regions.

Performance Metric Metrics typically used to measure the accuracy of a binary classification task consist in *precision* and *recall*. Let the two classes be referred to as the positive and negative class. *Precision* corresponds to the fraction of predictions to the positive class, that are indeed positive (true positive, TP):

$$\text{precision} = \frac{TP}{TP + FP}, \tag{2.37}$$

where FP corresponds to the false positives, that is incorrect predictions of the positive class. This defines how precise the model is at making predictions.

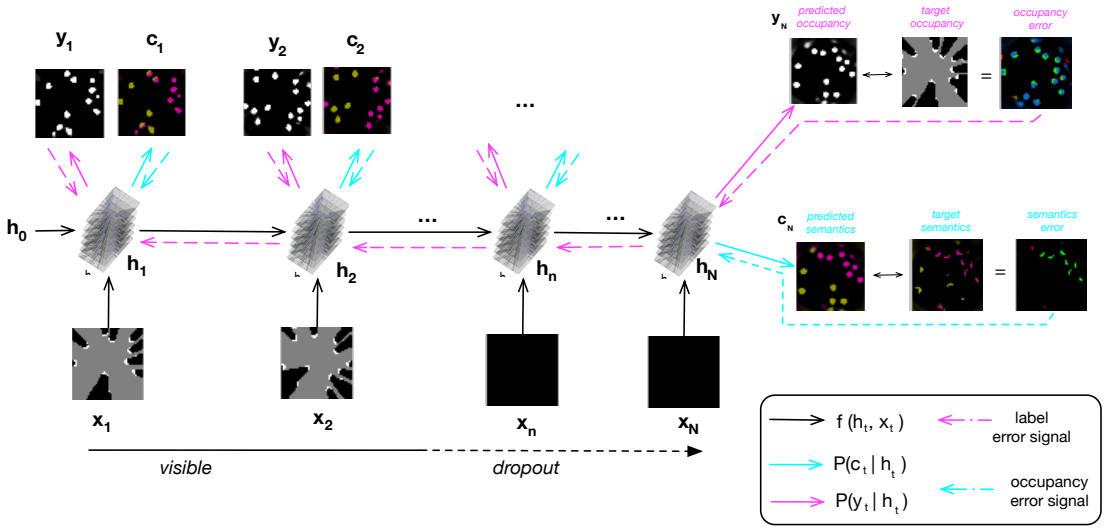


Figure 2.9: Self-supervised training of the recurrent neural network for predicting state occupancy and scene semantics. The network is trained end-to-end to predict future outputs $y_{t+n}|x_{1:t}$ consistent with future observations x_{t+n} by dropping out the input $x_{t+1:n}$. A similar formulation can learn to predict future scene semantics given a sequence of observations $c_{t+n}|x_{1:t}$.

Recall on the other hand reflects the fraction of all positive assignments that were actually predicted to be positive:

$$\text{recall} = \frac{TP}{TP + FN}, \tag{2.38}$$

where FN corresponds to the false negatives, that is positive predictions that were missed. These two measures balance out; if a model predicts all instances as positive, it will obtain a recall of 1 but risks over predicting and obtaining a poor precision score.

In this thesis we make recurrent use of the $F1$ measure, which balances out precision and recall:

$$F1 = \frac{(\text{precision}^{-1} + \text{recall}^{-1})^{-1}}{2} = 2\frac{\text{precision.recall}}{\text{precision} + \text{recall}}. \tag{2.39}$$

2.3 Robot Motion and Localisation

In chapters 3 and 5 we consider a moving robot which needs to obtain estimates of its ego-motion in the world using an onboard stereo-camera. This requires defining *reference frame* conventions and a visual odometry system to describe the position of sensors with respect to one another on the robotic platform (calibration), as well as the transformations between two robot positions in time (ego-motion). Below we present the frameworks and conventions used throughout this thesis.

2.3.1 Reference Frames

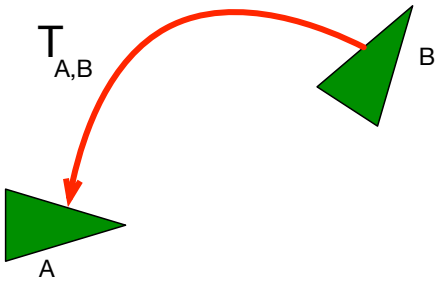


Figure 2.10: Example of an $\mathbb{SE}(2)$ transformation (2D). $T_{A,B}$ represents the position of frame B (source) as seen by frame A (destination) which can be decomposed as a rotation and translation.

Transformations between two reference frames can be defined using the $\mathbb{SE}(3)$ special Euclidean group in 3D space. This group represents the set of all possible rigid-body rotations and translations in 3-dimensional space. Any transformation $T_{\{d,s\}}$ representing the transform expressing the *source* reference frame s into the *destination* reference frame d takes the general form:

$$T_{d,s} = \begin{bmatrix} \mathbf{R}|\mathbf{t} \\ \mathbf{0}|1 \end{bmatrix},$$

where $\mathbf{R}$ is a 3×3 matrix, $\mathbf{t}$ is a 3×1 vector and $\mathbf{0}$ is a 1×3 matrix expressed as column major. Equivalently, $\mathbb{SE}(2)$ represents the set of rigid-body transformations in 2-dimensions as illustrated in Figure 2.10. Whether we consider 2D or 3D

transforms, if a point $\mathbf{p_B}$ is defined with respect to reference B, its position $\mathbf{p_A}$ with respect to reference A can be expressed as:

$$\hat{\mathbf{p_A}} = T_{A,B} \begin{bmatrix} \mathbf{p_B} \\ 1 \end{bmatrix},$$

where $\hat{\mathbf{p_A}}$ is the homogeneous form of $\mathbf{p_A}$:

$$\hat{\mathbf{p_A}} = \begin{bmatrix} \mathbf{p_A} \\ 1 \end{bmatrix}.$$

In the particular context of estimating the robot ego-motion through time, the transform between its reference frame at time step $t+n$ with respect to its position at time step t can be expressed with the following pose chain of matrix multiplications:

$$T_{t,t+n} = T_{t,t+1} \times T_{t+1,t+2} \times ... \times T_{t+n-1,t+n}$$

Convention

Two representations are commonly used to reflect the directions of the x, y, z axes in 3D reference frames. The computer vision community adopts the convention of z forward, x to the right and y down, which we use to project points from the world into the camera frames for example. Our sensor calibration system however adopts the NASA convention of x forward, y to the right, and z downwards. We illustrate both conventions in Figure 2.13 on page 55. Representations can easily be converted from one convention to the other via the transform:

$$T_{CV,NASA} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

converting the NASA convention to the Computer Vision (CV) one.

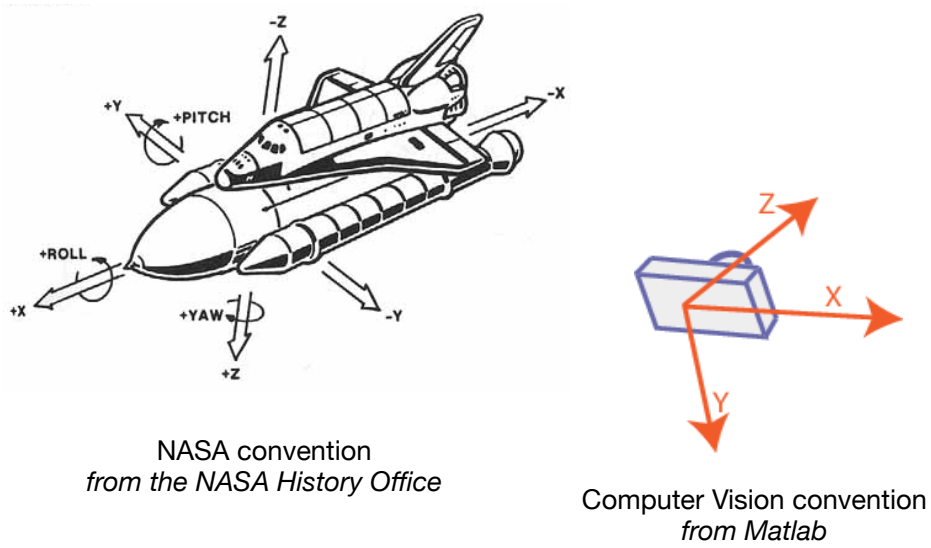


Figure 2.11: The NASA (left) and Computer Vision community (right) coordinate frame conventions.

2.3.2 Visual Odometry

A Visual Odometry (VO) system provides estimates of a robot's motion through the environment using onboard camera imagery. Initially proposed by Matthies [1989], it is able to estimate the full 6-DOF motion of the robot without being affected by wheel slip such as is the case with wheel odometry. Our system uses the implementation by Churchill [2012] and consumes a stream of stereo images provided by a Bumblebee stereo camera mounted on top of a vehicle facing forward (Figure 2.12).

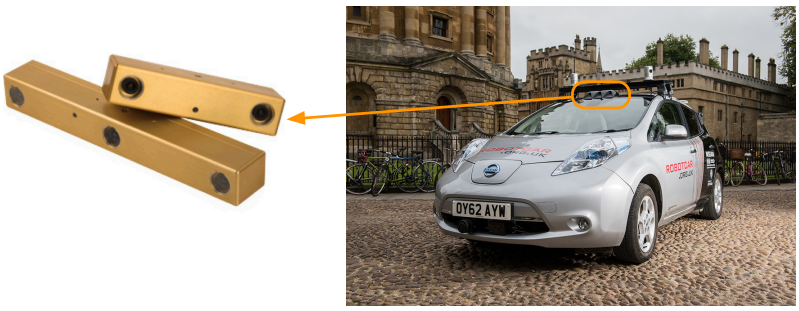


Figure 2.12: Our Visual Odometry system consumes a stream of stereo images provided by a Point Grey Bumblebee XB3 stereo camera mounted on the roof of the vehicle facing forward. Stereo cameras can have two or three sensors providing a flexible baseline between the sensors used for stereo matching.

Visual odometry aims to detect similar 3D landmarks in two subsequent pairs of images I_k and I_{k+1} to determine the 6-DOF transformation $T_{k,k+1}$ representing the pose of source image I_{k+1} in the frame of the destination image I_k (Figure 2.13).

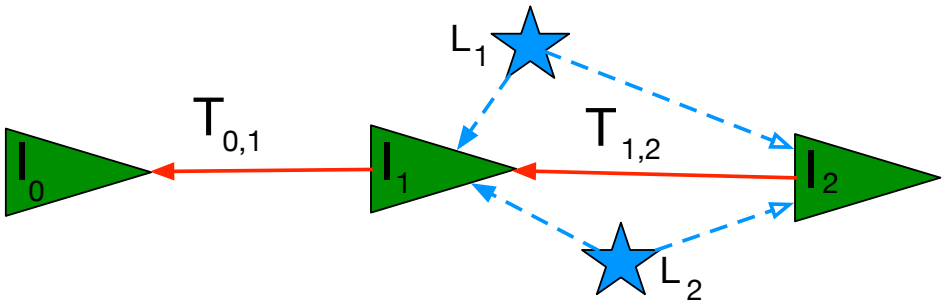


Figure 2.13: Visual Odometry aims to estimate the 6-DOF pose between subsequent images $T_{k,k+1}$ by detecting similar landmarks in both images. Here we illustrate with two landmarks L_1 and L_2 observed in images I_1 and I_2.

The system features a multi-stage pipeline illustrated as a block diagram in Figure 2.14 [Linegar, 2016]:

1. Input image pair: an incoming image pair I_{k+1} is undistorted, rectified and is input to the system

2. Feature detection: salient point features from 3D landmarks are identified in both images. The system detects FAST corners Rosten et al. [2005]

3. Stereo matching: detected features are matched between the left and right image. As images are rectified, any 3D landmark projects on the same vertical pixel co-ordinate in both images. Consequently, matching is performed by taking a reference keypoint in the left image and searching for a match along the corresponding row in the right image. For every FAST corner found in the row, a Sum of Absolute Differences score is computed with respect to the reference keypoint of the left image. The best score is returned as a match. In the absence of FAST features, no match is returned

4. Triangulation: the position of the 3D landmarks is triangulated with the knowledge of the disparity of the observations is both images, the stereo camera baseline, and the intrinsic calibration of the camera.

5. Feature description: a binary BRIEF Calonder et al. [2012] descriptor is computed for the matched features. This produces a representation of the landmark's appearance that can be identified in subsequent images. Feature descriptors should be robust to changes in translation, scale and orientation. While BRIEF does not offer the same level of robustness to rotation such as SIFT and SURF Bay et al. [2006], Lowe [2004] which are gradient-based methods, it is significantly faster at extracting and matching which is valuable for real-time applications on CPU machines.

6. Feature matching: each 2D feature descriptor is associated to a 3D landmark stored in the previous frame at time k. The similarity metric used for BRIEF descriptors is the Hamming distance.

7. Outlier rejection: a transform $T_{k,k+1}$ is calculated using the data associations between 2D features and 3D landmark positions. A RANSAC method is used to determine the transform that produces the greatest number of inliers. This allows robustness to incorrect data associations which may originate from dynamic objects, occlusions and image blur

8. Pose refinement: a non-linear least squares optimisation refines the RANSAC pose by minimising re-projection error of all data associations marked as inliers.

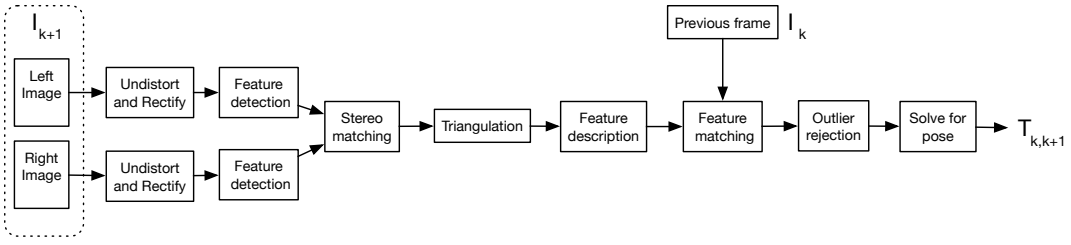


Figure 2.14: The Visual Odometry pipeline features multiple stages of feature matching across space and time and produces a transform between subsequent images. Inspired from [Linegar, 2016].

Stereo VO is popular in robotics as it can be implemented in real time to provide metric pose estimates for every frame of the camera. By building a local topometric map, it can further be used by systems such as Visual Teach and Repeat for enabling robot autonomy as we present in the next section.

2.3.3 Visual Teach and Repeat

The same VO pipeline can be used in the context of localisation such as Visual Teach and Repeat(VT&R). In this framework, a first teach pass consists in driving a robot along a desired path during which the system saves keyframes containing 3D landmarks at regular intervals [Churchill and Newman, 2013, Furgale and Barfoot, 2010], along with the relative transforms between each pairs of subsequent frames. The saved frames and relative transforms form a topometric graph represented in Figure 2.15.

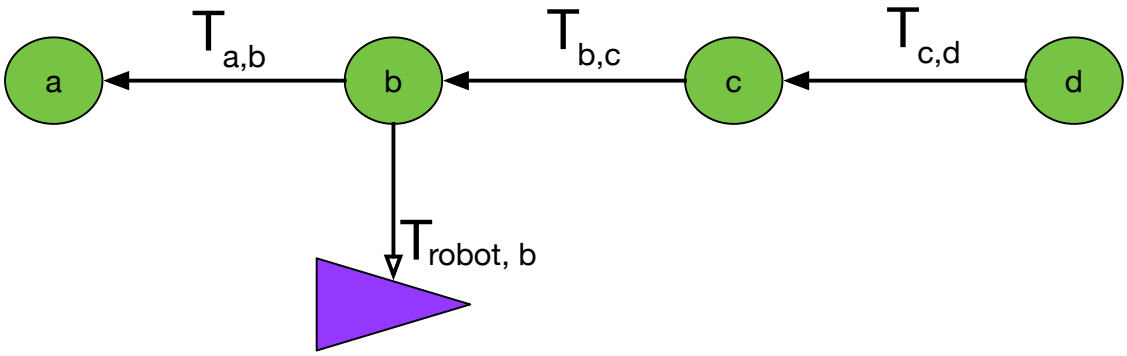


Figure 2.15: The topometric graph of the teach pass consists in nodes representing saved keyframes, and connecting edges representing transforms between the nodes. At repeat time, the robot attempts to compute a 6 DoF pose estimate relative to one of the nodes in the graph.

This means that the saved graph structure is not global where each keyframe would be referenced with respect to a single co-ordinate frame, but rather each keyframe has its own reference frame which is connected to other frames by relative transformations. Localisation at repeat time occurs in this *local* taught map, saved in memory. To repeat the route the system aims to match detected landmarks in live images to those saved in memory keyframes in the nodes of the graph, and to

compute a 6 DoF pose estimate relative to the saved taught path. The position estimate can then be passed on to the controller to retrace the taught trajectory.

The system first locates itself in the saved map using a topological estimate which is a place recognition approach to estimate the node nearest to the robot in the saved graph. This is achieved using FAB-MAP [Cummins and Newman, 2011], a large-scale loop-closure framework that uses a visual bag-of-words (BoW) [Sivic and Zisserman, 2003] model for place recognition. In this framework, a visual vocabulary is first constructed on a training set by clustering features extracted from SURF detectors and descriptors [Cummins and Newman, 2011]. Every image can then be represented by a binary vector indicating which *words* from this vocabulary are present, and in what proportion. Finally, loop closures are performed by comparing BoW representations to find the map keyframe that best matches the live image, and initialise localisation in the saved map. This BoW representation can also be used to characterise the appearance of images, which is our choice of feature for modelling scene appearance in Chapter 3.

It is far better to foresee even without certainty than not

— Henri Poincare (Foundations of Science)

3

Learning to Predict Localisation Performance in Novel Real World Environments

In this chapter, we address the limitation of short-sighted instant localisation and propose an extended localisation system able to provide a robot with *a priori* information of how well it will be able to localise along a predetermined route, given the surrounding environment. Specifically, we propose an *appearance-based* approach to estimating localisation performance in the context of Visual Teach and Repeat [Furgale and Barfoot, 2010], [Churchill and Newman, 2013] and showcase our results on real-world offroad urban data. We estimate the likely corridor around a taught trajectory within which a vision-based localisation system is still able to localise itself. In contrast to prior art, our system is able to *predict* this *localisation envelope* for trajectories in similar, yet geographically distant locations where no repeat runs have yet been performed. Thus, by characterising the localisation performance in one region, we are able to predict performance in another. In doing so, we aim to equip robots operating in the field with increased situational awareness in the context of visual navigation.

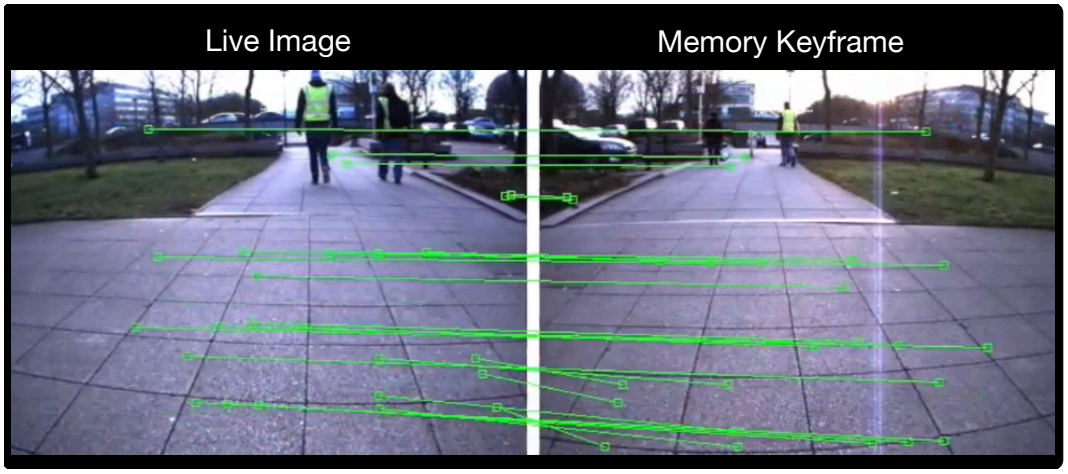


Figure 3.1: In Teach and Repeat, a Teach phase first consists in driving along a desired route and saving keyframes in memory along with image descriptors (right). At Repeat time, the localisation system tries to match live frame image descriptors (left) to those saved in memory (right). The matched keyframe descriptors are here linked by green segments between the live and memory images. Images are collected from a robot platform whilst test driving in Milton Keynes, UK.

3.1 Motivation

Whereas the main body of research of this thesis addresses improving situational awareness from a *perception* point of view ("what is going on around me?"), this research chapter addresses improving a robot's situational awareness from the localisation system's point of view ("where am I?"). In the three pillar view of autonomous robotics we adopt and presented in introduction of this thesis ("where? what? how?"), we find that this additional body of work contributes an interesting complement to our overall aim of aiding far-sighted planning ("how can I achieve my goal?").

The process of localisation is one that continuously provides a robot with the ability (or failure) to localise as it navigates its environment. By construction, this is an instantaneous process consisting in registering live data with either a local map stored in memory such as Visual Teach and Repeat [Furgale and Barfoot, 2010] or a global map accessible online such as GPS localisation. Figure 3.1 illustrates a successful localisation in a Teach and Repeat scenario. Local image features that have been successfully matched between the live image (left) and memory

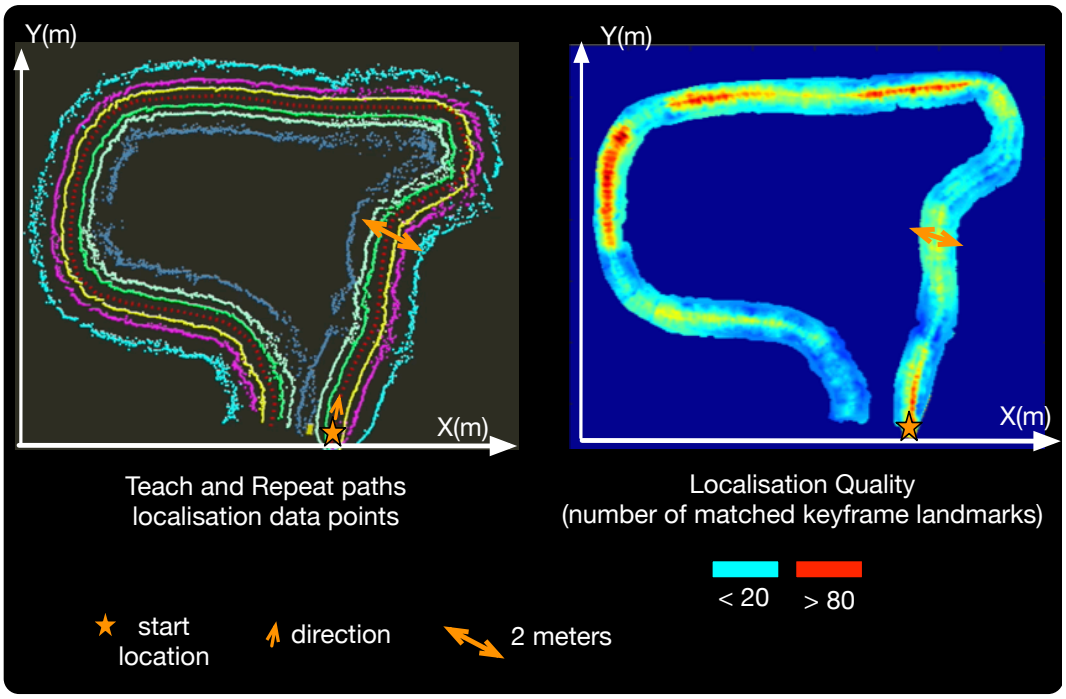


Figure 3.2: A bird's eye view of the variability of the localisation envelope in Teach and Repeat. Left is represented a Teach route (red path) along with a number of offset repeat paths. Right is represented the corresponding localisation envelope where warm colors correspond to a higher number of matched features. The scales are different in both figures but represent the same data collection. There is great variability in the localisation quality even within one meter of the taught trajectory.

keyframe (right) are linked with green segments. For details of the Visual Teach and Repeat paradigm please refer to Chapter 2.3.3.

Whether maps be local or global, traditional localisation systems do not provide the robot with *a priori* knowledge of how well (if at all) they will be able to localise along a given trajectory of interest. Figure 3.2 illustrates how variable localisation performance can be when deviating only a few meters away from the taught trajectory at repeat time. To the left is represented a Teach route (red path) along with a number of repeat paths performed within one meter deviation on either side. To the right is represented the corresponding localisation envelope representing the number of keyframe features matched along these repeat routes. As can be seen, the localisation envelope shows great variety along and across from the taught path.

The variability of the localisation envelope highlights how a robot may have limited ability to explore its environment in novel ways as getting lost can be

costly and restrain robot autonomy. This is a real limitation as many *real world* settings require robots to drive multiple times along a given route. Mars surface exploration may require a rover to retrace a route multiple times to collect rock samples from a given area. In earth-bound autonomous robotics, autonomous vehicles may be used to shuttle pedestrians from one area to another or inspect warehouses regularly. For each of these settings a robot will be reproducing a taught trajectory whilst occasionally needing to avoid a moving obstacle or wanting to explore away from the original path.

Our work is inspired by Churchill et al. [2015] who embed a spatial model of expected localiser performance in Teach and Repeat. By physically sampling the area around a taught trajectory they model the likely localisation envelope using Gaussian Processes. Though this approach allows to predict localisation performance at unseen points around the path, it cannot generalise to freshly taught trajectories. We build on this work and extend applicability by considering an appearance-based model of the envelope for inferring localisation performance *prior* to conducting any repeat runs.

3.1.1 Contributions

In this chapter, we extend the capacity of a localiser beyond instant localisation and propose an appearance-based approach to predict *a priori* how well it will be able to localise against a taught trajectory.

We use Gaussian Processes to predict the number of features likely to be matched by the localisation system against a particular map keyframe based on an appearance model of the keyframe as well as on factors such as intended path deviation and trajectory shape. Our model is trained using data from a prior experience and, due to its appearance-based nature, thus allows the system to *generalise* to predicting the localisation envelope of a novel teach run in a different location. Using data from real traversals, we demonstrate that our approach performs as well as prior art when it comes to interpolating localisation performance based on a number

of repeat runs, while also performing well at generalising performance estimation to freshly taught trajectories.

To the best of our knowledge our approach is the first able to do so, and we make the following contributions:

- Development of an *appearance-based* approach to estimating the likely *localisation envelope* around a taught route in the context of visual teach and repeat
- Demonstration on real-world offroad data that the extended system is able to *predict* this *localisation envelope* for trajectories situated in similar, yet geographically distant locations where no repeat runs have yet been performed
- Demonstration that the proposed solution performs as well as prior art when it comes to interpolating localisation performance based on a number of repeat runs

3.2 Problem formulation

Given the appearance and characteristics of a taught trajectory, our goal is to predict how well a vision-based system will be able to localise during a repeat trajectory offset from the path. We translate this goal into that of learning a function f which maps from an input $\mathbf{x}$ characterising the location along - and appearance of - the taught path, to an output y characterising the localiser's performance at that location:

$$y = f(\mathbf{x}) + \epsilon,$$

where ϵ is some observation noise which we model as a Gaussian $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

Rather than choosing y to be a binary output reflecting if the system is localised or lost, we choose y to correspond to the number of keyframe descriptors matched between the live image and keyframes saved in memory. We find this provides more nuanced information of localisation ability along the path, and does not depend on any choice of threshold for localisation ability.

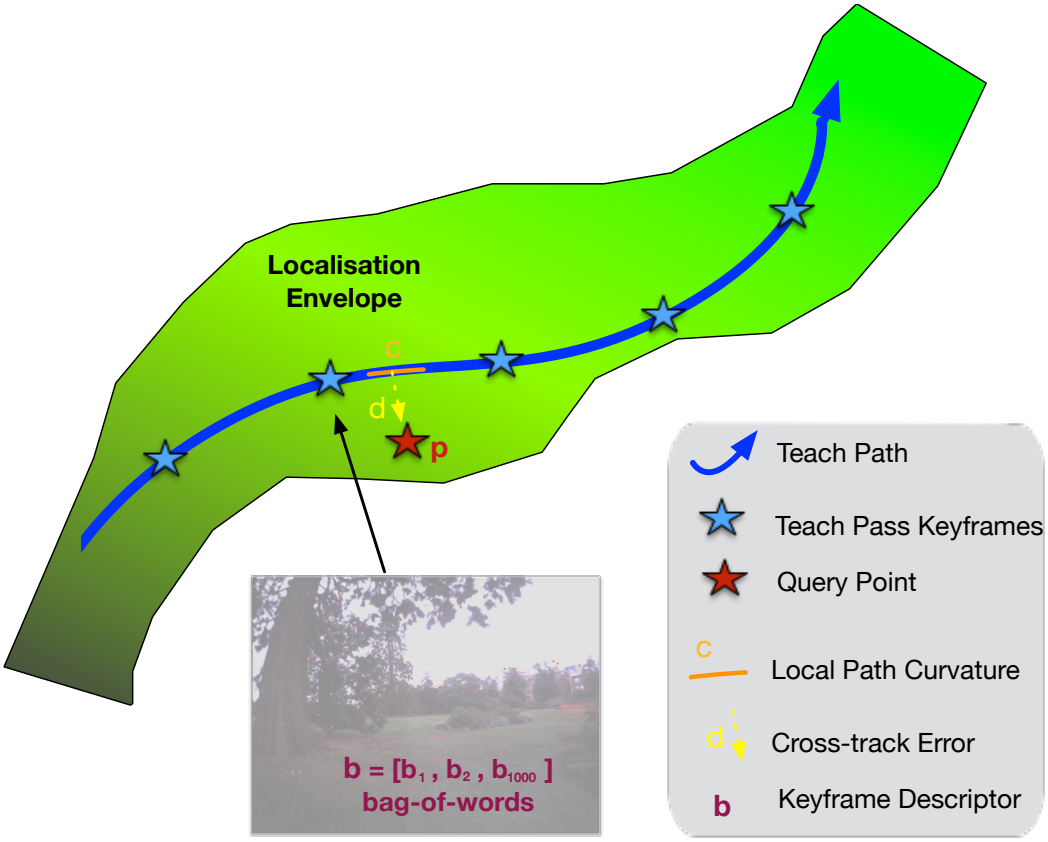


Figure 3.3: For a given query point corresponding to robot pose $\mathbf{p}$ along the path, the input vector $\mathbf{x}$ consists in the cross-track error d, the path curvature c local to the nearest point on the path, and the map features composed of the bag-of-words $\mathbf{b}$ descriptor of the keyframe closest on the path. Only the contours of the localisation envelope - rather than variability of - are shown.

Input features For every robot pose $\mathbf{p}$ along the path (see Figure 3.3), we construct the input $\mathbf{x}$ by combining local map features and pose characteristics. Pose characteristics consist in the local path curvature c after projecting the robot pose orthogonally on the taught path, and signed cross-track error d between pose and projection. Map features consists in the bag-of-words (BoW) [Sivic and Zisserman, 2003] descriptor vector $\mathbf{b}$ from the map keyframe closest to the projected point. These three input features are stacked together to form input $\mathbf{x}$:

$$\mathbf{x} = \begin{bmatrix} \mathbf{b} \\ d \\ c \end{bmatrix}. \tag{3.1}$$

The appearance descriptor $\mathbf{b}$ is constructed from a BoW visual *dictionary* constructed from a dataset collected in a similar area, though different from

the datasets used for training and testing the model. The BoW descriptor is a 1310-bit binary vector representing the relative occurrence of the 1310 words making the visual *dictionary*, where the vocabulary is built using SURF detectors and descriptors.

While our VO implementation is able to provide 6 DoF pose estimates, we choose to model the robot position using only the cross-track error d. This modelling choice reflects the fact that planners usually operate in a 2D plane, and that our repeat runs are conducted parallel to the path, introducing little yaw with respect to the taught track. We do not find this to be a limitation as this work focuses on verifying that an appearance model of the scene can be used to predict localisation performance. Future work could collect data with varying viewpoint angles as the robot repeats the taught trajectory, and include yaw in the input feature vector. As small rotations of the robot camera greatly change scene appearance, this would provide a richer appearance-based model reflecting more complex real-world scenarios.

Finally the path curvature results from fitting a cubic spline to the taught trajectory and provides information with regards to path geometry.

Model To characterise the localisation envelope, we choose a Gaussian Process (GP) with a Matérn kernel of parameter $\nu = 3/2$:

$$k_{\nu=3/2}(\mathbf{x}, \mathbf{x}') = \sigma_f^2(1 + \frac{\sqrt{3}|\mathbf{x} - \mathbf{x}'|}{l})exp(-\frac{\sqrt{3}|\mathbf{x} - \mathbf{x}'|}{l}),$$

where σ_f is a scaling factor shared by all input dimensions and l is a diagonal matrix encoding the three length factors l_b, l_d, l_c corresponding respectively to the bag of words, distance from path and local path curvature input features:

$$l = \begin{bmatrix} l_b^2 & 0 & 0 \\ 0 & l_d^2 & 0 \\ 0 & 0 & l_c^2 \end{bmatrix}. \tag{3.2}$$

We choose a Matérn kernel of parameter $\nu = 3/2$ as it is stationary, is a function of the relative distance between two inputs, and is once differentiable which provides more realistic smoothness characteristics than an infinitely differentiable

kernel such as the commonly used squared exponential when modelling real world data [Rasmussen and Williams, 2012].

3.3 Dataset

To test our approach on real world data, we deployed a Clearpath Husky A200 platform (Figure 3.4) equipped with a Point Gray Bumblebee XB2 stereo camera in Oxford University Parks, UK (Figure 3.5).

Figure 3.4: A Clearpath Husky A200 with a Bumblebee2 stereo camera.

This choice was motivated by the fact that exploration around a taught trajectory would occur more readily in off-road environments where the robot is able to move away from the path and where the environment features are more variable under change of pose observation than urban areas. Deviations in urban areas would generally remain within a lane reducing the need for an extended localisation envelope.

Two routes in separate locations of the park were identified, and for each route, we conducted a single teach pass followed by seven autonomous repeat passes with imposed lateral offsets set to $\{-1, -0.5, -0.25, 0, +0.25, +0.5, +1\}$ m. Repeat paths were conducted by localising against the taught trajectory. Lateral offsets were achieved by additionally imposing the fixed lateral offset values to the path tracking controller. In this way the robot remains parallel to the taught path at the desired distance and without need for manual control.We refer to the two routes in this section as *Route A* and *Route B*, which were 100m and 70m long respectively.

A third dataset was collected in the same area to create the visual dictionary used for building the Bag-of-Words feature vector.

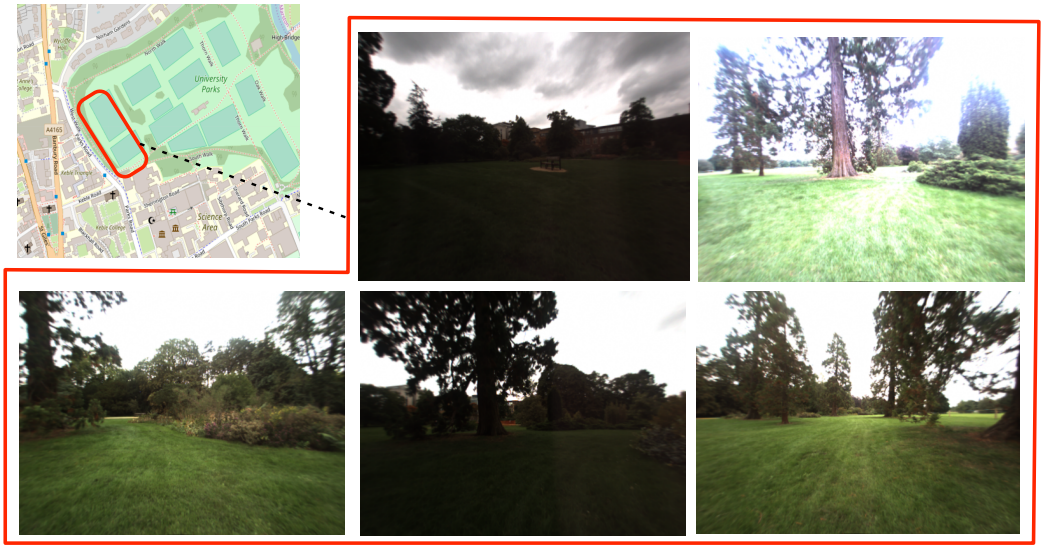


Figure 3.5: Location and selection of images collected in University Parks, Oxford, UK.

Ground truth localisation quality

While collected in similar environments, we observe a range of localisation performance around both paths, as shown in Figure 3.6. In both routes, some areas have as little as 5 matching landmarks, whereas other places yield more than 80. To provide intuition for why the performance can vary in such a small workspace, we visualise the number and distribution of keyframe landmarks for locations with high and low localisation scores for each route.

In Figure 3.8, we see that poor localisation performance can be attributed to the fact that many landmarks are detected in the cloudy sky, whereas a good location has landmarks spread throughout the image. Similarly, Figure 3.7 shows that low contrast leads to low feature count, whereas clear texture provides stable features for matching.

These images show us that localisation performance is correlated with appearance, and support our intuition that an appearance-based model may be appropriate for this task.

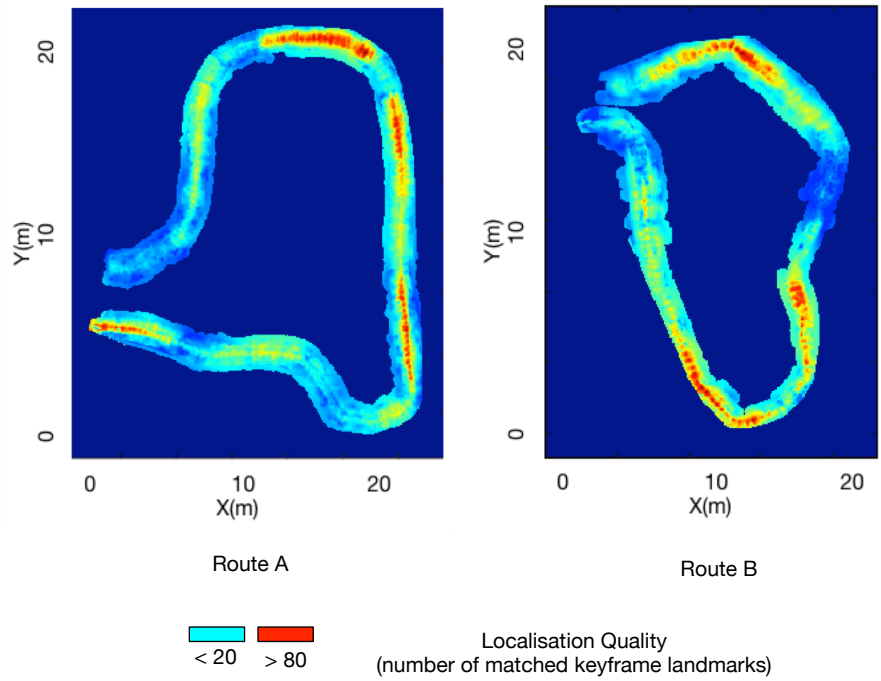
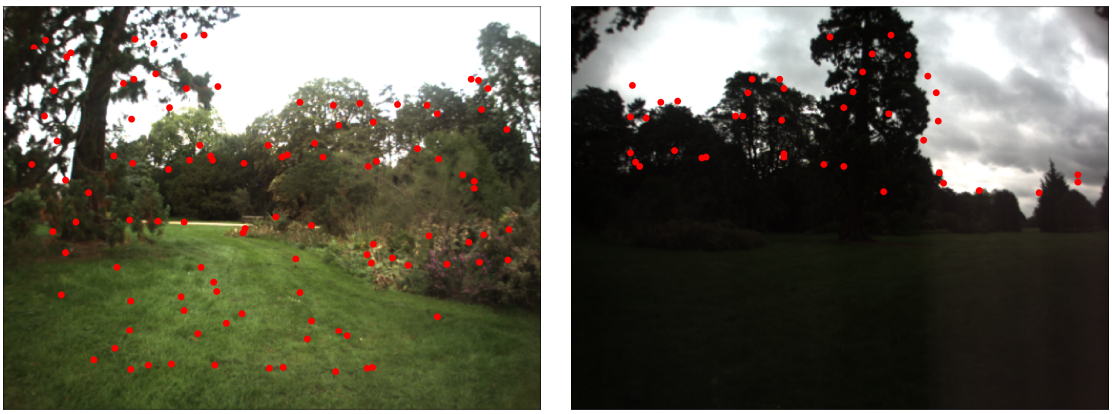


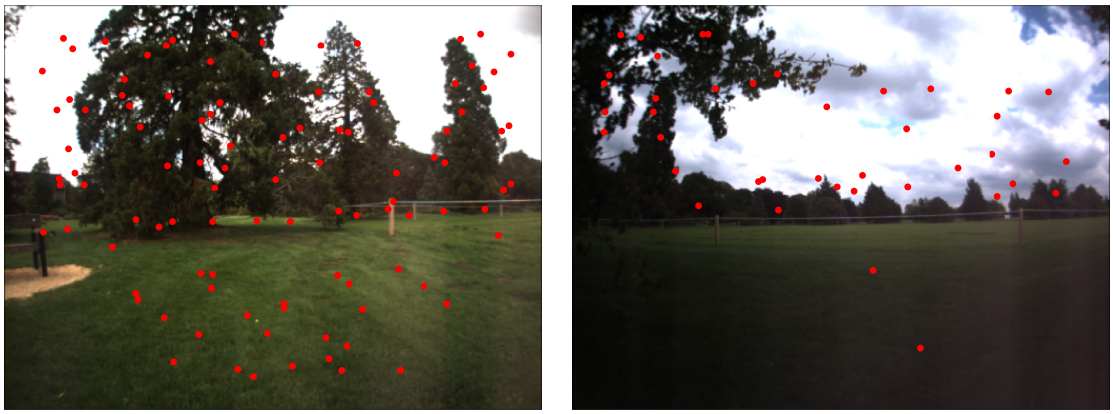
Figure 3.6: Localisation quality (number of matched keyframe landmarks) constructed from all collected data for both routes. Red represents a high number of landmark matches (>80), blue represents a low number of matches (<20). It can be seen that the localisation quality varies drastically over the course of the path.



(a) High localisation quality.

(b) Low localisation quality.

Figure 3.7: Landmark distribution for keyframes with high and low localisation quality in Route A. As can be seen, poor localisation results when there is an insufficient spread of landmarks throughout the image and poor contrast.



(a) High localisation quality.

(b) Low localisation quality.

Figure 3.8: Landmark distribution for keyframes with high and low localisation quality in Route B. As can be seen, low contrast and unstable features detected in the cloudy sky leads to poor localisation scores.

3.4 Experimental evaluation

In the following section, we present and discuss a number of experiments conducted to address a few key questions with respect to our proposed solution. We also explicit the training methodology followed across experiments.

3.4.1 Methodology

For each experiment we first learned the hyperparameters θ of our model on the training data by maximising the marginal log likelihood with respect to the parameters:

$$\log p(\mathbf{y}|\mathbf{X},\theta) = -\frac{1}{2}\mathbf{y}^T(K+\sigma^2 I)^{-1})\mathbf{y} - \frac{1}{2}\log|K+\sigma^2 I| - \frac{N}{2}\log 2\pi. \tag{3.3}$$

As this requires inverting matrix K in time $\mathcal{O}(N^3)$, we consider the practical implementation of Equation 3.3 which uses a Cholesky decomposition for faster and more numerically stable computation of the matrix inverse:

Algorithm 1: Practical Marginal Log Likelihood Implementation

input: $\mathbf{X}$(inputs), $\mathbf{y}$ (targets), K (covariance matrix), σ^2 (noise level), θ (hyperparameters) ;
$L \leftarrow \text{cholesky}(K+\sigma^2 I)$;
$\alpha \leftarrow L^T \setminus (L \setminus \mathbf{y})$;
$\log p(\mathbf{y}|\mathbf{X},\theta)) \leftarrow -\frac{1}{2}\mathbf{y}^T\alpha - \sum_i \log L_{ii} - -\frac{N}{2}\log 2\pi$;
return *log* $p(\mathbf{y}|\mathbf{X},\theta)$ *(log marginal likelihood)*;

Our code is implemented in Matlab, and we minimise the negative log likelihood with a quasi Newton method with respect to the length scales, kernel scaling factor σ_f and data noise σ^2:

$$\theta = [l_b, l_d, l_c, \sigma_f, \sigma].$$

Metric of Success To quantify the similarity between predictions and ground truth, we define a threshold of 25 matched landmarks as corresponding to successful localisation and compute the F1 measure. This is the same threshold as that used by the localisation system. We regroup our experimental results in Table 3.1.

Table 3.1: Experimental results F1 performance

Scenario	BoW (ours)	DAP [Churchill et al., 2015]	Description
Scenario I	0.949	0.939	Comparison to baseline model
Scenario II	0.915	–	Generalising from Route A to Route B
Scenario III	0.799	–	Generalising from Route B to Route A

3.4.2 Benchmarking Against an Existing Approach

To evaluate our model, we first looked at how the introduction of BoW appearance features compared to the baseline performance of Churchill et al. [2015], which consider the distance along the path (DAP) as input feature rather than our appearance-based BoW representation. Since this baseline model is unable to generalise to new routes, we considered a scenario where we gathered some training data during a few repeat passes, and wished to predict the localisation performance for future repeat passes. This was accomplished by splitting the data gathered from Route A into training and testing sets. We refer to this test case as *Scenario I*.

Both models' hyperparameters were optimised by maximising the log-likelihood of the training data. The results from both models along with the ground truth localisation quality values are shown in Figure 3.9.

The performance of both models were commensurate, supporting the use of BoW features as a surrogate for the notion of a "place". With a threshold of 25 matched landmarks as a successful localisation, we obtained an F_1 measure of $F_{1,BoW} = 0.949$ for our BoW model, and an $F_{1,DAP} = 0.939$ for DAP.

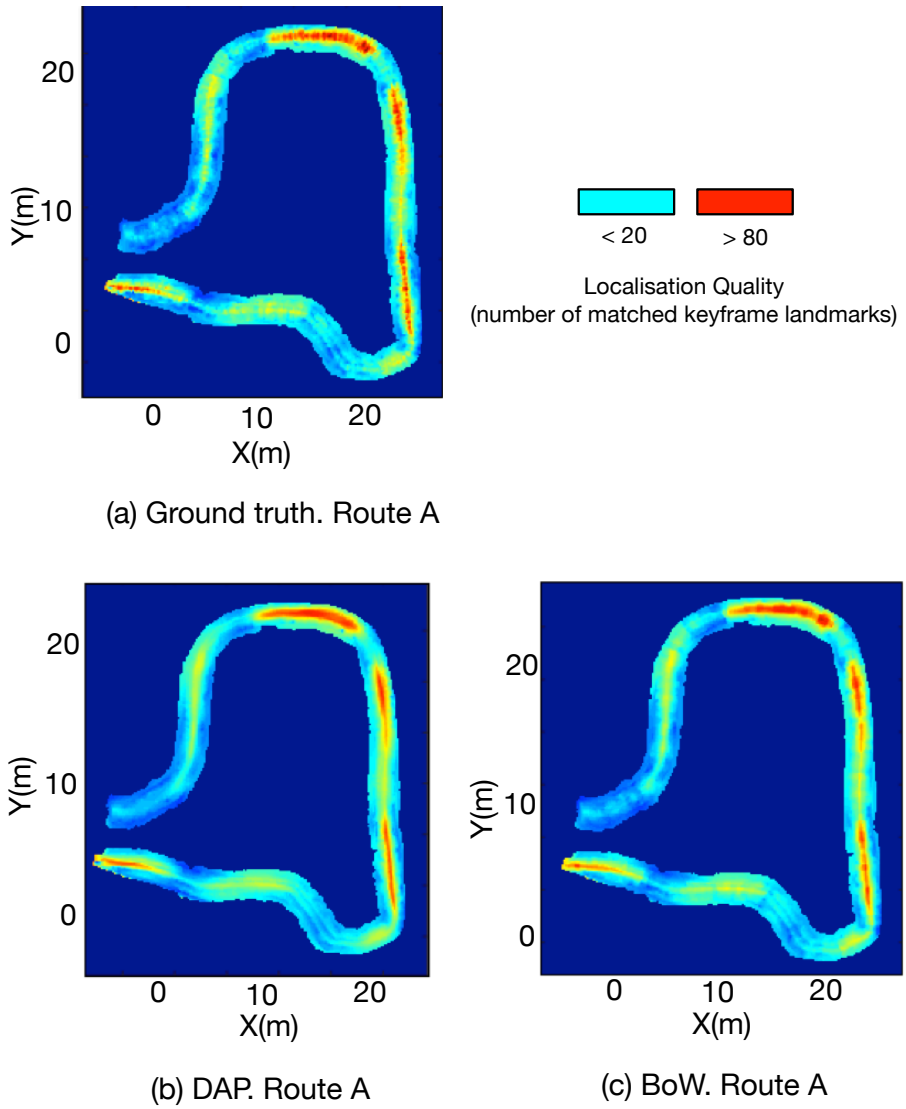


Figure 3.9: Scenario I: Localisation quality predictions for a scenario where the Route A data was split into a number of repeat passes for training, and the remainder for testing. Red represents a high number of landmark matches (>80), blue represents a low number of matches (<20). Both models are commensurate in capturing the localisation envelope along the path, suggesting that BoW appearance features are a suitable replacement for the distance-along-the-path parameterisation.

3.4.3 Generalising to New Routes

To investigate how well our model does at generalising predictions to new paths, we considered two additional scenarios, *Scenario II* and *Scenario III*, where we trained our model on a selection of repeat passes for one route and tested on a subset of repeat passes for the other. *Scenario II* consisted of training on Route A and testing on Route B, and vice versa for *Scenario III*. The results for both

scenarios are shown in Figures 3.10 and 3.11.

Both experiments show that we were able to capture the general evolution of the localisation envelope along the path. Quantitatively, using the same localisation success threshold of 25 landmark matches, we obtain F_1 measures of $F_{1,II} = 0.915$ and $F_{1,III} = 0.799$ for *Scenarios II* and *III*, respectively.

The lower performance in *Scenario III* may be attributed to two factors. On the one hand, the dataset of Route B is smaller than that of Route A, making it more challenging for the model to extrapolate than in *Scenario II*. On the other hand, the weather was more changeable during collection of the data in Route B than during that of Route A, which affected localisation negatively and resulted in the robot getting temporarily lost at several locations along the path. We hypothesize that we may have learned a pessimistic model when training on the data of Route B.

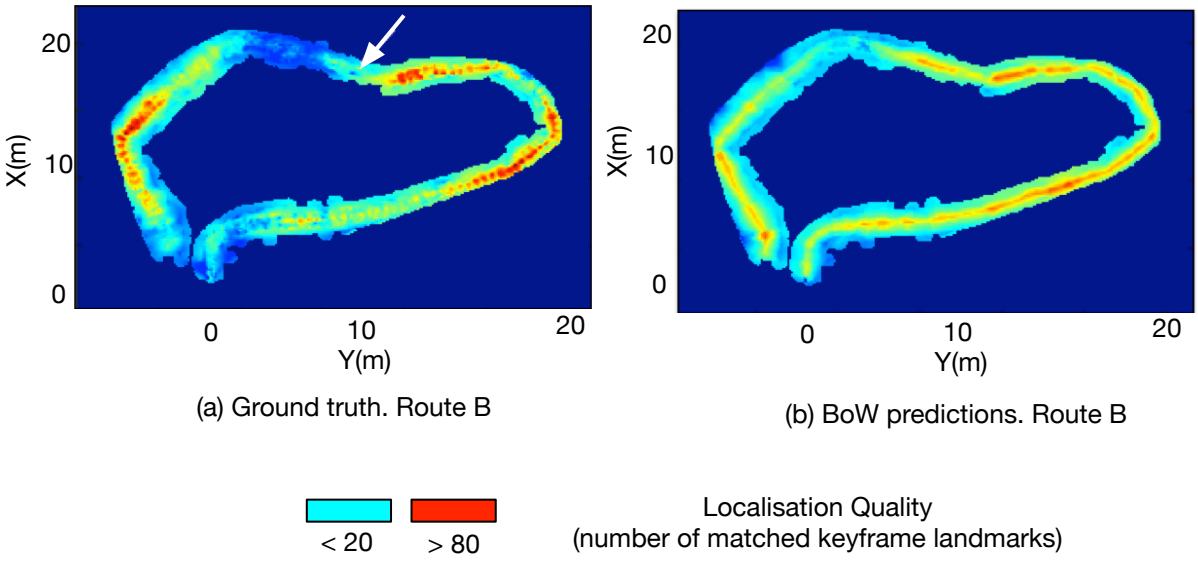


Figure 3.10: Scenario II: Localisation quality predictions for training on Route A and testing on Route B. Red represents a high number of landmark matches (>80), blue represents a low number of matches (<20). The BoW model captures the variability of the envelope well along the path, but is overconfident in a few select areas. The white arrow on the ground truth figure, indicates a section of the path where we over-estimate our ability to localise. We refer the reader to the discussion section where we further elaborate on this example.

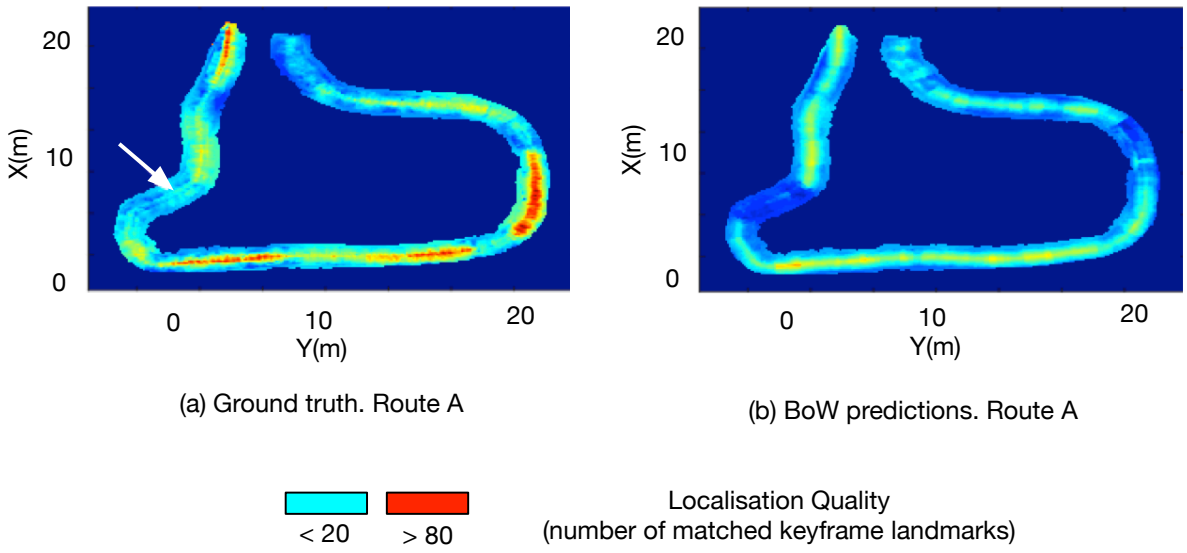
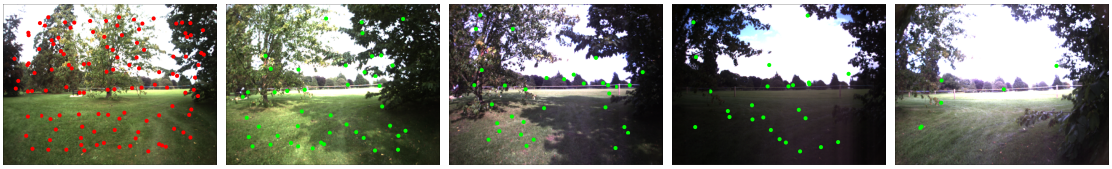


Figure 3.11: Scenario III: Localisation quality predictions for training on Route B and testing on Route A. Red represents a high number of landmark matches (>80), blue represents a low number of matches (<20). The model accurately captures the localisation performance for the turns along the trajectory but is overall under confident. The white arrow on the ground truth figure, indicates a section of the path where we particularly under-estimate our ability to localise. We refer the reader to the discussion section for a more detailed study of this example.

3.4.4 Discussion

While the overall results show promise, there were sections of Route B where our model was overconfident. In contrast, the system was generally underconfident on Route A. To form an intuition for this, we first consider a section on Route B where localisation was very poor, yet our model yielded an optimistic prediction. We indicate this section by a white arrow in Figure 3.10a. The corresponding keyframe obtained from the teach pass of Route B is depicted in Figure 3.12, along with a selection of repeat frames which matched against it. Interestingly, there were 102 widely distributed landmarks detected in the keyframe. However, the overall appearance of the scene changed dramatically during the repeat runs, introducing shadows and changing the colour of the grass. As a result, the model was optimistic given the feature-rich keyframe, but was unable to account for the reduced performance due to the environmental effects.

Conversely, we consider a section of Route A where our model under-predicted performance. We indicate this section by a white arrow in Figure 3.11a. As



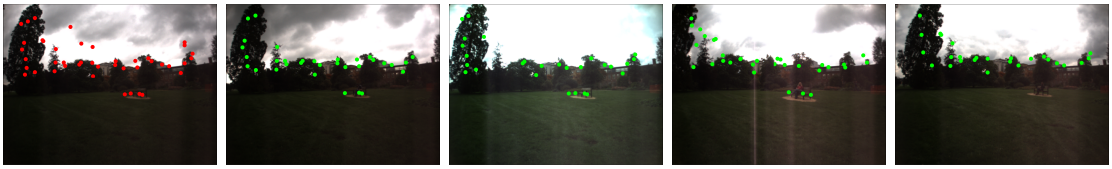
(a) Teach. Route B (b) Repeat with a lateral offset of +0.25m. (c) Repeat with a lateral offset of -0.25m. (d) Repeat with a lateral offset of -0.5m. (e) Repeat with a lateral offset of +1.0m.

Figure 3.12: Teach and repeat images for a location of Route B where the localisation prediction was optimistic in *Scenario II*, but the ground truth performance was poor. We overplot in red the landmarks in the teach pass keyframe, and in green the landmarks that effectively found a match in the repeat images. The keyframe shows good contrast and as many as 102 landmarks to localise against which poses it as a good candidate for localisation. The appearance of the repeat passes is however dramatically altered by environmental conditions resulting in a low number of matches. The two repeats shown to the right of the features effectively did not success in matching to the keyframe, and we here represent the match to the closest keyframe in the map. The model was not able to capture these environmental effects.

before, we show the corresponding keyframe from the teach pass of Route A along with a selection of successful matches from the repeat passes in Figure 3.13. The keyframe presents only 42 landmarks with very low contrast. Our model therefore inferred from this landmark-deprived keyframe that performance will be poor at that location. However, we found that the scene appearance remained consistent throughout the repeat runs of the training set, which provided a higher than expected localisation score. Despite their low number, the landmarks lie mostly on the skyline, providing robust matching.

There exist a number of limitations to our model. The primary limitation is that we consider the keyframes collected during the teach pass but do not account for a potential change in appearance during repeat. This can result in confusion for the regressor where both high and low localisation scores result for the same input feature vector, which would be accounted for in the model as measurement noise instead of modelling error. Other limitations include our choice of keyframe representation, and localisation performance metric.

Although FAST/BRIEF descriptors are considered for localisation, we model the appearance with BoW features based on SURF detectors and descriptors. This mismatch may cause poorer performance. In addition, localisation performance



(a) Teach. Route A

(b) Repeat with a lateral offset of +0.25m.

(c) Repeat with a lateral offset of -0.25m.

(d) Repeat with a lateral offset of -0.5m.

(e) Repeat with a lateral offset of +1.0m.

Figure 3.13: Teach and repeat images for a location of Route A where the localisation prediction was pessimistic with regards to the ground truth in *Scenario III*. We overplot in red the landmarks in the teach pass keyframe, and in green the landmarks that effectively found a match in the repeat images. The keyframe shows low contrast due to overcast conditions and offers only 43 landmarks for localisation. The appearance of the repeat passes remains however admirably similar to that of the teach phase resulting in nearly perfect matching during the repeats.

seems to correlate with the quality rather than the quantity of detected features. An interesting follow-up to this work would be to unify the representation to which features are particularly amenable to robust localisation, as indicated in Figure 3.13.

3.5 Conclusion and Future Work

This introductory research chapter proposes an appearance-based approach to predicting the likely localisation envelope given only a single teach pass at an otherwise novel location. In doing so it aims to aid the planning process in the context of visual teach and repeat systems. Our results indicate that our method successfully generalises to provide if not an exact, then nevertheless a faithful representation of the localisation envelope of novel trajectories – the key objective of this work. We suggest future work in Chapter 7. This chapter proposed an approach for improving a localiser system's situational awareness. In the next chapters we present extended developments of the Deep Tracking framework [Ondruska and Posner, 2016] to improve a perception system's situational awareness.

"Prediction is very difficult, especially if it's about the future."

— Nils Bohr

4

Estimating the State of the World in Complex Dynamic Environments

In this thesis, we argue that situational awareness is a pivotal requirement for autonomous robotics. In the navigation, perception and planning triad of autonomy, a robot constantly needs to be able to localise in its environment in order to plan. In the previous chapter, we investigated an appearance-based approach for providing a localiser system *a priori* knowledge on its ability to localise in a given real-world environment. Rather than providing a robot with only instant localisation, the localiser system is able to additionally provide localisation information ahead of time which can aid more far-sighted planning. In this chapter, we investigate how a perception system module can be enhanced and provide information beyond the instantaneous data received from its sensors, in order to contribute to more far-sighted planning, crucially in real-world environments.

4.1 Motivation

In the context of perception, for a robot to safely navigate its environment, it must be able to interpret its surroundings and predict the state of that environment over time. In particular, it must be able to successfully track objects through

occlusions, beyond the limited instantaneous information provided by its sensors. Such capabilities are relied upon by decision-making processes such as motion planning and control.

In urban environments this is typically a challenge, as dynamic objects such as buses and cars commonly occlude pedestrians and cyclists making it difficult for a robot to plan a route through the scene with far-sightedness (Figure 4.1). Traditional approaches for tracking often feature numerous hand-engineered stages, often developed independently, and which make some degree of assumption or simplification regarding object dynamics, appearance, and interaction models. In this chapter, we propose an alternative approach and present an end-to-end trainable framework to learn a model of the world dynamics directly from data, in an entirely *self-supervised* manner.

To this end, we leverage and extend the framework of *Deep Tracking* proposed by [Ondruska and Posner, 2016] for tracking dynamic objects through occlusion in an end-to-end manner, from data, and originally demonstrated on synthetic scenarios. We scale up the original architecture and conduct extensive model comparison on 75 minutes of data acquired while stationary in the middle of a busy urban intersection. The achieved tracking accuracy is superior to the original architecture presented in [Ondruska and Posner, 2016], as well as to that of an alternative state-of-the-art model-free tracking solution targeted at the same problem [Wang et al., 2015]. We also extend our solution to estimate the scene semantics via the application of the principle of inductive transfer. In essence, we are interested in investigating how the framework of *deep tracking* can be further developed as a powerful state representation and predictor that could equip robots with a more far-sighted understanding of their environment, beyond natural occlusions.

4.1.1 Contributions

In this chapter, we leverage the capacity of recurrent neural networks to track dynamic objects through occlusion in a self-supervised end-to-end manner, directly

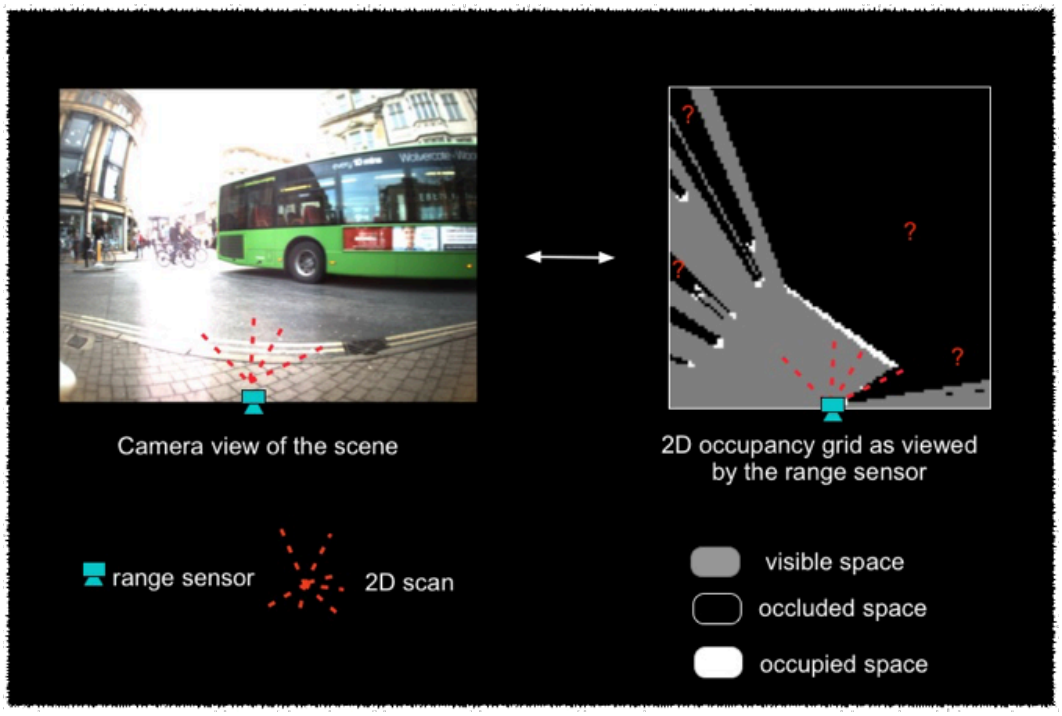


Figure 4.1: A typical occlusion caused by a passing bus at a busy intersection (left), and as seen from a range sensor (right). Most of the scene is occluded and the sensor cannot "see" if the scene is occupied or not behind the bus.

from data, and extend the novel framework of *Deep Tracking* proposed by Ondruska and Posner [2016] which was originally demonstrated on synthetic scenarios. The simple recurrent neural network proposed in Ondruska and Posner [2016] was demonstrated to be sufficient for the simulated dynamic scenario evaluated in that work. However, an effective deployment in real-world robotics applications poses a set of challenges for which a more appropriate architecture must be chosen.

We set our work in a real-world environment and consider a static robot equipped with a horizontal 2D laser at a busy urban intersection. We present our choice of extensions to scale up the baseline framework's capacity to deal with a complex real-world robotics task in Section 4.2, and extend the framework to additionaly learn to semantically classify the scene in a data efficient manner via inductive transfer of knowledge in Section 4.3. We test our framework on a 75-min long dataset collected from a static sensor at a busy urban intersection and release the code and dataset for use in the community. In doing so we make the following contributions:

- extension of *deep tracking* to learning real-world dynamics at a busy urban intersection
- motivation for and ablation studies of design choices compared to the original framework for addressing real-world requirements
- demonstration of improved tracking ability compared to a state-of-the-art model-free tracking system
- demonstration of effective (in terms of supervised effort) transfer of knowledge from tracking to semantically labelling the scene

We consider the problem of predicting the unoccluded state of a dynamic urban scene in Section 4.2 and that of predicting the scene semantics in Section 4.3.

4.2 Tracking in the Real World

In this section we first outline our problem formulation and present our proposed extended *deep tracking* architecture tailored for effectively tracking objects through occlusion in the real-world in Section 4.2.2. We then detail the real-world dataset collected and network training procedure in Sections 4.2.3 and 4.2.4. We finally present our tracking results and benchmark against an existing approach in Section 4.2.5.

4.2.1 Problem Formulation

The tracking objective consists of predicting the future unoccluded state of the world $\mathbf{y}_{t+n}$ at time $t+n$, given a sequence of occluded observations of the world $\mathbf{x}_{1:t}$. At training time, empty inputs $\mathbf{x}_{t+1:t+n}$ are provided to force the network to iteratively update and use its belief $\mathbf{h}_t$ to infer future state occupancy:

$$p(\mathbf{y}_{t+n}|\mathbf{x}_{1:t}) = p(\mathbf{y}_{t+n}|\mathbf{h}_{t+n}).$$

In practice, this means providing input grids $\mathbf{x}_{t+1:t+n}$ where all grid cells have value 0 from timestep $t+1$ to $t+n$. At deployment time, the network can either be used to predict future unoccluded scene occupancy $\mathbf{y}_{t+1:t+n}$ by providing empty

inputs from $t+1$ to $t+n$, or instantaneous unoccluded scene occupancy $\mathbf{y}_t$ by providing the sequence of occluded laser inputs $\mathbf{x}$ up to timestep t. The predicted occupancy grid is modelled as pixel-wise independent Bernoulli variables for which the binary cross-entropy loss is calculated:

$$\mathcal{L} = -\sum_i y^i \log(p^i) + (1-y^i)\log(1-p^i), \tag{4.1}$$

where p^i correspond to pixel-wise output predictions and y^i is the corresponding occupancy ground truth. As we do not have access to the true unoccluded occupancy $\mathbf{y}_t$, we train the model in a self-supervised fashion. The predicted output occupancy $p(\mathbf{y}_{t+n}|\mathbf{x}_{1:t})$ is compared to $\mathbf{x}_{t+n,vis} = \mathbf{y}'_{t+n}$, which corresponds to the visible (and therefore known) part of the world at time $t+n$. The binary cross-entropy loss is therefore calculated and backpropagated only on the ground truth available, i.e the *visible* part of the space. This is achieved by simply masking the output prediction $\mathbf{y}_t$ with the visible input $\mathbf{x}_{t,vis}$ and multiplying the resulting grid element-wise with the occupancy part grid $\mathbf{x}_{t,occ}$.

Finally, we compute the mean F1 occupancy score on the test set sequences for every timestep. A threshold of 0.5 is used to determine whether a cell is predicted as occupied or free. We refer the reader to Section 2.2.5 for a more in depth tracking problem formulation.

4.2.2 Network Architecture

An overview of the original framework of Ondruska and Posner [2016] and our proposed network extensions are shown in Figures 4.2 and 4.3.

In the original architecture, the input $\mathbf{x}_t$ at time t is processed by an 8-layer encoder which is used to update a basic Recurrent Neural Network memory cell of 16 feature maps. For a real-world deployment, we address the following questions:

- can the architecture be adapted to track objects of different sizes such as vehicles and pedestrians?

- can the architecture be adapted to effectively remember information for long periods of time to deal with real-world occlusions?
- can the framework learn to exploit place-dependent information such as the presence of static obstacles?

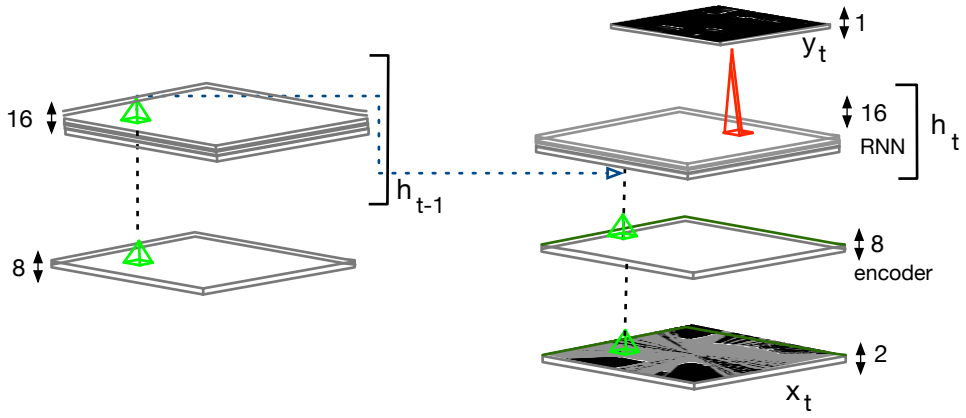


Figure 4.2: The original architecture proposed by Ondruska and Posner [2016]. It features a simple recurrent unit of 16 layers, an input encoder of 8 layers and a decoder module.

Our proposed extensions feature multi-layered memory cells where activations for each layer l at time t are a function of the activations of layer $l-1$ below, as well as its own activations at time $t-1$, thus implementing the recurrence. Combined with the incorporation of dilated convolutions [Yu and Koltun, 2015] this allows the model to simultaneously track objects of very different sizes, such as buses, cars, and pedestrians. A variant of gated recurrent units [Cho et al., 2014, Xingjian et al., 2015] allows the network to extract and remember information from the past and use it for prediction of occluded objects or future states of the world. Finally, an additional static memory can also be utilised, in which the network is given the capacity to learn individual pixel-wise biases that are added to the output of each convolution. A simple convolutional sigmoid function decoder is then applied to the feature maps from all the layers of the network, to obtain the final cell occupancy output grid $\mathbf{y}_t$. For model performance comparison, we also experiment with architectures that decode only the 16 feature maps of the

last hidden unit. We specify this in the text when applicable. These features are described in greater detail below.

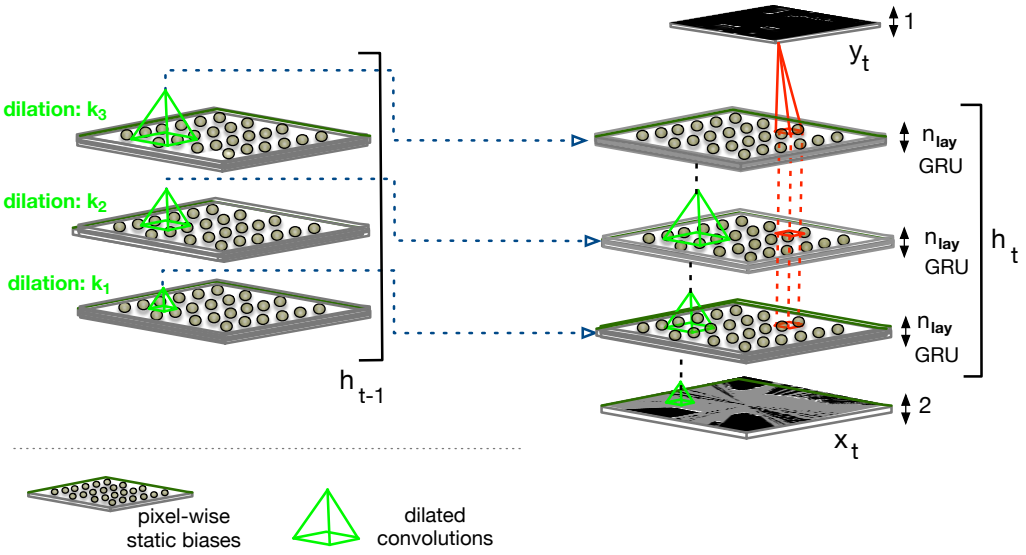


Figure 4.3: The architecture proposed for tracking and semantically classifying real-world scenes. It features dilated convolution of varying sizes (k_1, k_2, k_3), enhanced static and dynamic memory capabilities with pixel-wise static biases and three layers of n_{lay} Gated Recurrent Units, and decodes part or all of the hidden state into an unoccluded grid occupancy (*statet*). These architectural choices are motivated in the main text.

Multi-Scale Convolution

In order for the network to correctly predict the occupancy and label of a moving object at location i, the object must fall within the *receptive field* of the hidden unit in the final layer. The *receptive field* specifies the part of the input data that can affect the activation value of the hidden unit. For a traditional 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 limits the size (in terms of the input data) of objects that can be effectively tracked. This is unrealistic in real-world settings, as dynamic objects can be of vastly different sizes, from pedestrians to buses. This can be ameliorated by increasing the receptive field size, either by increasing the kernel size or stacking multiple convolutions on top of one another. This however creates a computational challenge as the number of parameters and computational complexity grows quadratically with K in the first case and linearly in the second case.

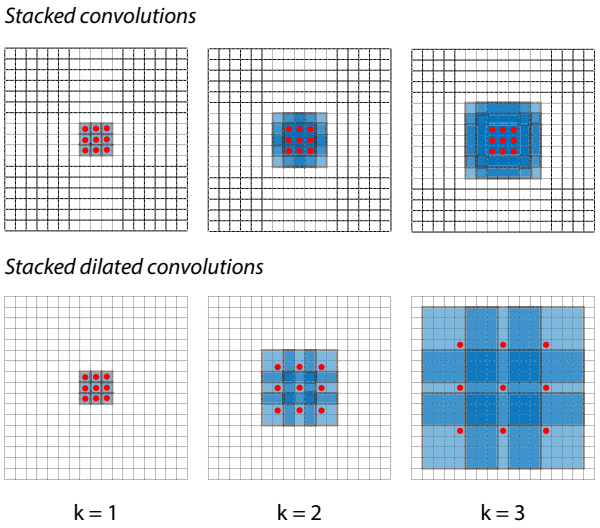


Figure 4.4: Multi-scale context aggregation preserve the image resolution by stacking *dilated convolutions* [Yu and Koltun, 2015]. At layer k the red pixels are convolved with a skip of $2^{k-1} - 1$ pixels. This results in exponential growth of the blue *receptive field* **[bottom]** as opposed to stacking classical convolutions, which result only in linear growth **[top]**.

We circumvent this issue by instead exploiting dilated convolutions [Yu and Koltun, 2015] through the network layers, where the receptive field grows exponentially with the number of layers. That is, we perform the classical 3×3 convolution but skip $2^{k-1} - 1$ pixels in between convolved pixels at each layer k, as illustrated in Figure 4.4. This results in a $(2^{K+1} - 1) \times (2^{K+1} - 1)$ receptive field at final layer K. This dilated convolution is then used as a building block to implement the *dynamic memory* described in the following section. This technique affords the ability to capture structure at multiple scales while reducing the amount of weights necessary to obtain the desired receptive field.

Dynamic Memory

To be able to track a moving object through extended periods of occlusion, the network must keep in memory the location of the object and other properties such as its shape and velocity. Prior art on recurrent neural networks stresses the importance of specially dedicated units such as *long short term memory* [Hochreiter and Schmidhuber, 1997] to support information-caching, to ensure training is not hampered by the vanishing gradient problem [Pascanu et al., 2012]. Inspired

by [Xingjian et al., 2015], we implement a convolutional variant of *gated recurrent units* [Cho et al., 2014], or GRUs, as the processing step at each layer (Figure 4.5). GRUs are computationally simpler than LSTM units, and have been shown to exhibit very similar performance. The output of each unit is given by the weighted combination of its previous output at time $t-1$ and a candidate memory $\bar{\mathbf{h}}_t$ computed from the output of the layer below. The amount of information retained or forgotten can be modulated by the unit using the reset and forget gates $\mathbf{r}_t, \mathbf{f}_t$

$$\mathbf{f}_t = \sigma(\mathbf{W}_{xhf} * (\mathbf{x}_t + \mathbf{h}_{t-1}) + \mathbf{b}_z) \ , \tag{4.2}$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_{xhr} * (\mathbf{x}_t + \mathbf{h}_{t-1}) + \mathbf{b}_r) \ , \tag{4.3}$$

$$\bar{\mathbf{h}}_t = \tanh(\mathbf{W}_{xh} * (\mathbf{x}_t + \mathbf{r}_t \circ \mathbf{h}_{t-1}) + \mathbf{b}_h) \ , \tag{4.4}$$

$$\mathbf{h}_t = \mathbf{f}_t \circ \mathbf{h}_{t-1} + (1 - \mathbf{f}_t) \circ \bar{\mathbf{h}}_t \ . \tag{4.5}$$

Here $*$ denotes dilated convolution described earlier and $\circ$ denotes element-wise multiplication. An illustration is shown in Figure 4.5.

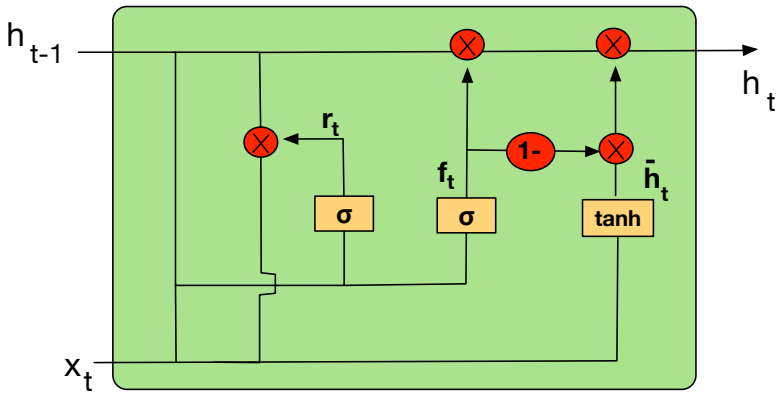


Figure 4.5: A Gated Recurrent Unit (GRU) allows for the modulation of the amount of information retained via the use of reset ($\mathbf{r}_t$) and forget ($\mathbf{f}_t$) gates.

Static Memory

Another extension of the model is to allow each cell to learn a unique and universally accessible piece of information different from all other cells. This is achieved by biases $\mathbf{b}_z$, $\mathbf{b}_r$, and $\mathbf{b}_h$ in Equations 4.2- 4.4 which are not a per-layer constant as

in the case of classical convolution, but are learned individually for each neuron during training. This can enable the network to learn place-specifc information such as the static occupancy of the cell or the usual object classes in a particular area, which can then be used to aid the network prediction.

The updated architecture is illustrated in the functional diagram of Figure 4.6.

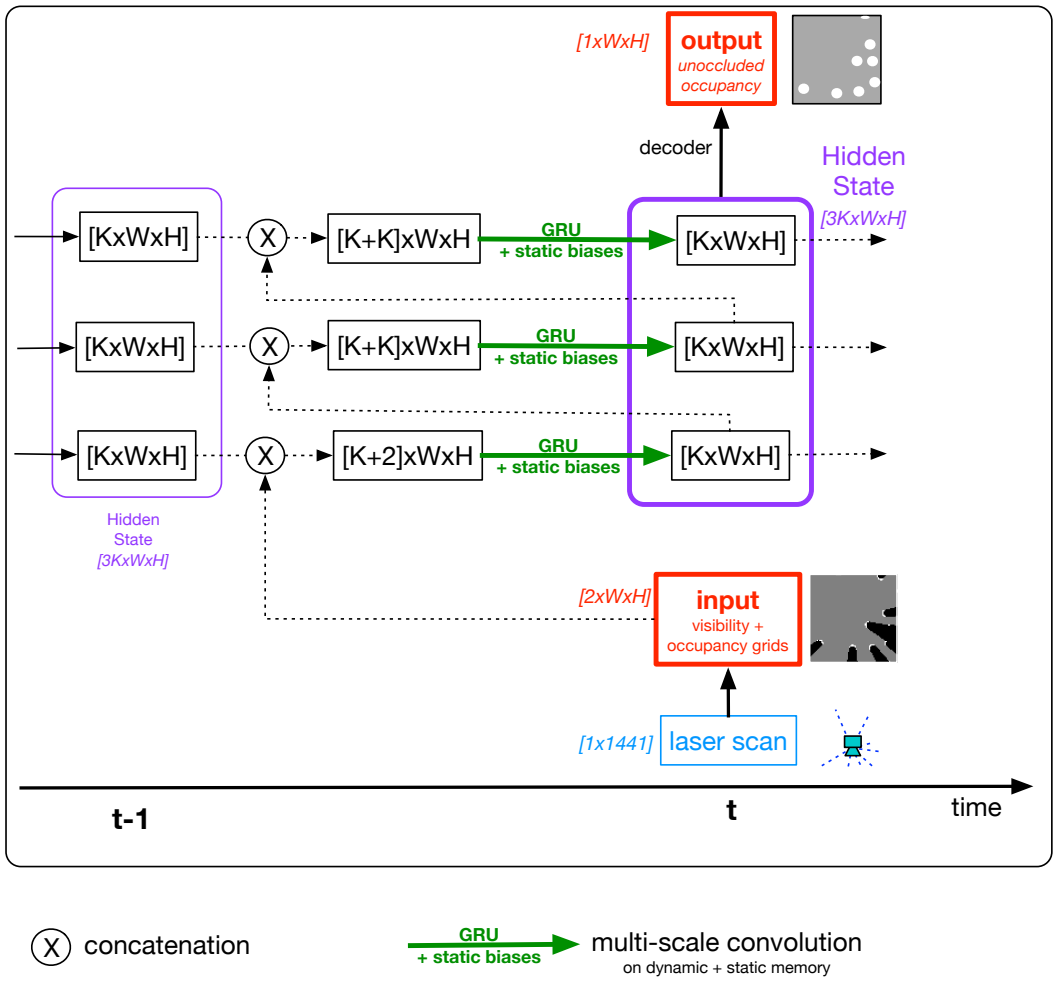


Figure 4.6: Functional diagram illustrating how a new input at timestep t is incorporated into the network to update the hidden state and output an unoccluded occupancy grid. The hidden state is composed of three layers $l = 1 : 3$ of K feature maps. Each layer l at timestep t is updated sequentially by concatenating layer $l - 1$ and the same layer at previous timestep $t - 1$. For $l = 1$, the previous layer consists of the input grid. The update step consists in a multi-scale convolutional gated recurrent unit (dynamic memory) with added pixel-wise biases (static memory).

4.2.3 Dataset

To analyse the capacity of these different architectural choices in aiding the tracking task, we consider a robotic platform equipped with a *Hokuyo UTM-30LX* 2D laser scanner. We collect 75 minutes of data acquired while stationary in the middle of a busy urban intersection, as depicted in Figure 4.7. This was subsampled at 8Hz and split into a 65 minute unsupervised training set and a 10 minute long test set to measure the occupancy prediction performance. The dataset is challenging due to the dense traffic 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.

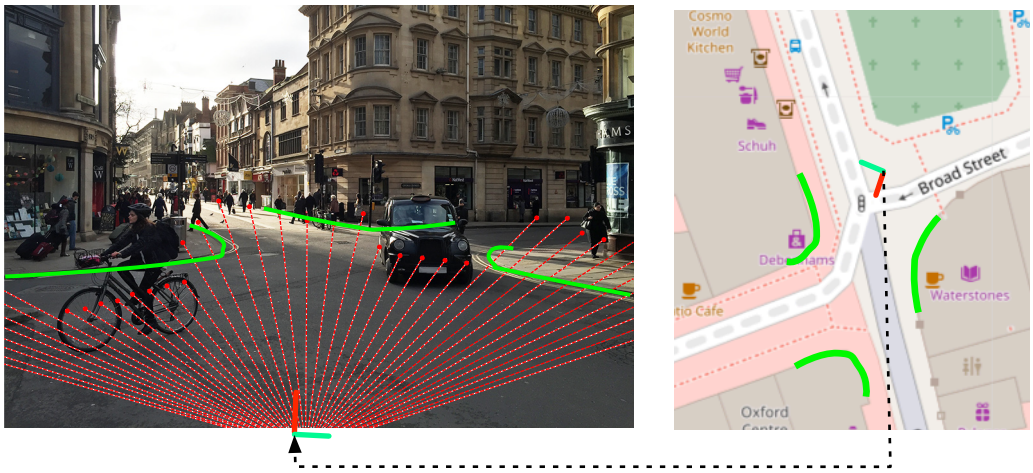


Figure 4.7: Location of the experiment from the robot's point of view with a superimposed illustration of laser measurements (left) and Oxford location in Open Street Maps (right). The area is occupied by a variety of different dynamic objects such as pedestrians, cyclists and cars.

Preparation The occluded occupancy map used as input to the network $\mathbf{x}_t$ is computed from raw 2D laser scans by performing ray-tracing. Cells corresponding to a laser range measurement are marked as occupied, all cells from the sensor origin to the ray ending at the occupied cell are marked as free, and cells beyond the ray are marked to be unobserved. Thus, a 100×100 grid is constructed, with the origin centred at the sensor location. Each cell covers a 20×20 cm^2 area, and the input grid thus spans a total area of 20×20 m^2.

4.2.4 Network Training

The network is presented the input sequence $\mathbf{x}_{1:t}$ and trained to predict the next n input frames $\mathbf{x}_{t+1:t+n}$. Since the loss encourages the model to predict future states of the environment, the model is forced to capture the motion of each object in the hidden representation.

The training set was split into mini-batch sequences of 40 frames (5 seconds). For every mini-batch, the network was shown 10 frames and trained to predict the next 10 frames, leading to two such sequences per 40-frame mini-batch. This length of sequence covers the typical lengths of occlusions observed but could be tuned accordingly on other datasets.

Our model architectures are implemented in the Lua programming language using the Torch library. Models are trained on an Nvidia Tesla K40 GPU until convergence, using the unsupervised training procedure described in the preliminary Section 2.2.5. The Adagrad optimizer is used for stochastic gradient descent with an initial learning rate of 0.01, and the loss is monitored on the validation set to perform early stopping and prevent over-fitting. A single forward pass and a 3-second forward prediction through the model with an $8Hz$ input stream takes 4 and 130 ms on an NVIDIA GeForce GTX Titan GPU with 6Gb of RAM making it suitable for real-time application.

4.2.5 Results

In the following sections we benchmark our proposed solution to an existing model-free solution, and investigate the contributions of each of our proposed network extensions.

Benchmarking Against an Existing Approach

To justify the proposed end-to-end system, we first assess its performance in comparison to a more traditional multi-stage pipeline. In particular, we evaluate the ability of the model to predict the future location of dynamic objects and compare it with a recently proposed state-of-the-art approach [Wang et al., 2015] based on

model-free tracking of dynamic objects using a Kalman filter. This method also operates on raw laser scans, but explicitly performs clustering, data association, and velocity estimation of moving objects. This information is then used to predict the positions of individual points in the future. The parameters of this method are tuned on the training set, and the outputs on the test set are converted into occupancy grids for comparison with the proposed method.

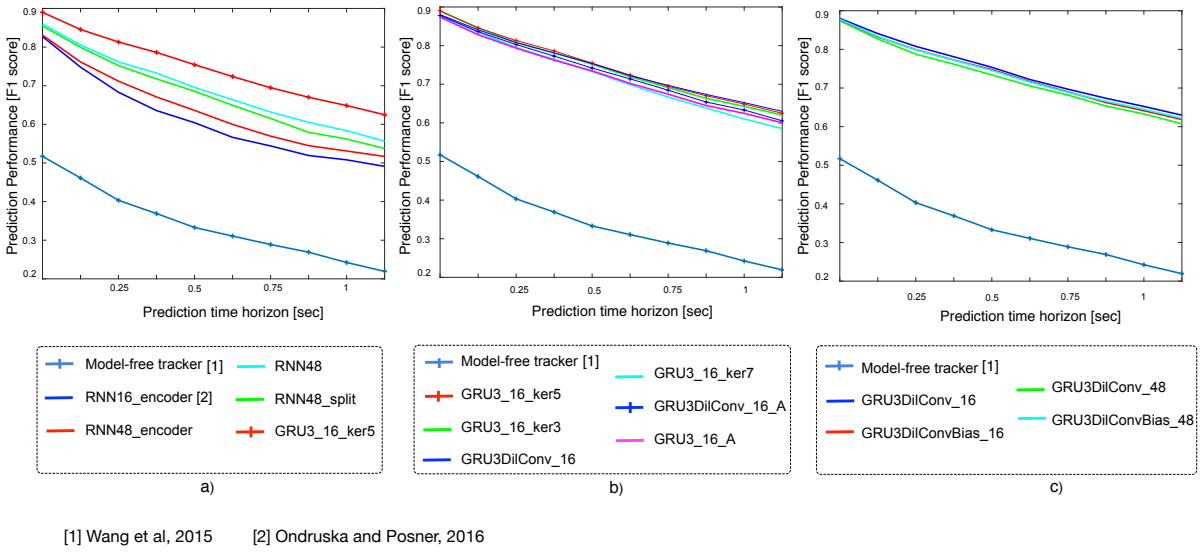


Figure 4.8: F1 scores of network architectures when attempting to predict the future occupancy of the scene in a 1.25 second time horizon. The F1 measure is computed with a threshold of 0.5 when considering a cell to be predicted as occupied or free. Investigating model capacity and memory unit suggests that incorporating *gated recurrent units* accounts for the most significant performance gain. (b) Investigating output receptive field and dilated convolutions suggests that dilated convolutions achieve commensurate performance in a parameter-efficient manner and that small output receptive fields (7×7) suffice. (c) Investigating static biases and full hidden state decoding suggests that these do not affect model performance.

Quantitative results

In the first experiment, we look to quantify the gain in performance achieved by scaling up the original *Deep Tracking* network with the proposed architecture of Figure 4.3 on page 83, and individually explore the benefits of each of the improvements proposed. Accordingly, we compare the performance of a number of different architectures, ranging from the original [Ondruska and Posner, 2016]

to the proposed model, on the task of predicting the observable near-future scene occupancy $\mathbf{y}'_{t+1:t+n}$, given the input sequence $\mathbf{x}_{1:t}$.

Figure 4.8 reports the F1 measures computed on each of $n = 10$ blacked out future frames, given the 10 frames in the past, for numerous model architectures. Intuitively, the predictions of all models are poorer over time, as the uncertainty of the state of the world increases with the prediction horizon. There is a considerable increase in performance when utilising Deep Tracking architectures compared to the model-free tracking pipeline of **?**], illustrating the efficacy of the proposed end-to-end approach.

Dynamic Memory Figure 4.8(a) investigates the effect of scaling up the original deep tracking framework of [Ondruska and Posner, 2016] in terms of model capacity and memory unit (GRU vs RNN). We find that merely increasing the capacity of the original RNN architecture (RNN16_encoder) from 16 to 48 feature maps (RNN48_encoder) does not yield any significant performance increase, indicating that the capacity of the network is not the limiting factor. Removing the 8 feature map encoder however increases performance (RNN48) suggesting there is no added value in encoding the input. We discard the encoder in all subsequent models.
We investigate the effect of a multi-stage processing of the input by separating the 48 feature maps of the hidden state into three hidden layers of 16 feature maps each (RNN48_split), similarly to the architecture proposed in Figure 4.3. This model increases the output receptive field from 5×5 to 13×13 and yields poorer performance. However, we find that changing the internal memory state from RNN to *gated recurrent units* leads to the most significant performance increase (GRU3_16_ker5). Similarly to RNN48_split, GRU3_16_ker5 decodes only the last 16 feature maps of the last hidden layer to the output occupancy $\mathbf{y}_t$, and considers kernels of size 5×5. We consider GRU architectures in all subsequent models.
Similarly to the empirical evaluation of Chung et al. [2014], we find that the our GRU3_16 model converges faster to a better solution than RNN48.

Dilated Convolutions Figure 4.8(b) investigates the effect of varying output receptive fields and utilising dilated convolutions. Similar to what was observed in the RNN architecture, we do not find any significant model performance increase between an output receptive field of 7×7 (GRU3_16_ker3 with 3×3 kernels) and output receptive field of 13×13 (GRU3_16_ker5 with 5×5 kernels). Building upon GRU3_16_ker5, replacing the standard convolutions with dilated convolutions (GRU3DilConv_16) achieves similar performance for a comparable receptive field of 15×15 using 3×3 kernels. Further increase of the receptive field to 19×19 (GRU3_16_ker7 with kernels of 7×7) however results in poorer performance than that of GRU3_16_ker5. A small performance increase is obtained by reducing the number of parameters with dilated convolutions (GRU3DilConv_16_A). This model retains dense kernels of 7×7 in the feature maps of the first hidden layer, and uses dilation with kernels 3×3 in the rest of the hidden state. This maintains receptive fields of 7×7, 11×11 and 19×19 in the hidden units, which is comparable to those of GRU3_16_ker7. Performance nonetheless remains below that of its lower receptive field counterpart GRU3DilConv_16. These results suggest that a smaller output receptive field of 7×7 suffices in capturing the overall scene dynamics. We hypothesise that too large an output receptive field may contribute to a loss of resolution. Further, we find that dilated convolutions achieve similar receptive fields in a parameter-efficient manner, without loss of performance. Both dilated architectures considered here allow for a parameter factor reduction of nearly 3.
Unlike typical convolutional networks this network *does not* feature max pooling and maintains the same resolution in each layer. Max-pooling reduces computational and memory requirements by reducing the size of the feature maps. Although this is a desired characteristic for robotics application, we suggest that dilated convolutions provide additional freedom for the network to learn which features to select, where max-pooling would only select the most prominent ones.

Static Memory Figure 4.8(c) suggests that performance of the full model obtained when adding static biases (GRU3DilConvBias_16) remains commensurate to that of GRU3DilConv_16, and the learned static bias values may convey useful

information such as that of the static background layout. This is for example valuable when learning the scene semantics as presented in section 4.3.
One key departure from a traditional convolutional architecture would be to consider the full hidden state (all three layers with 16 feature maps each) when decoding to the output occupancy $\mathbf{y}_t$, rather than the feature maps in the final layer. We compare both of these cases for our best two models (GRU3DilConvBias_16 and GRU3DilConv_16), and find that these four architectures are commensurate in capturing the scene dynamics and bring favourable improvement to the original architecture. Similarly to the role of static biases however, we find that decoding the full hidden state favourably influences the model's capacity to classify the scene semantically. We discuss this further in Section 4.3.

Qualitative results

To measure the effectiveness of the desired ability to learn place-specific information, we visualise the network prediction $\mathbf{y}_t$ without providing it with any input as displayed in Figure 4.9. Even without input sensor information, the network is able to provide an estimate of the expected occupancy 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. This visualisation was performed on a higher resolution network after 10 epochs of training.

By construct, the learned static biases cannot be used at a different location. This approach can be used to provide a semantic map with occupancy patterns, and the relevant static biases can be uploaded when the robot relocalises to the same place. In this chapter we investigated pixel-wise static biases as a means of learning useful semantic information about a specific place. In Chapter 6 we extend these findings and learn to adapt pixel-wise biases to the scene at hand in an online fashion. In practice this means that this architectural choice can be used from a moving robot, provided it has access to the local scene layout.

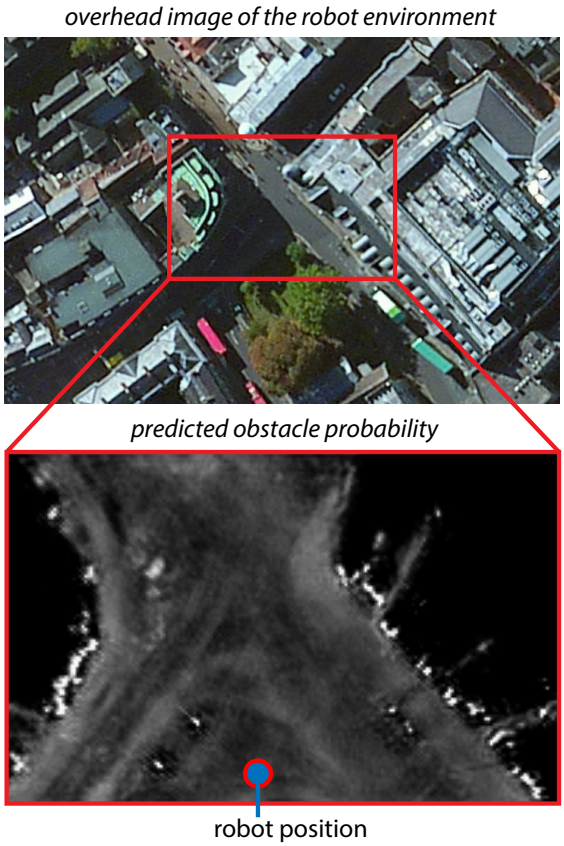


Figure 4.9: Trained network output when provided no input [**bottom**] and corresponding aerial view of the robot workspace [**top**]. The ability of the network to learn per-pixel information allows adaptation to the training environment. This allows the network to confidently predict the position of 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 of visualisation, we show here the log of the probabilities of occupation.

To better understand what the network has learned, we also perform qualitative analysis of a typical 3 second test sequence, visualising the network output and selected hidden state feature maps in Figure 4.10. The network is able to track pedestrians through full occlusion and the unobserved hallucinated tracks are represented in blue in the output sequence.

It can be seen that some of the first layer feature maps (GRU1) appear to have learned to capture the static background during the sequence with a stationary set of activations (i), and track moving pedestrians, as highlighted through the pink circles (ii). The second layer (GRU2) also captures the motion of pedestrians moving upwards to the left, while, interestingly, the GRU3 unit is activated only

on the car that appears to the top right at frame 2 (indicated by the green box). This provides empirical support for the use of dilated convolutions, which make it possible to capture patterns of varying width with limited computation.

In general, we observe that information regarding objects in the scene is captured in the hidden state, and moves spatially through the feature maps according to the motion of the object in the input.

4.3 Semantic Classification Through Inductive Transfer

In this section, we extend the tracking solution to the partial problem $p(\mathbf{y}_t|\mathbf{x}_{1:t})$ presented in preliminary Section 2.2.5, to the full problem of simultaneously estimating both occlusion-free occupancy and scene semantics $\mathbf{y}_t, \mathbf{c}_t$. We show that this can be achieved relatively easily by exploiting the knowledge the network has already learned to predict $\mathbf{y}_t$, through the principle of inductive transfer [Pan and Yang, 2010]. The significance of this is that *only a small amount* of labelled training data is needed to allow the same network to master this additional task.

4.3.1 Problem Formulation

The clue resides in the hidden representation $\mathbf{h}_t$ 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 to make is that any information necessary for the prediction of the position of the objects in the near future must be already contained in this hidden representation. The semantic class of an object falls in this category as different objects differ mainly in their shape and motion patterns. Similar to predicting the unoccluded state $\mathbf{y}_t$ from the learned state representation $\mathbf{h}_t$, extracting $\mathbf{c}_t$ can be achieved simply by building a classifier to predict $p(\mathbf{c}_t|\mathbf{h}_t)$. This is achieved by

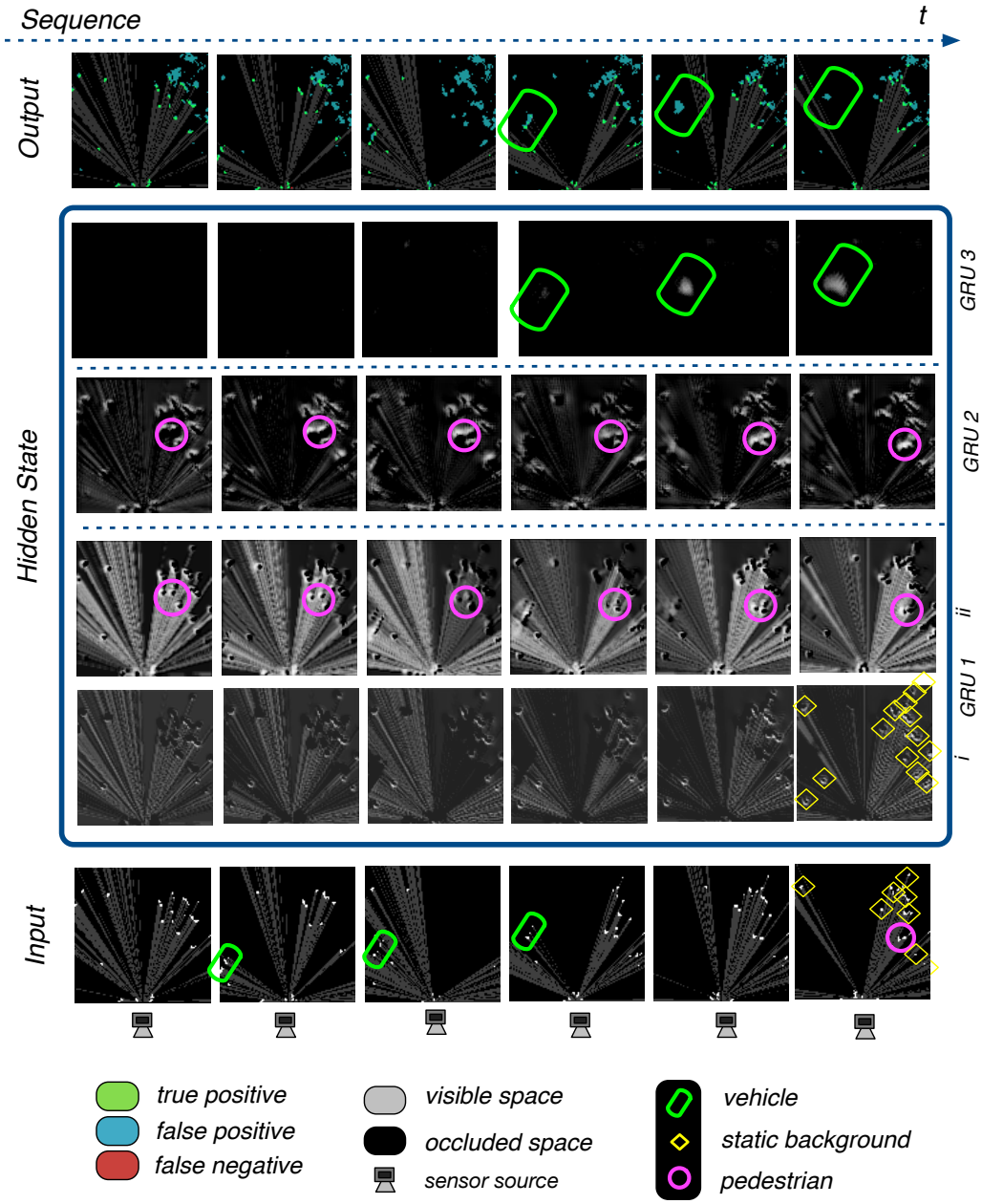


Figure 4.10: Example of outputs produced by the system (GRU3DilConv_48) along with a selection of activations in the hidden layers. The figure is best read from bottom to top, with the bottom layer providing the raw sequence input to the network along with ground truth annotation. As highlighted in colour-coded circles, the network is able to propagate the assumed motion of the objects even when in complete occlusion. The sample hidden layer activations shown highlight the fact that lower layers in the hidden units (corresponding to a low dilation of the convolutions) capture the motion of small and slow moving objects such as pedestrians (e.g pink circle) and static background (e.g yellow squares), whereas a higher level layer learns to detect moving vehicles (green contour). Best viewed on pdf.

employing a simple convolutional softmax function decoder to decode the hidden state $\mathbf{h}_t$ to a semantic grid $\mathbf{c}_t$ of 1 of K class labels.

4.3.2 Network Architecture

In order to assess the contribution of static biases in the task of semantically labelling the scene, we consider two of our best tracking models (GRU3DilConv_48), as well as its corresponding full network which additionally contains static biases (GRU3DilConvBias_48). Both networks contain gated recurrent units, dilated convolutions, and consider the full hidden state for decoding to the output.

Similarly, to assess the contribution of decoding the full hidden state vs only the 16 layers of the last hidden unit, we compare our two full models (GRU3DilConvBias_48 and GRU3DilConvBias_16) on the task of semantically labelling the scene. Both networks contain gated recurrent units, dilated convolutions, and static biases and the training procedure is the same as previously described.

The semantic decoder consists in a single layer decoding the entire state to a 4-layer softmax output for multi-class classification with kernel size 7×7. The output is reduced to a single output layer with sigmoid activation for binary classification. This decoder was chosen empirically to provide a large enough receptive field.

4.3.3 Dataset

To solve the problem of estimating both occlusion-free occupancy and scene semantics $\mathbf{h}_t, \mathbf{c}_t$ we hand-label 1000 scans from the training set into four classes for the purpose of network semantic training, and 400 scans from the test set for the evaluation of its semantic classification accuracy. We consider two classification tasks. When performing multi-class semantic prediction, the classes considered are: *pedestrian*, *vehicle*, *cyclist*, and *static obstacle*. In this task, the cyclist and pedestrian classes also contain a few examples of temporarily static members. When performing binary classification, we consider a static object class, and a dynamic object class. In this classification problem, temporarily static pedestrians and cyclists are considered members of the static class along with the permanently static background.

4.3.4 Network Training

We train a classifier using the same training procedure as the unsupervised tracking task. The training process is illustrated in Figure 4.11. This is to ensure that the model is forced to utilise the memory and be able to predict the semantics of objects in occlusion. Unlike the unsupervised task however, the classification errors corresponding to the visible part of the scene at $\mathbf{x}_{t+n}$ are *only* back-propagated through the classifier's decoder layer. This is unlike the unsupervised tracking task where errors were back-propagated in time through the hidden states.

Models are trained until convergence using the 3 minute long labelled dataset, and we monitor the loss on the validation set to perform early stopping and prevent over-fitting. The Adagrad optimizer is used for stochastic gradient descent with an initial learning rate parameter of 0.01. Because the dataset is skewed and contains many more objects of a particular type, e.g. pedestrians, we used a weighting scheme assigning class weights equal to the inverse of class frequency. We do not consider any class weighting when training on the binary classification task (static vs. dynamic).

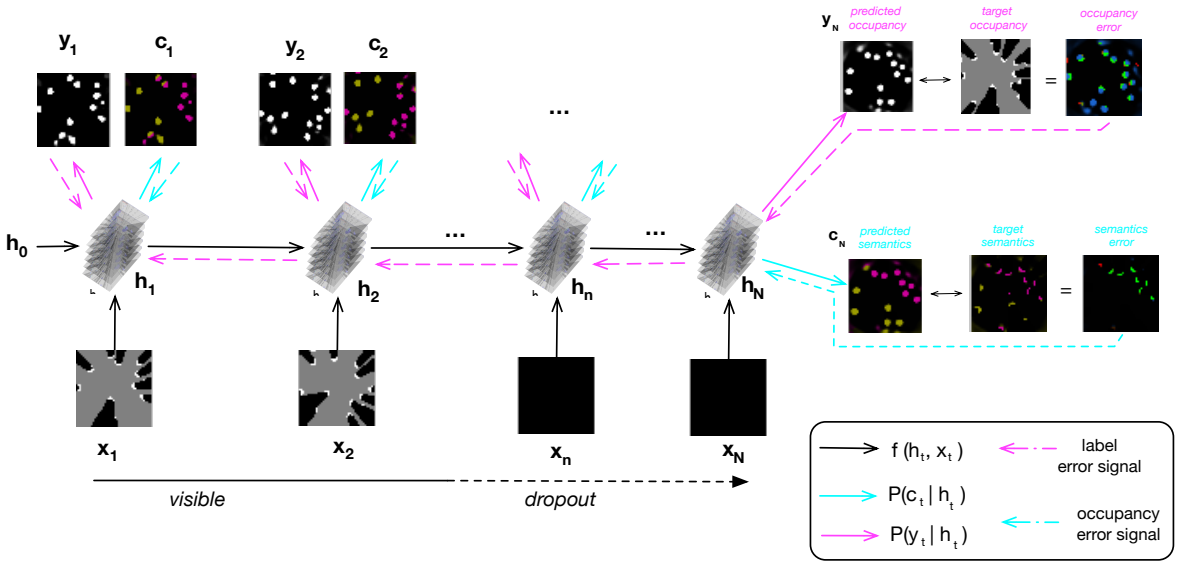


Figure 4.11: Training of the recurrent neural network to produce both space occupancy $\mathbf{y}_t$ and semantic labels $\mathbf{c}_t$. The network is trained to predict outputs consistent with future inputs. This allows training without the need for ground-truth information of the full, unoccluded scene. First, the network learns how to track by predicting correct occupancy using large amounts of unlabeled data, then a small set of labelled data is used to induce semantic classification.

All models are first pre-trained on the task of predicting the unoccluded occupancy scene on the previously described dataset.

4.3.5 Results

We quantitatively and qualitatively observe the network's ability to semantically classify the scene in terms of static and dynamc objects, as well as pedestrians, vehicles, and cyclists.

Static vs dynamic objects

To quantify the network's ability to classify scene semantics we first consider a binary classification of the predicted output into dynamic and static classes. We compare the F1 measure for the two models (GRU3DilConvBias_48 and GRU3DilConv_48) with a 0.5 threshold for predicting dynamic objects. We find that both models are commensurate in the binary prediction of static vs. dynamic obstacles. The full model with static biases predicts the dynamic objects with a mean F1 score of 0.92 while the model without static biases performs with an F1 score of 0.91.

Qualitative Results Figure 4.12 illustrates a six second sequence of binary classification prediction (static vs. dynamic) for both models, along with the visible ground truth (Figure 4.12.a). As can been seen, both models accurately capture the overall dynamics of the scene, consistently predicting the static background and moving pedestrians on the pavement and road. Circled in red are two temporarily *static* pedestrians which both models are able to capture. It however remains a challenge for both models to consistently label temporarily static obstacles as such as illustrated by the unclear classification of the dashed orange circles. We find that the full model is able to utilise its dynamic memory to capture temporarily static objects despite having the ability to rely on its static memory to predict the permanent static background. This is supported by the high probabilities of cell occupancy provided by the static biases in Figure 4.9. We hypothesise that the accuracy of classification of temporary static objects would further improve for both models with more such examples in the data.

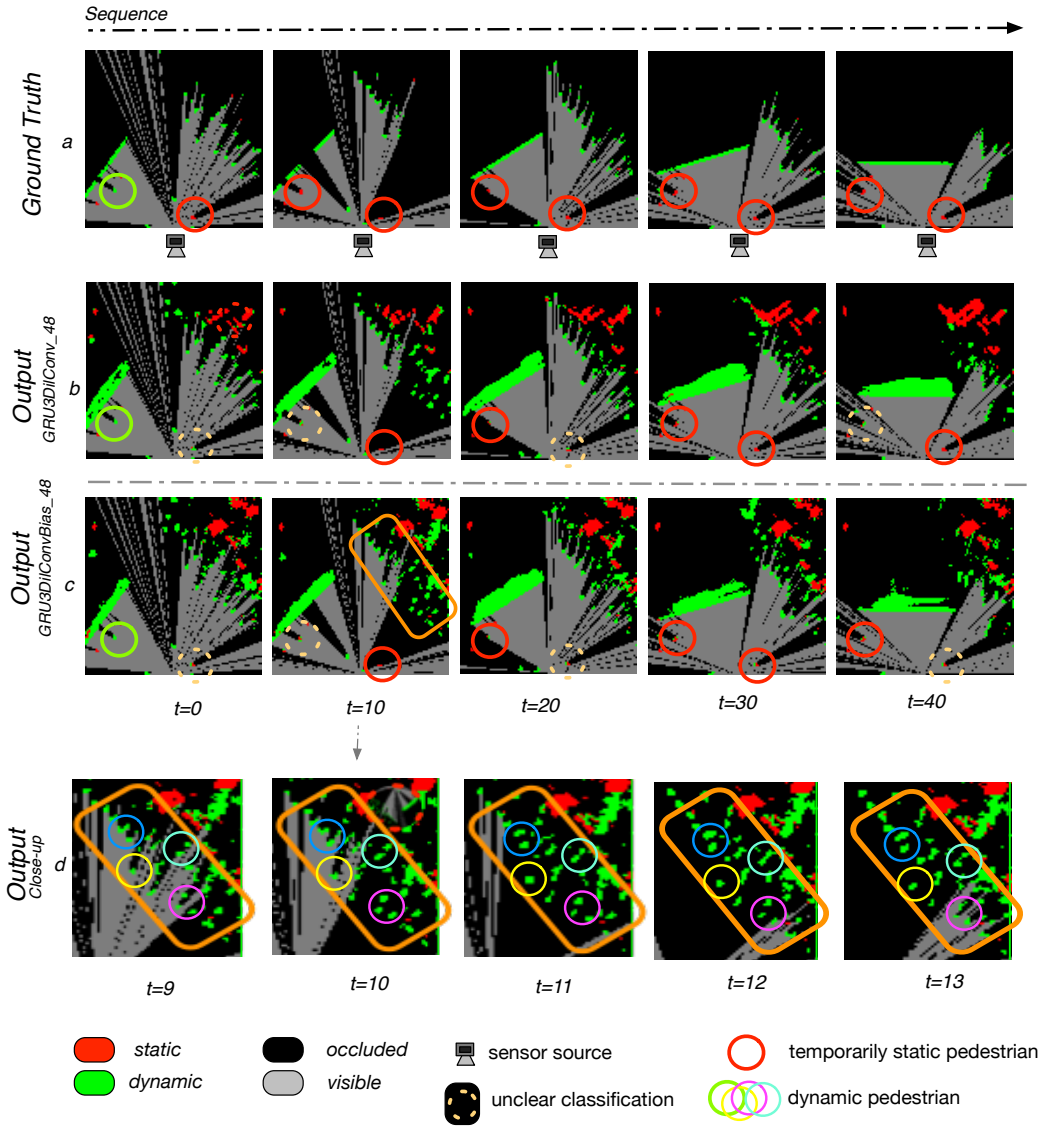


Figure 4.12: Example sequence (6 sec) of binary classification output (static vs. dynamic) for the full model (GRUDilConvBias_48, c) and best model (GRU3DilConv_48, b). Red labels refer to static obstacles while green illustrates dynamic obstacles. The ground truth is shown for comparison (a). Both models capture the static background and motion of pedestrians in the scene (orange contour close-up, d). The red circles show examples of two temporarily static pedestrians which both models capture well. The dashed orange circles represent unclear classifications of the static pedestrians which reflects the difficulty of capturing momentarily static objects. Best viewed on pdf.

Multiple object classes

To quantify the network's ability to classify scene semantics, we compute the Intersection over Union (IoU) for individual and combined classes, as well as compute the confusion matrices for three of our best models (GRU3DilConvBias_48,

GRU3DilConvBias_16, and GRU3DilConv_48). These models differ by their use or lack of static biases, and by their decoding of either the last unit, or the whole of the hidden units. Results for class IoU and class confusion matrices are respectively shown in Table 4.1 and Figure 4.14. As can be seen, GRU3DilConvBias_48, which contains static biases and decodes the entire hidden state, produces the best classification for all classes considered, reaching an overall classification performance of 0.93 (Table 4.1a.).

We further investigate the confusion of classes by observing the recall of classification illustrated in the row normalised table of Figure 4.14a. The lowest precision, illustrated in the column normalised table of Figure 4.14a, additionally suggests that one third (31.8%) of the model's cyclist predictions correspond to pedestrians. This is a confusion that appears in all models pre-trained on the tracking task (Figure 4.14a,b,c and Table 4.1a,b,c). We suggest this is because in addition to exhibiting similar shapes in 2D laser data, cyclists sometimes stop or cycle more slowly, thus exhibiting the behaviour of a pedestrian. Inversely, some pedestrians were observed to run across the scene in both training and testing sets, exhibiting the velocity of a cyclist more than that of a strolling pedestrian.

Our second best model (GRU3DilConv_48), which does not contain static memory, achieves lower performance on all classes (Table 4.1). The background class particularly achieves a lower IoU metric of 0.82 (vs. 0.90). We further observe enhanced confusion between the static background class and respectively the pedestrian and cyclist classes as suggested by the confusion matrices. We hypothesise that the static biases inform the classifier that cyclists evolve on the road, whereas pedestrians are mainly on the pavement and away from static obstacles. These results suggest a positive contribution of static biases to scene understanding.

Performance of the full model when only considering the last output unit in the decoder (GRU3DilConvBias_16) is significantly lower than when considering the full hidden state (GRU3DilConvBias_48), with an overall IoU metric of 0.81. We find that there is more confusion between the pedestrian, cyclist and vehicle classes. We suggest that the three hidden layers learn to recognise objects of different

sizes and motion patterns, consistent with their different receptive fields. Although information gets passed forward through the units of the hidden state, we hypothesise that allowing the network to directly access information from the varying scales of the input, may assist positively in discriminating between the semantic classes.

The three models present some small amount of confusion between the static background and pedestrian classes, although static biases contribute favourably in discriminating between the two. As pedestrians walk very close and through the static background on the pavement (mostly shops and buildings), we suggest that these errors are very sensitive to mis-labelling the ground truth.

Class	Background	Pedestrian	Cyclist	Vehicle	Global
a. GRU3DilConvBias_48	0.90	0.94	0.58	0.98	0.93
b. GRU3DilConv_48	0.82	0.91	0.55	0.97	0.89
c.GRU3DilConvBias_16	0.85	0.84	0.35	0.86	0.81
d.GRU3DilConvBias_48 (no pre-training on tracking)	0.93	0.66	0.1	0.66	0.62

Table 4.1: Intersection over union (IoU) for individual classes, the combined Pedestrian and Cyclist classes, and all classes combined, when considering the GRU3DilConvBias model. As explicited by the confusion matrices of Figure 4.14 (top matrices), the main confusion lies between the cyclist and pedestrian classes as they exhibit similar shapes and velocities in 2D laser data.

.

Qualitative Results A typical input sequence and the corresponding predicted network output as given by the best tracking network (GRU3DilConv_48) and full model (GRU3DilConvBias_48) are shown in Figure 4.13. The network is able to uncover the unoccluded scene occupancy $\mathbf{y}_t$ and object labels $\mathbf{c}_t$. Despite a long occlusion by a turning bus, the full model is particularly able to retain the position and presence of the occluded pedestrians and static background. The four classes in the scene are accurately captured by both models.

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.

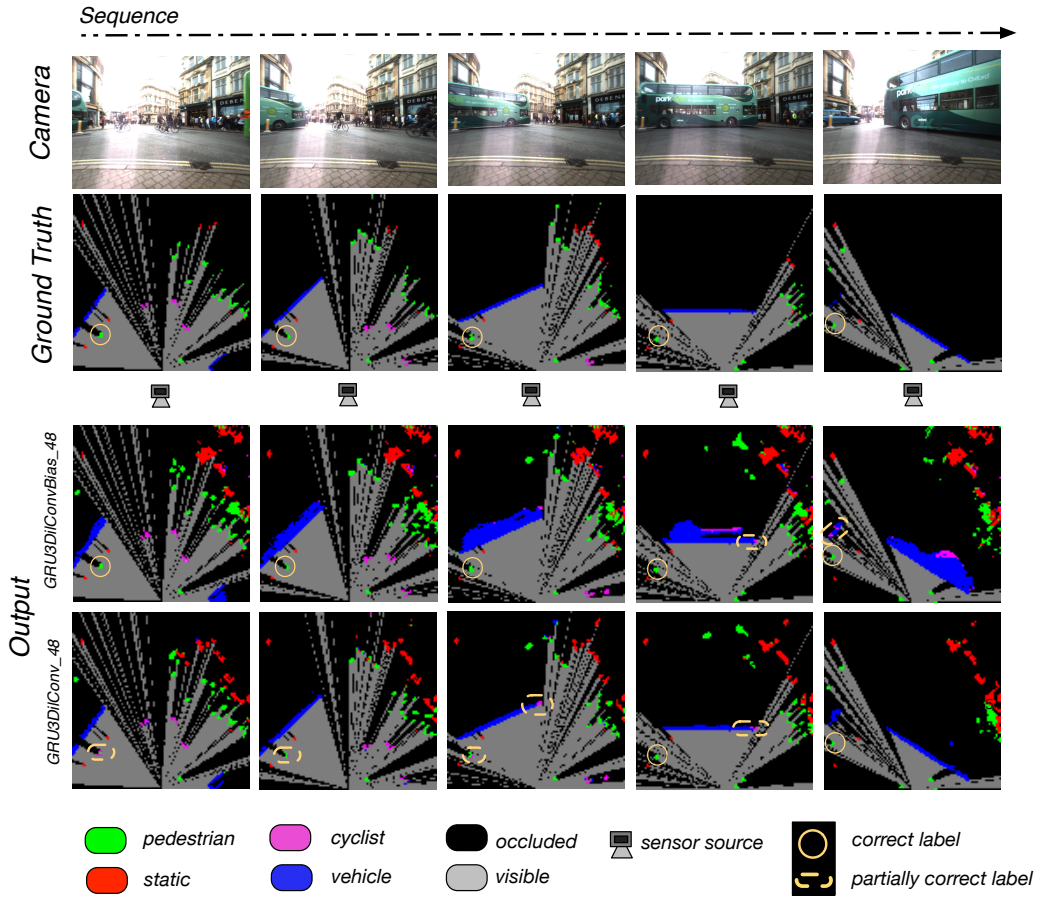


Figure 4.13: Example sequence (6 sec) of scene classification as produced by GRU3DilConvBias_48 and GRU3DilConv_48 along with the available ground truth camera and laser (top two rows). Despite a long and full occlusion by the turning bus, both models retain an accurate understanding of the scene. A pedestrian is partially and temporarily confused as a cyclist in the GRU3DilConv_48 model (orange circle). This pedestrian is crossing the road at a traffic light (not visible in the camera view) suggesting that the model with static memory may have a prior for pedestrian presence at this crossing. Both models present very partial and limited labelling of the passing vehicles as a cyclist. Best viewed on pdf.

Value of Inductive Transfer

We verify the value of the proposed inductive transfer of knowledge by comparing these results to that of learning the semantics of the scene when the model *has not* been pre-trained on the task of tracking. Results can be seen in Figure 4.14d and Table 4.1d, and are qualitatively illustrated in Figure 4.15. The accuracy of the system only achieves an overall IoU metric of 0.62 due to high confusion between the dynamic classes. The permanent static background and buses, characterised

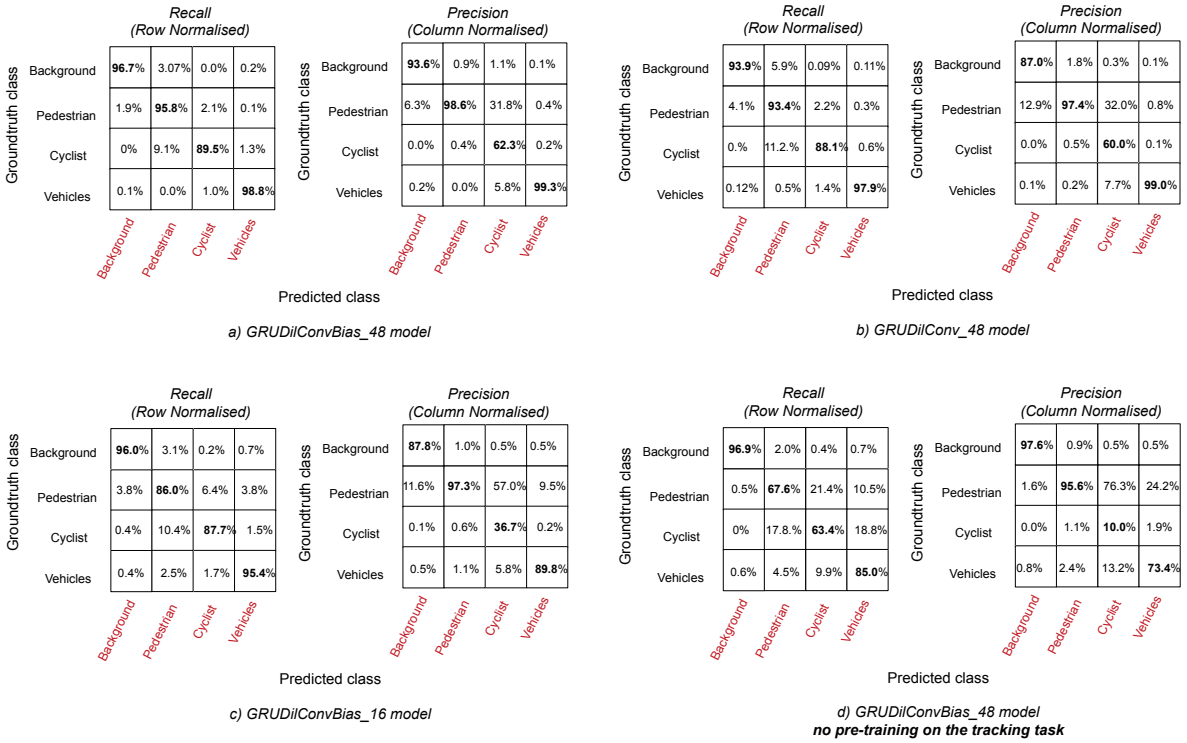


Figure 4.14: The confusion matrices of the semantic classification performance. Top Left a): GRU3DilConvBias_48 represents the full model and decodes the entire hidden state. This model achieves the highest classification results. Top Right b): GRU3DilConv_48 decodes the entire hidden state but does not consider static memory. This model is less able to differentiate the dynamic classes from the static background. Bottom Left c): GRU3DilConvBias_16 decodes only the last 16 hidden layers. This model is less able to differentiate cyclists and pedestrians from the other classes which suggests that high resolution information may be diluted in the higher layers of the network. Bottom Right d): GRU3BiasDilConv_48 decodes the full hidden state but was not been trained to capture the scene dynamics in a prior task of tracking. The static biases appear to positively contribute to classifying the static background but the dynamic classes suffer high confusion.

by long segments in the data, are however well captured. This is reflected in high values for the row-normalised matrix which suggests that 96.9% and 85.0% of these two classes are correctly classified. Pedestrians and cyclists are however notably confused with vehicles (10.5% and 18.8% mis-labelling respectively) which constitutes the greatest error in the context of safe autonomous robotics. Finally, this model was significantly slower to converge than it's pre-trained counterpart ($\times 4$ training epochs). This demonstrates that $\mathbf{h}_t$ offers a powerful semantic descriptor of the scene and can be used as input for accurate semantic classification.

In our application we assume the number K of semantic classes is known a priori. This could be extended to include a $K+1$ class of "unknown/other" to account for possible other classes. The distribution of semantic predictions could also be studied to detect possible patterns of ambiguity: are static cyclists labelled equally probable as pedestrians and cyclists? at which point does a temporary static object get labelled as static background? Currently the supervised task of decoding the scene semantics is only backpropagated in the decoder weights. It would be interesting to investigate whether feeding back the predicted output semantics into the input at the next timestep helps the tracking. This may risk propagating model errors and producing semantic dirft. Alternatively, future work could look into whether the supervised task of decoding the scene semantics might in return improve the tracking task by back propagating semantic loss throug the hidden state and time.

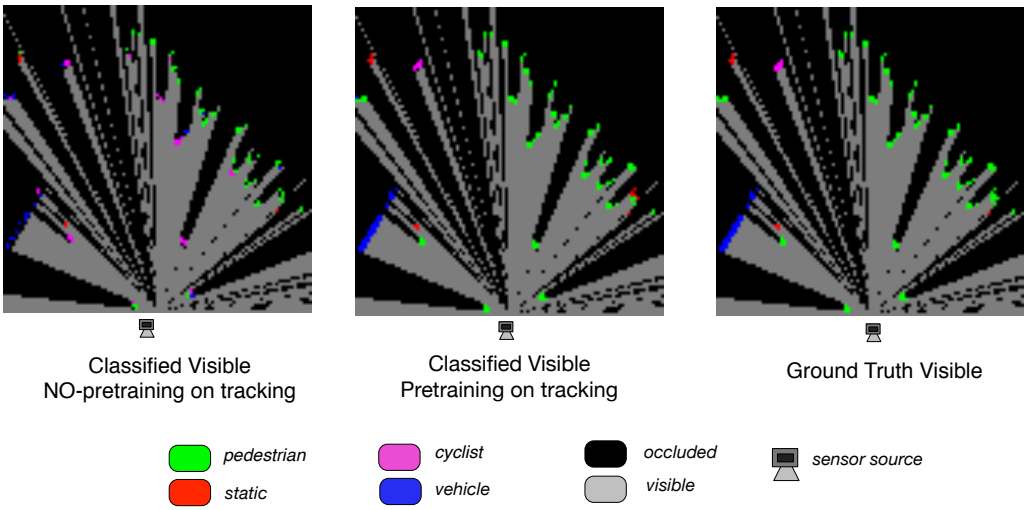


Figure 4.15: Visible scene semantics as classified by the full model (GRU3DilConvBias_48) without pre-training on tracking (Left), by the full model with pre-training on tracking (Middle), and by the available visible ground truth (Right). For a fair comparison that does not require tracking through occlusion, we only show the predicted labels of the visible occupancy grid. To the exception of the incoming bus and part of the static background which are accurately labelled by the untrained network, the scene demonstrates the powerful semantic descriptor offered by the $\mathbf{h}_t$.

4.4 Conclusion and Future Work

In this chapter we extended 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 trained in a self-supervised manner, we demonstrate greater performance compared to a state-of-the-art model free tracking pipeline whilst substantially reducing the requirement for model design. We suggest a positive contribution of gated recurrent units, dilated convolutions and static biases to the more traditional recurrent architecture used in the original Deep Tracking framework. The most notable step change in terms of improvement on the tracking task is observed when using gated recurrent hidden units. With little extra labelled data we further show that the learned state representation can be used to semantically label the scene.

This framework could be further developed in a number of ways. In its essence, it naturally lends itself to providing occupancy maps for planning through space and time, though in practice it remains a "black box" system with no measure of trust. We suggest two future directions for linking this system to a planner. On the one hand this deep tracker could be used as a "super sensor" and considered as such. The provided pixel output occupancy can directly constitute an input to a traditional model-free or model-based tracking pipeline similarly to raw laser scans, though with continuity in occlusion. Alternatively, in contrast to the current deterministic framework trained to produce only one output sequence, the system could be developed to synthesize a number of likely future frames in a probabilistic manner similar to Xue et al. [2016]. Future work could also look into estimating model uncertainty to provide a planner with confidence intervals on the output predictions (see for example dropout as a Bayesian approximation [Gal and Ghahramani, 2016]) and investigate what the model has learned and how it reasons (see for example Fuchs et al. [2018]). Finally with adequate labelled data the network could also learn to translate the learned implicit data association into bounding boxes similar to Hoermann et al. [2018].

In terms of architecture and training procedure, we currently consider a Cartesian grid with full input dropout/occlusion during training. Though full occlusion of the sensor occurs rarely naturally, it forces the hidden state to capture the dynamics of the entire scene in order to predict into the future. This also makes use of all the available visible data as a self-supervisory signal during training. It is unclear whether training with smaller and more naturally imposed occlusions could facilitate learning. Simulating natural occlusions would require adding objects in the scene which could both get in the way of dynamic objects and create unnatural setups. However future work could investigate if occluding specific objects in the scene could aid in learning object interactions. Laser scans are further projected into a Cartesian grid which introduces loss of resolution with increasing distance from the laser source. Further work could consider projecting the laser information into a polar $[R, \theta]$ 2D grid corresponding to the return distance of the various laser beams to maintain resolution.

In the next Chapter we address the limitations of the current system implemented in a static frame, and extend the framework to operate from a moving platform.

We must perceive in order to move, but we must also move in order to perceive

— J.J.Gibson

5 Learning to Predict the World State from a Moving Vehicle

In the previous chapter, we presented a first enhancement to a perception system with the aim of increasing a robot's situational awareness in a real-world setting. We demonstrated how a static laser-based tracking system could be augmented to build a representation of the environment from a sequence of instantaneous and occluded laser measurements. Crucially, compared to a traditional tracking pipeline, the proposed framework is able to predict scene occupancy through occlusion, and into the future. In this chapter, we further extend the framework to operate on a *moving platform*, to enable deployment in more realistic real world field robotic settings where robots actively need to move in their environment.

5.1 Motivation

It is unreasonable to envisage field robotics without robot motion. Truly capable and useful robots need to move in their environment to conduct tasks such as inspection, exploration, and transportation. As we are interested in developing predictive sensing capabilities for robotic systems working in the field, we are motivated to extend our tracking framework so that it can be deployed on a moving platform. Beyond usability in the field, we argue that motion is a crucial component

to learning about the world, and that it is important to develop frameworks able to handle and leverage robot motion if we are to build robots that have increased situational awareness.

Moving through a dynamic world brings additional challenges to a tracking system. As a robot changes position in its environment, its perception of the world changes as well. Viewed from a range sensor, static objects appear at different positions, are viewed from different angles, and can present a different shape depending on the robot's motion. Dynamic scenes add a further layer of complexity, as they create an array of complex relative motions between the moving vehicle and moving objects in the environment. If vehicle motion is ignored in such scenarios, a tracker model is forced to learn all possible motion interactions between the vehicle and its environment as if the vehicle were stationary. This is neither scalable, nor satisfactory.

In traditional tracking pipelines, trackers typically decouple the vehicle motion from the dynamic objects' motion using vehicle ego-motion estimates provided by a visual odometry system. At every time step of the tracking algorithm, object states are first predicted using the motion models, then transformed into the new robot frame at time t using the ego-motion estimate, and finally corrected with the incoming sensor measurement at time t. This three-step update is summarised in Algorithm 2, and would typically account for noise in both robot motion estimate and measurement.

Algorithm 2: Tracking Belief State Update with Vehicle Ego-Motion

$\mathbf{h}_{t-1} \leftarrow$ Object states at time $t-1$;
$T_{t,t-1} \leftarrow$ Ego-motion from source $(t-1)$ to destination (t) frames;
$\mathbf{x}_t \leftarrow$ Measurement at time t;
$\epsilon_t, \delta_t \leftarrow$ robot motion and observation noise;
while t **do**
  $\mathbf{h}_t^{'} = f_1(\mathbf{h}_{t-1}, dt) \leftarrow$ State prediction with motion models;
  $\mathbf{h}_t^{''} = f_2(\mathbf{h}_t^{'}, T_{t,t-1}) + \epsilon_t \leftarrow$ State correction with ego-motion;
  $\mathbf{h}_t = f_3(\mathbf{h}_t^{''}, \mathbf{x}_t) + \delta_t \leftarrow$ State correction with measurement update;
end while
return $\mathbf{h}_t$;

Our tracking framework however does not explicitly predict object states but rather captures a latent representation of the world from an incoming stream of laser data and directly produces unoccluded world occupancy grids. Though this is the strength of the deep tracking framework, it requires adapting the ego-motion update presented above. We revert back to the original two-step belief estimation of the dynamic Bayes formulation introduced in Section 2.2.3. Estimating the state of the world at timestep t consists in first *predicting* the state of the world after taking action $\mathbf{u}_t$:

$$\overline{bel}(\mathbf{h}_t) = \int p(\mathbf{h}_t|\mathbf{h}_{t-1}, \mathbf{u}_t) bel(\mathbf{h}_{t-1}) d\mathbf{h}_{t-1}$$

and then updating this belief after incorporating measurement $\mathbf{x}_t$:

$$bel(\mathbf{h}_t) = \eta p(\mathbf{x}_t|\mathbf{h}_t)\overline{bel}(\mathbf{h}_t)$$

where η is a normalizing constant for the distribution to integrate to 1.

Framed in our deep tracking formulation, we represent the corresponding graphical model in Figure 5.1. Compared with the original Deep Tracking graph of Figure 2.7, we incorporate the robot action $\mathbf{u}_t$ as the ego-motion transform $T_{t,t-1}$. Following the convention detailed in Section 2.3.1 of our preliminary chapter, we represent transforms as $T_{\text{destination,source}}$; transform $T_{t-1,t}$ therefore represents the motion of the robot source frame at time t into the robot destination frame at time $t-1$.

As our work is embedded in robotic systems, we consider that this ego-motion estimate is provided by a visual odometry module, and can be used as a proxy for the robot's actions in the world. At time t, $\mathbf{u}_t$ can then be estimated to be the transform $T_{t,t-1}$, representing the motion of the robot from $t-1$ to t as viewed in reference frame at t. From $t-1$ to t the new hidden state $\mathbf{h}_t$ results from the state transition of $\mathbf{h}_{t-1}$ and robot motion $T_{t,t-1}$. This new state produces an unoccluded state of the world $\mathbf{y}_t$ which is partially observed by the robot $\mathbf{x}_t$.

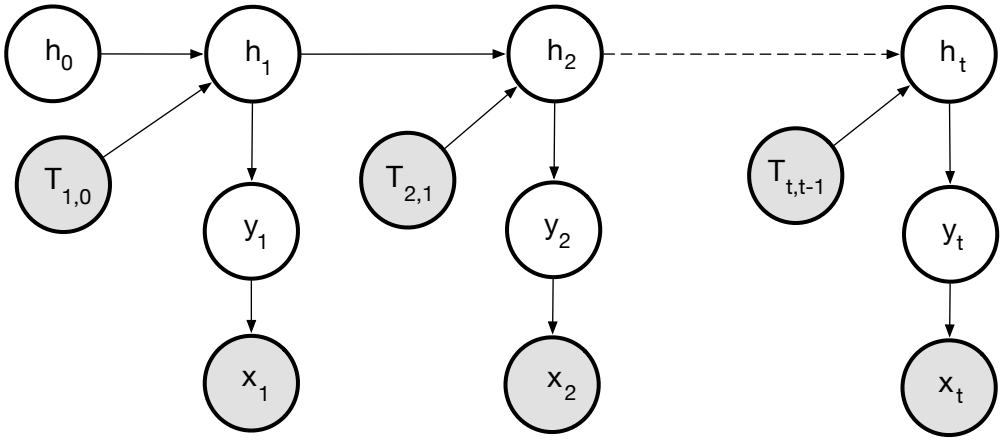


Figure 5.1: Graphical model when tracking from a moving vehicle. The graph relates to the dynamic Bayes network, with the robot control input $\mathbf{u}_t$ represented as the ego-motion between two timesteps, $T_{t,t-1}$. Compare to the original Deep Tracking graph of Figure 2.7.

With this formulation, we can decouple the motion of the vehicle from the motion of dynamic objects in the environment by adopting a two-step update of the robot's belief from $t-1$ to t which we illustrate in Figure 5.2.

We first produce an updated belief $\mathbf{h}_t^{'}$ considering the robot's motion through the scene from $t-1$ to t:

$$\mathbf{h}_t^{'} = f_1(\mathbf{h}_{t-1}, T_{t,t-1})$$

This is a reasonable step as we note that the hidden state representation of the world is centred on the robot frame and spatially coherent with the input and output grids. With this in mind, we choose to forward transform the memory grid from robot frame $t-1$ to robot frame t so that it remains spatially coherent with the new observation $\mathbf{x}_t$.

To do so we draw inspiration from Spatial Transformer Networks Jaderberg et al. [2015] originally developed to *learn* and *apply* transformations to network feature maps. A notable difference is that as our tracker is embedded in a robotic system, we do not need to explicitly *learn* the transform to apply to the feature maps, but rather we can utilise the ego-motion readily available from a visual odometry system. This first hidden state update is featured in purple in Figure 5.2.

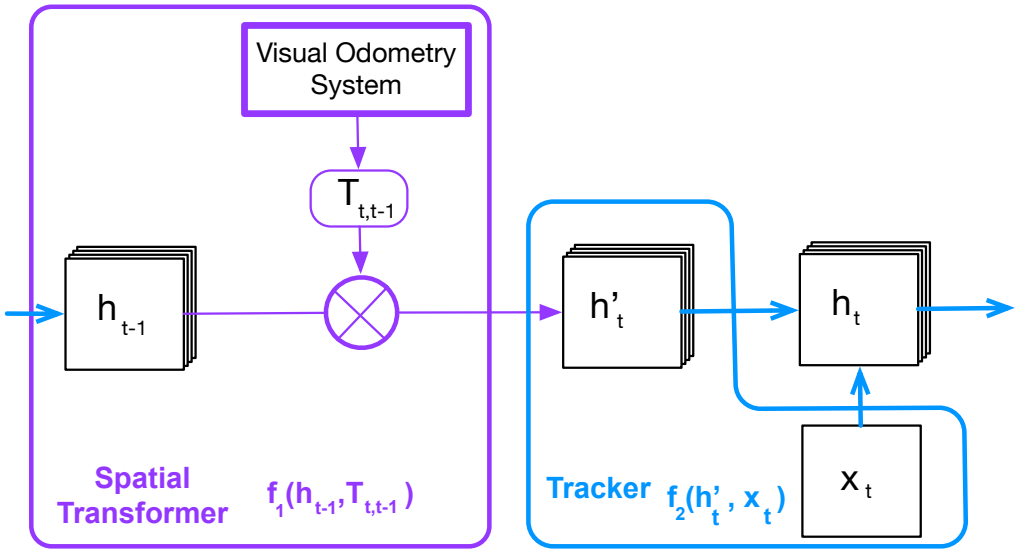


Figure 5.2: Two-step update of the hidden state when tracking from a moving sensor. The hidden representation $\mathbf{h}_{t-1}$ is first forward transformed into reference frame at time t with $f_1(\mathbf{h}_{t-1}, T_{t,t-1})$ using the ego-motion estimate $T_{t,t-1}$ to produce $\mathbf{h}_t^{'}$. We then perform the belief update with $f_2(\mathbf{h}_t^{'}, \mathbf{x}_t)$ using the incoming observation $\mathbf{x}_t$ similarly to the static sensor setting. This update is *learned* during the training phase.

We then update the network's belief by incorporating the sensor measurement $\mathbf{x}_t$ and obtaining $\mathbf{h}_t$:

$$\mathbf{h}_t = f_2(\mathbf{h}_t^{'}, \mathbf{x}_t).$$

This update step is *learned* similarly to the *static* sensor approach, and featured in blue in Figure 5.2. The corresponding pseudo-code is represented in Algorithm 3.

Algorithm 3: Deep Tracker Update with Vehicle Ego-Motion

$\mathbf{h}_{t-1} \leftarrow$ Object states at time $t-1$;
$T_{t,t-1} \leftarrow$ Ego-motion from source $(t-1)$ to destination (t) frames;
$\mathbf{x}_t \leftarrow$ Measurement at time t;
while t **do**
  $\mathbf{h}_t^{'} = f_1(\mathbf{h}_{t-1}, T_{t,t-1}) \leftarrow$ State update according to ego-motion $T_{t,t-1}$;
  $\mathbf{h}_t = f_2(\mathbf{h}_t^{'}, \mathbf{x}_t) \leftarrow$ State prediction and update with measurement $\mathbf{x}_t$;
end while
return $\mathbf{h}_t$;

Unlike traditional pipelines which explicitly account for robot motion and observation noise, we do not need to do so. The network can learn to track from

raw laser measurements and noise-free ego-motion estimates. This is an additional benefit of the proposed deep tracking approach.

5.1.1 Contributions

In this chapter we address the need to extend our tracking framework to function on a moving platform. Though vehicle ego-motion is easily accounted for in traditional tracking pipelines where object states are explicitly tracked with global or local coordinates, the tracking here occurs implicitly in the hidden state of the network. We however turn the explicit output grid layout to our advantage and choose to transform the memory state according to vehicle ego-motion, thus carrying the belief along with the robot frame in a consistent and coherent manner. We revisit the tracking formulation to reflect the full Dynamic Bayes framework and incorporate vehicle ego-motion as robot action input $\mathbf{u}_t$, to update the belief state. We draw inspiration from Spatial Transformer Networks recently introduce by Jaderberg et al. [2015] to decouple robot ego-motion from world dynamics, and demonstrate the effectiveness of our solution on synthetic data. We also show improved tracking ability on urban real-world data collected from a moving vehicle. In doing so we provide the following contributions:

- Extension of the tracking framework to perform on a moving platform by harnessing ego-motion estimates readily available as part of a robotic system. We conduct extensive synthetic and real world analysis to show that adaptation of this traditional incorporation of ego-motion to deep architectures works effectively

- Demonstration of the effectiveness of our proposed solution on synthetic data featuring dynamic objects and a moving sensor

- Demonstration that our proposed solution improves tracking ability on 28km of real-world urban data collected from a vehicle in the Oxford centre, UK

- Demonstration on synthetic data that the state learned on the task of tracking can be used to further semantically label the scene in static and dynamic objects.

5.2 Deep Tracking from a Moving Platform

By construct, the deep tracking architecture is built such that the hidden representation is geometrically coherent with the input occupancy grid constructed from the raw laser scan. As there is no loss of spatial resolution throughout the computed layers, any information relevant to predicting the future unoccluded scene occupancy at a given spatial location in the world will be located at the spatially overlapping pixel location in the hidden feature maps. This is one of the strengths of the approach as it provides a very intuitive spatial grid representation, and the output occupancy grid can be be readily used as a spatial map for either planning or directly registered with other maps. This was embraced in the previous Chapter 4 with the use of static biases to learn place specific information.

If the robot moves from time t to $t+1$, its hidden state or "memory" at time t no longer spatially overlaps with the new incoming observation $\mathbf{x}_{t+1}$. We can adopt two different viewpoints of the robots' motion which we illustrate in Figure 5.3. On the one hand we can consider a *global frame*, where the network grid is fixed in the world frame and the robot moves through the grid between t and $t+1$ as illustrated in the top right figure. In this case, static objects maintain a constant coordinate position in the grid, and the motion of dynamic objects in the grid is independent from the robot motion. New observations $\mathbf{x}_{t+1}$ remain spatially consistent with the memory at time t if their location is transformed back into the world frame $T_{t,t+1}$. This however limits the range of motion of the robot.

Alternatively, we can consider a *local frame* where the network grid moves along with the robot, as illustrated to the bottom right of the figure. In this case, from t to $t+1$ the centre of the grid moves with the robot frame with relative pose $T_{t,t+1}$ as seen in frame at time t. Consequently, the relative position of objects as seen from the robot frame at $t+1$ will change according to both the robot

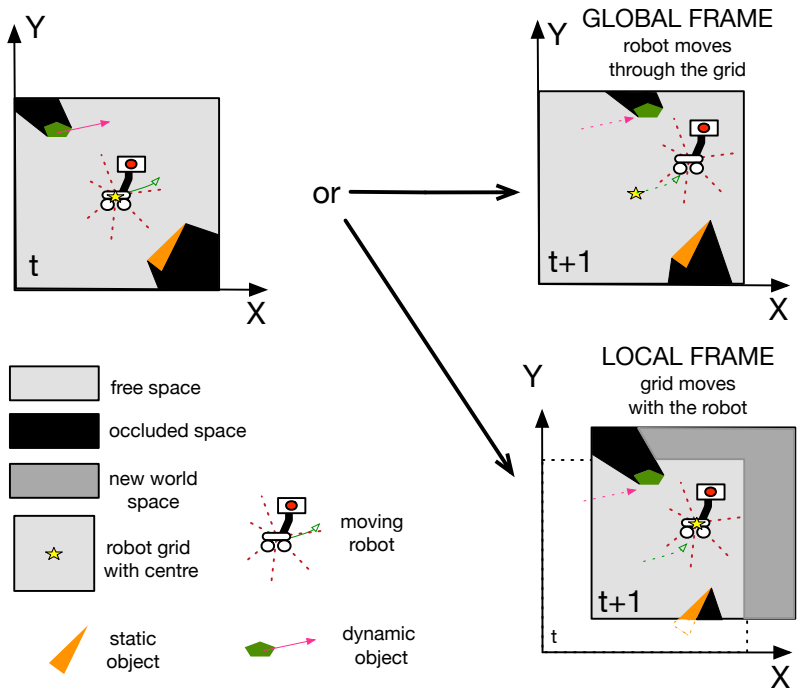


Figure 5.3: The network memory is forward transformed by the *sampler* using the estimated ego-motion transform $T_{t,t+1}$. The transform is converted into a spatial *sampling grid* which specifies where information from reference frame t needs to be copied into reference frame $t+1$.

ego-motion and the objects' dynamics. To decouple robot motion with object motion, we start by considering a static object with coordinates (i^t, j^t) in the grid at time t. If the robot moves with $T_{t,t+1}$ representing the transform from robot frame at time $t+1$ into t, in the incoming new sensor measurement $\mathbf{x}_{t+1}$, these coordinates will be displaced by the inverse transform:

$$\begin{pmatrix} i^{t+1} \\ j^{t+1} \end{pmatrix} = T_{t+1,t} \begin{pmatrix} i^t \\ j^t \\ 1 \end{pmatrix}, \tag{5.1}$$

where (i^{t+1}, j^{t+1}) represent the coordinates of the static object in the grid at time t. Similarly, in the new coordinate frame at time $t+1$, the previous memory at time t will have to be shifted by transform $T_{t+1,t}$ in order to remain coherent spatially with the new observation $\mathbf{x}_t$.

We choose this latter option, and adopt a two-step update of the network memory. We first ego-correct the network's memory by forward transforming the

hidden state according to $T_{t,t-1}$, and then predicting scene occupancy according to the network tracker as first illustrated in Figure 5.2.

In the following section we introduce our solution for updating the hidden state to incorporate vehicle ego-motion. We demonstrate as proof-of-concept on a synthetic dataset, that our extended network - unlike the original architecture - is able to handle ego-motion and effectively track moving objects through occlusion.

5.2.1 Problem Formulation

Our tracking objective remains that of predicting the future unoccluded state of the world $\mathbf{y}_{t+n}$ at time $t+n$, given a sequence of occluded observations of the world $\mathbf{x}_{1:t}$. Empty inputs $\mathbf{x}_{t+1:t+n}$ are provided to force the network to iteratively update and use its belief $\mathbf{h}_t$ to infer future state occupancy:

$$p(\mathbf{y}_{t+n}|\mathbf{x}_{1:t}) = p(\mathbf{y}_{t+n}|\mathbf{h}_{t+n}).$$

As we do not have access to the true unoccluded occupancy $\mathbf{y}_t$, we train the model in a self-supervised fashion and only calculate the loss function of the *observable* ground truth, which corresponds to future observable inputs. The binary cross-entropy loss is calculated and backpropagated only on the ground truth available, i.e the *visible* part of the space. This is achieved by simply masking the output prediction $\mathbf{y}_t$ with the visible input $\mathbf{x}_{t,vis}$ and multiplying the resulting grid element-wise with the occupancy part grid $\mathbf{x}_{t,occ}$.

We refer the reader to Section 2.2.5 for a detailed tracking problem formulation.

5.2.2 Network Architecture

In our application, we wish to forward transform the hidden state of our tracking network according to the robot ego- motion. To do so we draw inspiration from *Spatial Transformer Networks* [Jaderberg et al., 2015] which was originally introduced as a learnable module to explicitly allow the spatial manipulation of feature maps within a network. We refer the reader to Chapter 2.1 for a description of the module. The initial module comes in three parts. A *localisation net* first outputs

parameters of the transform to apply to the feature maps. A second *grid generator* then specifies how to sample the feature maps according to the desired transform, and a third part conducts the sampling to output the updated feature maps.

Though we could utilise the full capacity of the spatial transformer and *learn* the transformation to apply to the network's memory, we choose to leverage the fact that our tracker is embedded in a robotic system where ego-motion estimates are readily available. Our proposed adaptation is illustrated in Figure 5.4. We choose to bypass the localisation net, and directly consider ego-motion transform parameters θ provided by the visual odometry module to construct the sampling grid $\mathcal{T}_\theta(G)$.

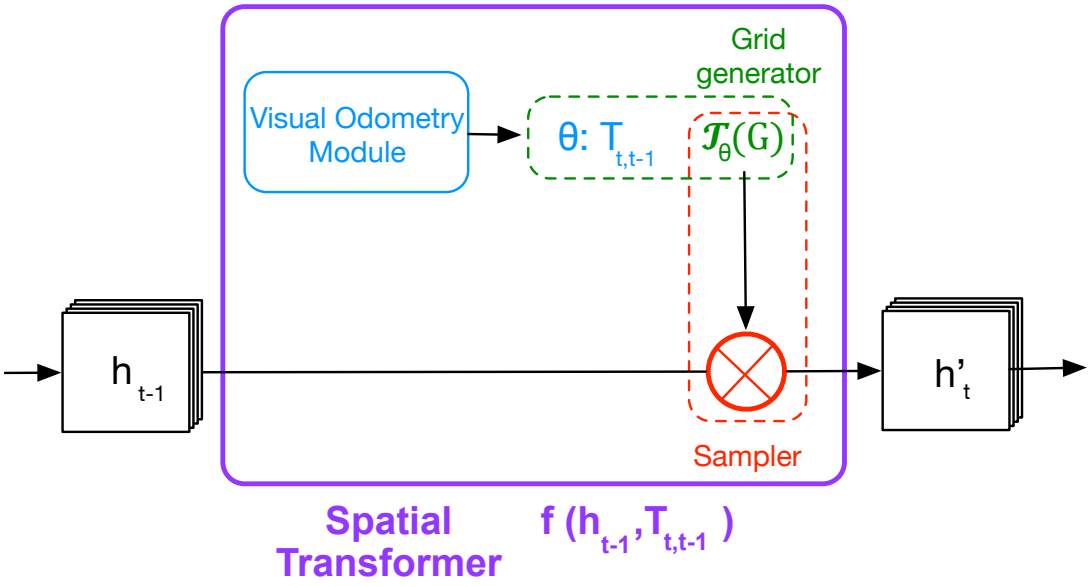


Figure 5.4: The hidden state update $f_1(h_{t-1}, T_{t,t-1})$ consists in a spatial transformation of the feature maps using ego-motion estimates $T_{t,t-1}$ provided by a visual odometry system. The transform parameters are used to produce a sampling grid $\mathcal{T}_\theta(G)$ which transforms the memory according to the robot's motion between $t-1$ and t.

The spatial transformer module inserts naturally in our baseline network, as illustrated in Figure 5.5.

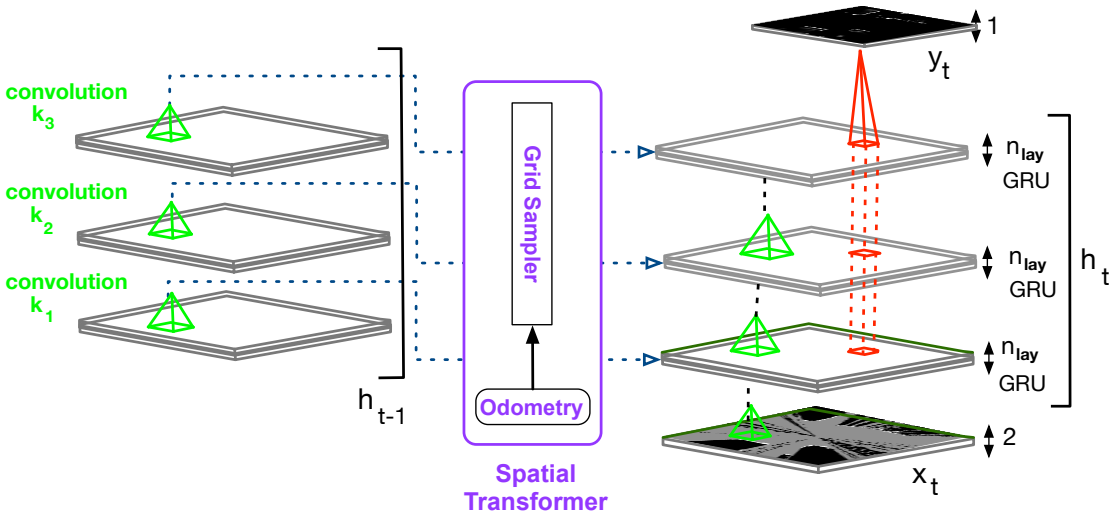


Figure 5.5: The architecture proposed for tracking from a moving vehicle. The spatial transformer module is introduced to update the hidden state before incorporating the new sensor measurement. The hidden state consists in three layers of n_{lay} GRU feature maps with either traditional or dilated convolutions of kernel size k_i, as explicited in the experimental section.

The ego-motion estimate is represented as an $\mathbb{SE}(2)$ transform and used directly to produce the *sampling grid*:

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = T_{t,t+1} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}, \tag{5.2}$$

where $T_{t,t+1}$ represents the 2D affine transformation from source frame at time $t+1$ to destination frame at time t. We choose a bilinear sampling kernel for the sampling grid. We note V_i^c to represent the value of the pixel located at coordinate (x_i^t, y_i^t) of the feature map c of target hidden state $\mathbf{h}_t^{'}$. Similarly, we note U_i^c to represent the pixel value located at coordinate (x_i^s, y_i^s) of the feature map c the input hidden state $\mathbf{h}_{t-1}$. The bilinear sampling kernel produces:

$$V_i^c = \sum_n^H \sum_m^W U_{nm}^c \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|). \tag{5.3}$$

This *sampling* kernel is applied to *all* feature maps c of the hidden state $\mathbf{h}_{t-1}$.

Backpropagating through the bilinear sampling kernel is easily achieved in the following way:

$$\frac{\partial V_i^c}{\partial U_{nm}^c} = \sum_n^H \sum_m^W \max(0, 1 - |x_i^s - m|)\max(0, 1 - |y_i^s - n|).$$

5.3 Datasets

We test the capacity of our extended network to track the world from a moving platform on three datasets. A first synthetic dataset features a moving sensor with static world objects to verify that the spatial transformer module is able to correctly update the network memory according to the robot ego-motion. A second synthetic dataset features a moving sensor and both static and dynamic world objects to verify that our updated architecture is able to decouple ego-motion from world dynamics, and accurately track objects into the future. We also produce semantic labels for the static and dynamic objects. Finally, we collect data from a moving vehicle in a busy urban environment to test our extended framework in a real-world setting. We regroup all three datasets in Table 5.1 and detail them in the following sections.

Dataset	Training Frames	Test Frames	Characteristics
Dataset 1	800	200	synthetic, static obstacles
Dataset 2	800	200	synthetic, static and dynamic obstacles
	800	200	static vs dynamic labels
Real World	Total: 39,560	Total: 9360	NISSAN LEAF, Oxford, UK: 28.7 km
Drive 1	14,640	4344	12.4 km (8.1 + 4.3)- 32 min
Drive 2	24,930	5016	16.3 km (11.3 + 5) - 50 min

Table 5.1: Characteristics of the three datasets considered, including number of training and test frames.

5.3.1 Synthetic

A sample of our generated dataset is illustrated in Figure 5.6. We build a world grid of size 45×45 and place the sensor in the centre (left of the figure). To avoid collision of objects with the sensor along its path, we restrict sensor motion to a 12×12 grid centred on $\{0, 0\}$, referred to as the *motion grid*, and drawn in green.

Objects are free to move outside of this motion grid only. Objects are either static or dynamic and represented with red circles.

At every given time, the sensor aims for a waypoint randomly located outside the motion grid (blue square). The motion of the sensor within the motion grid is controlled so as to minimise both angle and distance to the waypoint. The angle corresponds to that formed by the sensor's forward x coordinate and the segment linking waypoint and sensor coordinate frames. When the waypoint is first produced, the sensor starts moving towards it whilst rotating to point towards it. This means that the sensor can be moving forwards or backwards depending on its orientation. When the sensor is pointing towards the waypoint, it stops rotating and moves forwards. The angular velocity is proportional to the angular difference between sensor pose and waypoint reference frame, and linear motion is constant. When the sensor gets close to the motion grid edge, it turns back towards the centre, and a new waypoint is produced at a random location outside the motion grid. To add variation, a counter is incremented at every step and can produce a waypoint change, and there is a 5% chance that the sensor's motion

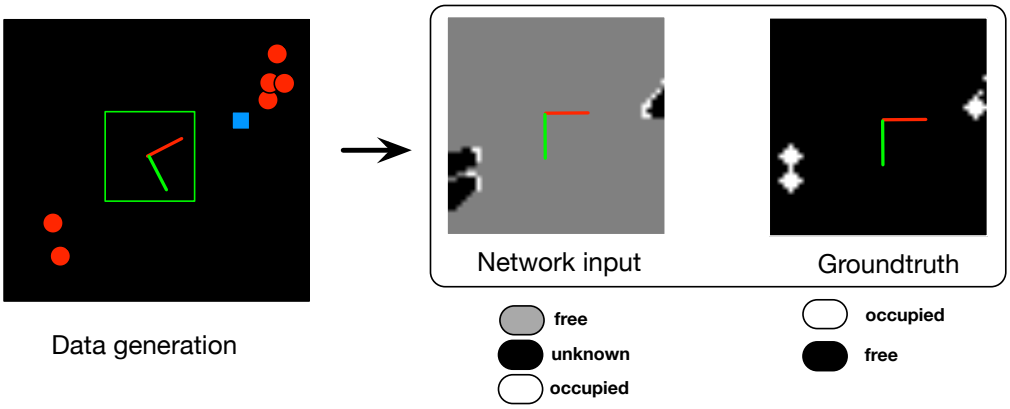


Figure 5.6: Illustration of our data generation setup. In our world view (left), objects are represented as red full circles, and the range sensor navigates freely within the central cyan square. The dark blue full square represents the next waypoint to guide the sensor's motion. The sensor frame follows the NASA convention with X forward (red) and Y to the right (green). The network input is obtained with raytracing (middle), and the corresponding ground truth occupancy is shown to the right.

be restricted to pure rotation during a sequence. All these choices were made empirically to produce dynamic and varied sensor motion.

At every timestep the robot moves through the world, the world occupancy is updated according to the object dynamics, and a laser scan is produced by raytracing from the objects to the sensor. The robot ego-motion between two timesteps is also recorded as an $\mathbb{SE}(2)$ pose transformation. An example with occluded input and corresponding ground truth occupancy grids is shown to the right of Figure 5.6.

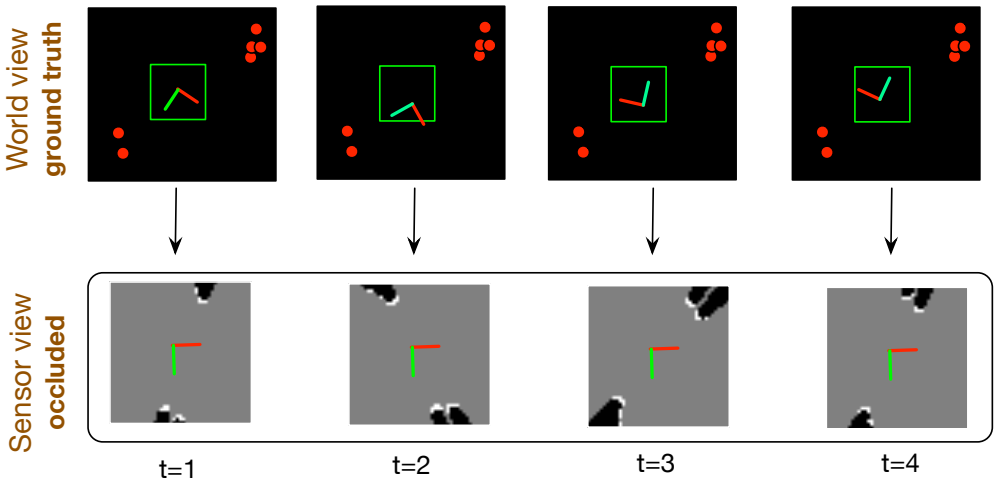


Figure 5.7: Short sequence to illustrate our synthetic simulation on a dataset containing only static objects. The world view (top) is projected into the sensor frame as a partially occluded occupancy grid (bottom). Objects are static (red circles) and the sensor moves with random motion.

We construct two datasets of 1000 sequences of length $N = 20$ which we split 80/20 between training and test sets:

- **Dataset 1**: consists of sequences where all world objects are static, and only the sensor is permitted to move. This is to test the network's ability to handle ego-motion without the additional challenge of moving obstacles

- **Dataset 2**: consists of sequences where the sensor moves, and world objects are either static or dynamic. Dynamic objects move in linear patterns, along paths that cannot intersect with the motion grid. Both static and dynamic objects have the same radius and appearance in the laser data. The training

set contains 1.5 times the amount of dynamic obstacles vs. static obstacles. This was empirically chosen so as to produce more dynamics objects which are of particular interest for testing the network's ability to track moving objects, whilst not cluttering the world grid.

Figure 5.7 illustrates a short sequence of simulation where the sensor moves within its grid and observes static objects.

5.3.2 Real World

To test our proposed solution in a real world environment we drove around the busy centre of Oxford, UK using a NISSAN LEAF. Figure 5.8 shows the vehicle equipped with two Velodyne HDL64E spinning lasers located on either side of the roof, which are combined to produce a 360 degree field of view around the vehicle. A Bumblebee XB3 stereo camera is also mounted to the front of vehicle facing forward, providing continuous feed to a visual odometry system to produce ego-motion estimates.

Figure 5.8: We collect data using a NISSAN LEAF vehicle equipped with two Velodyne HDL64E lasers located on either side of the roof (green), as well as a stereo Bumblebee3 camera facing forward (orange). The lasers provide a 360 degree view of the environment around the vehicle, and the stereo images are fed to a visual odometry system to produce ego-motion estimates. We show a selection of forward camera views from the Oxford route driven.

For consistence with the Oxford Robotcar Dataset [Maddern et al., 2017] which covers an interesting busy urban route, we collected an 83 minute long dataset in two outings, following the Oxford 10K route. The first drive consists in 32 minutes of driving, split into 8.1 km for training and 4.3 km for testing. The second drive consists in 50 minutes of driving, split into 11.3 km for training and 5 km for testing. The data split was conducted so as to have no geographical overlap between training and test sets, to obtain a 80/20 split in number of frames, and to include straight lines and turns. The total number of training and test frames resulted in $D_{train} = 39560$ and $D_{test} = 9360$ as regrouped in Table 5.1. The frame rate corresponds to about $10Hz$ and the geographical layout of the training and test sets are shown in Figure 5.9.

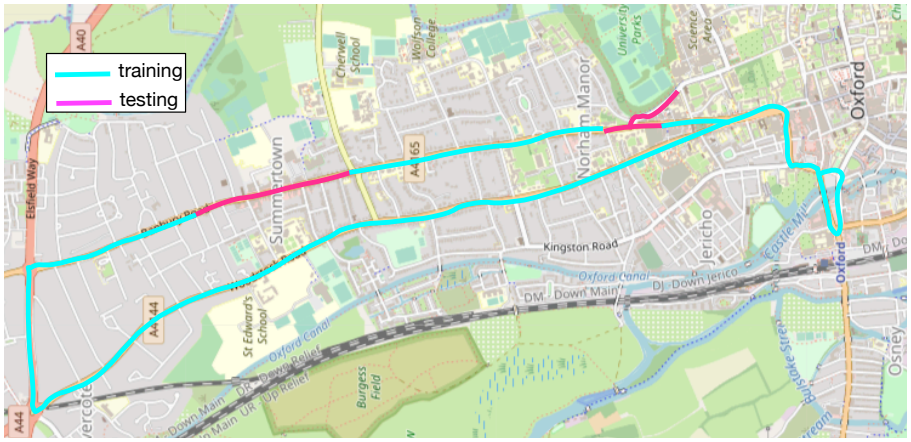


Figure 5.9: Dataset collected in the urban centre of Oxford, UK from a NISSAN LEAF vehicle equipped with 3D lasers and a stereo camera. The training and testing sets do not overlap geographically.

To adapt our collected data to our tracking architecture, we reduce the 3D point clouds to 2D scans by considering the range of points within a height of 0.6-1.5m from the ground (Figure 5.10). The resulting points are binned to produce a 360 degree planar laser scan of 1441 returns in meters, corresponding to an angular precision of $\theta = 0.25°$. The network input grid is centered on the robot, and covers an $W \times H = 28\text{m} \times 18\text{m}$ area with a grid spacing of 0.2m.

To obtain estimates of ego-motion we feed the stereo images to a visual odometry module which produces $\mathbb{SE}(3)$ transforms between frames. We query a relative pose

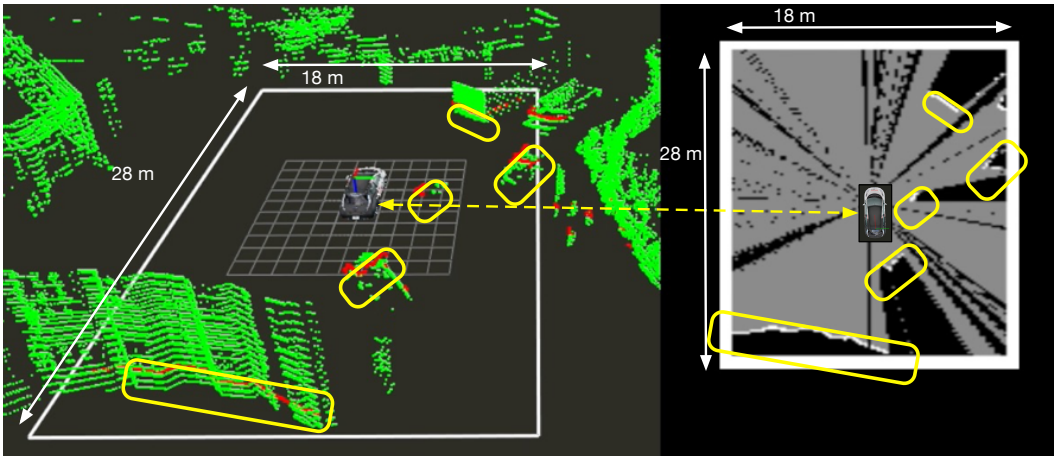


Figure 5.10: Conversion of 3D point cloud data into a 2D occupancy and visibility grid as input to the network. The points considered in the original point cloud range within 0.6-1.5 meters vertical from the ground plane local to the robot's location.

client to obtain transforms between laser timestamps, and express the ego-motion estimates in the laser frame:

$$T_{t-1,t}^{laser} = T_{laser,camera} \times T_{t-1,t}^{camera} \times T_{camera,laser}.$$

To produce our sampling grid in the horizontal plane, we convert the resulting $\mathbb{SE}(3)$ transforms to $\mathbb{SE}(2)$ by selecting the $[X, Y, \theta]$ planar components of the vehicle frame, where the z axis in the $\mathbb{SE}(3)$ transform corresponds to the component locally vertical to the ground. As this is performed at every timestamp, we find that the approximation involved introduces negligible error in the sequences considered.

Finally we split the data into $D_{train} = 989$ and $D_{test} = 232$ training and test sequences of length $N = 40$.

5.4 Experimental Evaluation

In the following section, we present and discuss a number of experiments conducted to address the key questions with respect to our proposed solution: can the original architecture track a dynamic world from a moving sensor? can the proposed solution account for robot ego-motion? can the proposed solution decouple ego-motion from world object motion? does the proposed solution aid tracking in a real-world setting? We also explicit the training methodology followed across experiments.

5.4.1 Methodology

In this section we detail our input data format, training procedure, and outline the different models compared in the results section.

Input data Training consists in dropping out inputs regularly within each sequence, to force the network to learn to predict into the future. For synthetic datasets, we choose $N = 20$, i.e we show the network the first 10 frames, and drop out the next 10 frames. This prediction length of 10 timesteps was empirically chosen to be long enough to show that correctly predicting into the future requires using the network memory. For the real world dataset, we consider sequence lengths of $N = 40$, and drop out every other 5 frames. We found this was a reasonable length to consider in view of the car velocity and position of the robot at the centre of the network grid. With a mean velocity of 20mph the vehicle on average travels 4 meters during 5 frames of prediction. We find this is enough travelling length to reflect the effect of natural occlusions whilst avoiding growing too large of a new unknown region ahead of the robot.

Training schedule At every epoch of training, the training set indices are randomly shuffled and the model is trained on single sequences. All models are trained using stochastic gradient descent using the Adagrad optimizer with an initial learning rate of 0.01. Training is conducted until convergence on the training set though we perform early stopping if we observe overfitting on the test set. For each experiment, we conduct five runs of each model. For the static datasets, runs are conducted until completion, i.e early stopping to avoid overfitting and the best run chosen. For the real world dataset, as convergence is achieved after ten days of training on a Titan Tesla P100, we choose to run five iterations of the model until epoch $k = 90$. We select the model with best performance on the training set, and run until convergence. We report details of the runs for each experiment.

Evaluation We evaluate our models by comparing the mean μ and one standard deviation from the mean σ of the F1 performance $\{s_i\}$ calculated on the D_{test} sequences of the *test* set:

$$\mu = \frac{1}{D_{test}} \sum_{i=1}^{D_{test}} (s_i) \tag{5.4}$$

$$\sigma^2 = \frac{1}{D_{test} - 1} \sum_{i=1}^{D_{test}} (s_i - \mu)^2. \tag{5.5}$$

When comparing the series of runs up to k epochs we compare the training loss averaged over the D_{train} sequences to choose the best model.

Comparing different architectures' F1 performance allows comparison of their relative ability to predict scene occupancy. As we are particularly interested in the task of tracking under robot motion, we additionally find it meaningful to study the distribution of performance around this mean. In synthetic contexts, this aims to verify that the proposed solution can effectively account for robot motion regardless of sensor yaw and forward translation (low variance). In a real world context, we find that the F1 performance is dominated by benign robot motion due to the prevalence of straight roads and constant velocity. To verify the efficacy of the proposed solution in this context, we analyse the distribution of performance as a function of robot forward motion and yaw, and compare to the baseline model.

Models All experiments aim to verify that the proposed solution is able to decouple sensor ego-motion and world object dynamics. We therefore consider the following models:

- *model*_VO: a model which considers ego-motion via use of the spatial transformer

- *model*_no_VO: a model which *does not* take into consideration ego-motion. the belief state update is in this case similar to the static sensor case.

The suffix "VO" here refers to Visual Odometry which is the system used to provide estimates of robot ego-motion. Details of the visual odometry module used can be found in Section 2.3.2.

We clarify the architectures considered and compared within each experiment. Our code is implemented in Torch, with a GPU implementation of Spatial Transformers[1].

5.4.2 Handling sensor ego-motion in a static world

We first seek to answer the following question: can the proposed spatial transformer solution handle sensor ego-motion compared to the baseline architecture? We consider synthetic **Dataset 1** consisting in sequences where the sensor is *moving* but where world objects are *static* (Figure 5.11). The data is produced according to Section 5.3.1 and split in $D_{train} = 800$ and $D_{test} = 200$ sequences for the training and test sets.

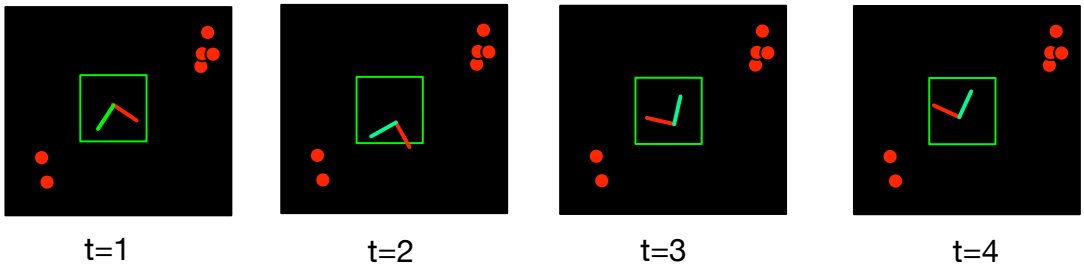


Figure 5.11: Example sequence from **Dataset 1** featuring a moving sensor and static objects (red circles) as seen from the world view. The sensor represented by its reference frame moves within the limits of the green square.

We choose a common base architecture consisting in a three-layer hidden state of 16 GRU units each (GRU3), with convolutional kernel sizes of 7×7 and where the last hidden layer is decoded to the output with kernel size 7×7. We found this architecture was sufficient in terms of capacity and receptive field.

We compare GRU3_no_VO and GRU3_VO which have the common architecture presented above and only differ by their incorporation or not of the spatial

[1] `https://github.com/qassemoquab/stnbhwd/`

transformer with ego-motion information. The spatial transformer for GRU3_VO is introduced as illustrated in Figure 5.5.

Quantitative analysis We compare both models' performance in Figure 5.12. Model runs with ego-motion (GRU3_VO) are represented in blue, model runs without ego-motion (GRU3_no_VO) in red. Both architectures are trained five times with early stopping. The top left table compares end of training epochs and costs for all runs. Circled in blue and red are the best runs for GRU3_VO and GRU3_no_VO respectively. Notice an order of magnitude lower cost for the informed model runs. Bottom left plot compares the corresponding mean F1 score of all runs at end of training from timesteps $t = 10 - 20$, with $t = 11 - 20$ representing full prediction as the input is dropped. The plot to the right highlights the mean and one standard deviation from the mean of the F1 score on the test set for the two best runs.

As expected we observe a clear difference in F1 performance between the two series of models. GRU3_no_VO (red) fails at predicting into the future as observed by the sharp drop in F1 measure from 1 to close to zero for all five runs as soon as the input is dropped out ($t = 11 - 20$). In contrast, GRU3_VO which takes into account sensor ego-motion (blue) achieves near perfect prediction into the future as shown by the F1 measure which remains close to 1 even when the input is dropped. For both models, all five runs show commensurate performance. The two best models show little variance between sequences which suggests robust ability (or failure) to predict into the future.

Qualitative analysis We qualitatively compare the two best models' ability to predict into the future. Figure 5.13 represents from left to right two typical sequences from the test set.

The input and output occupancy grids are always viewed from a sensor-centric viewpoint and we represent every other timestep between $t = 8$ and $t = 20$. For both sequences, the top row shows the forward translation Δx and yaw $\Delta\theta$ components of the sensor $\mathbb{SE}(2)$ ego-motion transform $T_{t,t+1}$ to illustrate sensor motion throughout

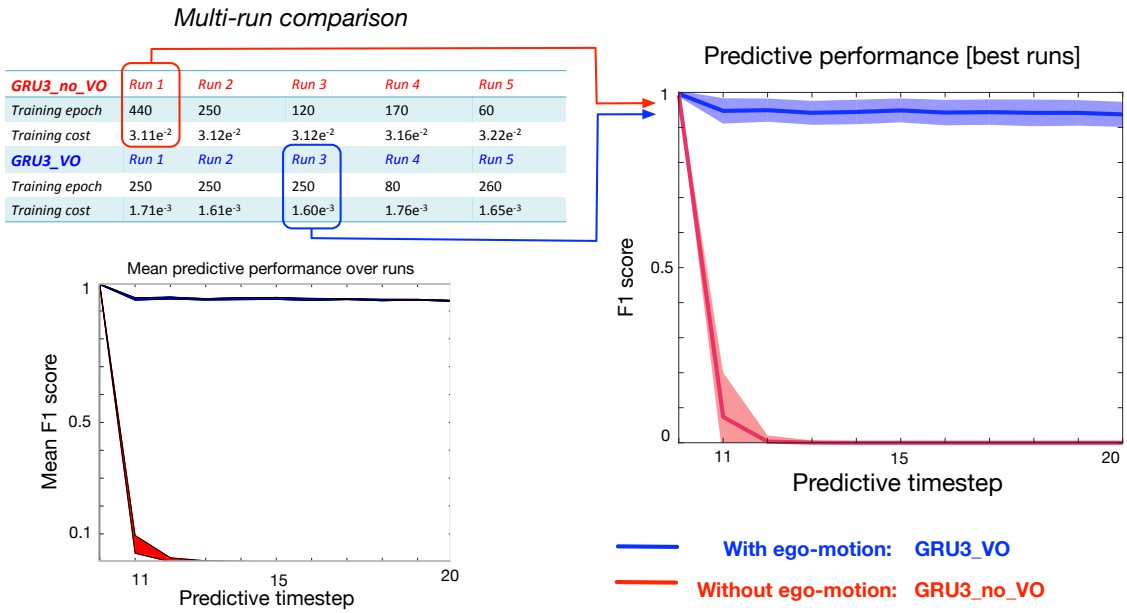


Figure 5.12: Comparison of model performance on **Dataset 1**. Blue: GRU3_VO (ego-motion), Red: GRU3_no_VO (no ego-motion). *Left column*: model performance over five runs with different weight initialisations. Top: training epoch and cost at end of training for each run. The two best models are circled in red and blue. The training cost is one order of magnitude lower for GRU3_VO compared to the baseline models GRU3_no_VO. Bottom: mean F1 score on the D_{test} sequences of the test set for all five runs of both models at end of training. Timesteps $t = 11 - 20$ represent full prediction (input dropped out). The shaded regions represent the minimum/maximum span of mean F1 performance over the five runs of both models. *Right Column*: Predictive performance (F1 mean and one standard deviation) of the best fully trained models. Only the models which are aware of sensor ego-motion (GRU3_VO) are able to correctly predict future occupancy states (F1 nearing a value of 1). Best viewed on pdf.

the sequence. Second row from the top shows the input, which is shown to the network until timestep $t = 10$, and then dropped out. The two bottom rows show the output predictions from the two models, color-coded according to the target output as false positive (blue), true positive (green), and false negative (red). The output predictions are thresholded at 0.5 to calculate these metrics.

The network with ego-motion predicts perfectly despite numerous changes in orientation and direction. It has also learned to fill the circular objects (false positives) as it is the simplest representation it can learn to make sense of the sequence of observations. On the other hand, the network without ego-motion fails to predict the sequences entirely.

These findings suggest that spatially transforming the network memory according

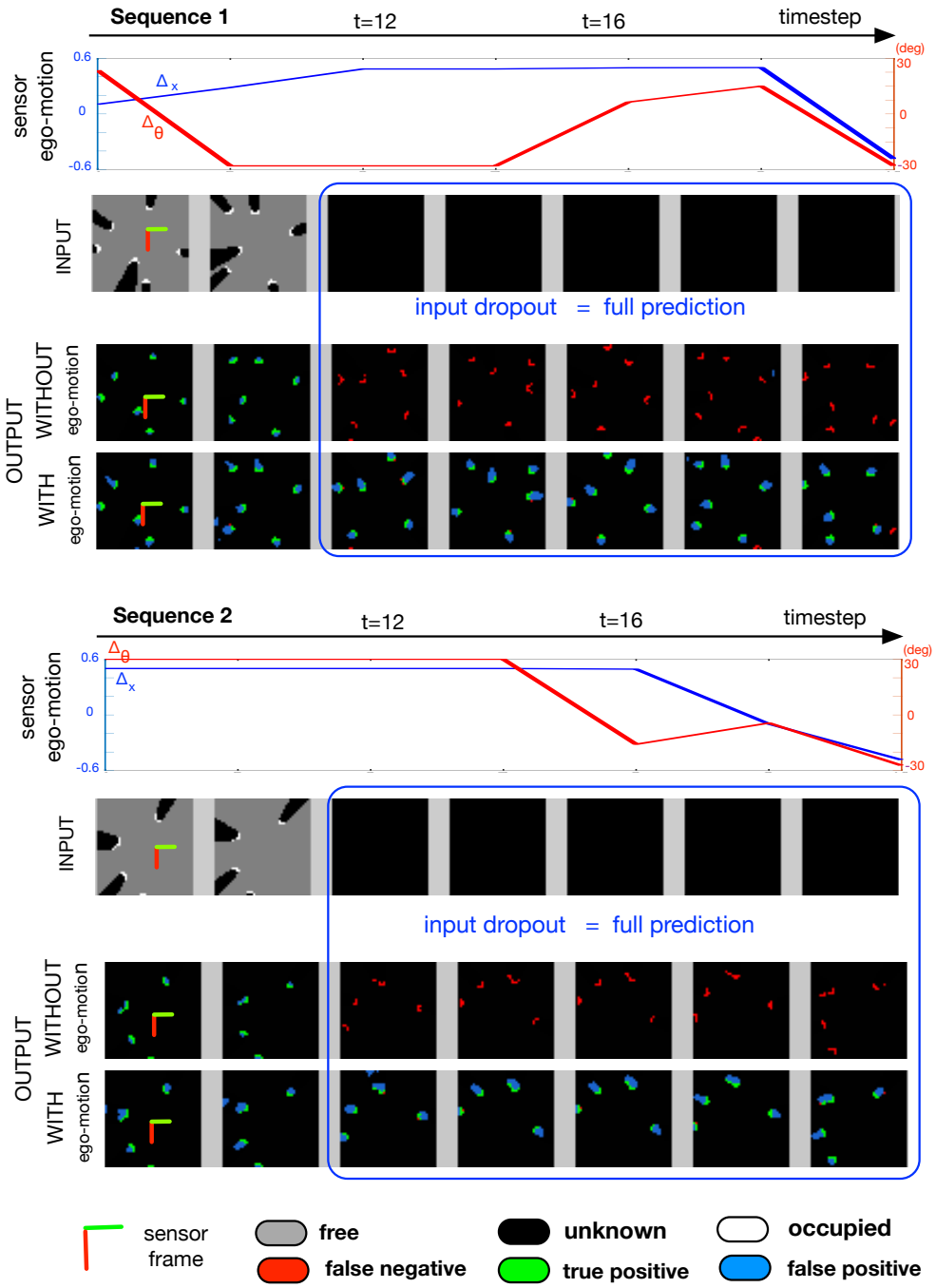


Figure 5.13: Model prediction comparison from $t = 8 - 20$ on typical sequences of Dataset 1 featuring a moving sensor and static obstacles. For both sequences, top to bottom rows: 1) forward translation Δx and yaw $\Delta\theta$ components of the sensor ego-motion; 2) model input, dropped out from $t = 11 - 20$; 3) model prediction without ego-motion (GRU3_no_VO); 4) model prediction with ego-motion (GRU3_VO). The network which incorporates ego-motion predicts flawlessly as indicated by the lack of false negatives. It has additionally learned to predict full circular objects. The model without ego-motion completely fails in the task. Best viewed on pdf.

to the sensor ego-motion is a necessary and effective way to handle sensor motion in a static world.

5.4.3 Handling sensor ego-motion in a moving world

We now seek to answer the following question: can this same spatial transformer mechanism additionally handle decoupling sensor ego-motion from world object motion? We consider synthetic **Dataset 2** consisting in sequences where the sensor is *moving* and world objects are either *static or moving* (Figure 5.14). The data is produced according to Section 5.3.1 and split in $D_{train} = 800$ and $D_{test} = 200$ sequences for training and test sets.

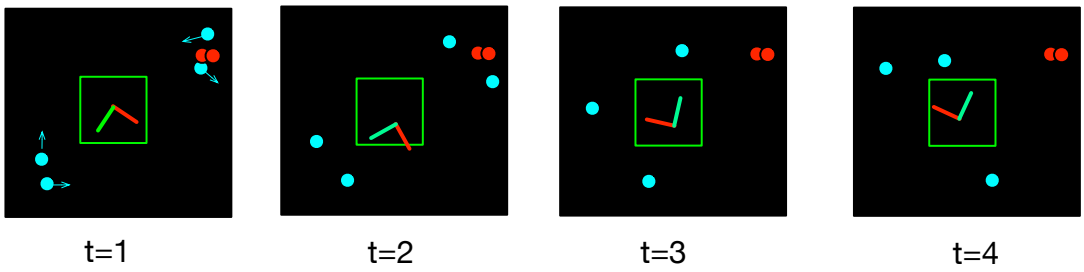


Figure 5.14: Example sequence from **Dataset 2** featuring a moving sensor and both static and moving objects (red circles) as seen from the world view. The sensor represented by its reference frame moves within the limits of the green square.

We choose a similar baseline architecture as in the previous experiment: a three-layer hidden state of 16 GRU units each (GRU3), with convolutional kernel sizes of 7×7 and where the last hidden layer is decoded to the output with kernel size 7×7.

We consider the two same architectures as previously which only differ by their use or not of the spatial transformer.

Quantitative analysis We compare both models' performance in Figure 5.15. Similarly to the previous experiment, model runs with ego-motion (GRU3_VO) are represented in blue, model runs without ego-motion (GRU3_no_VO) in red. Both architectures are trained five times with different weight initialisations and with early stopping to prevent overfitting on the training set. The top left table compares the different runs with the best runs encircled in blue and red GRU3_VO

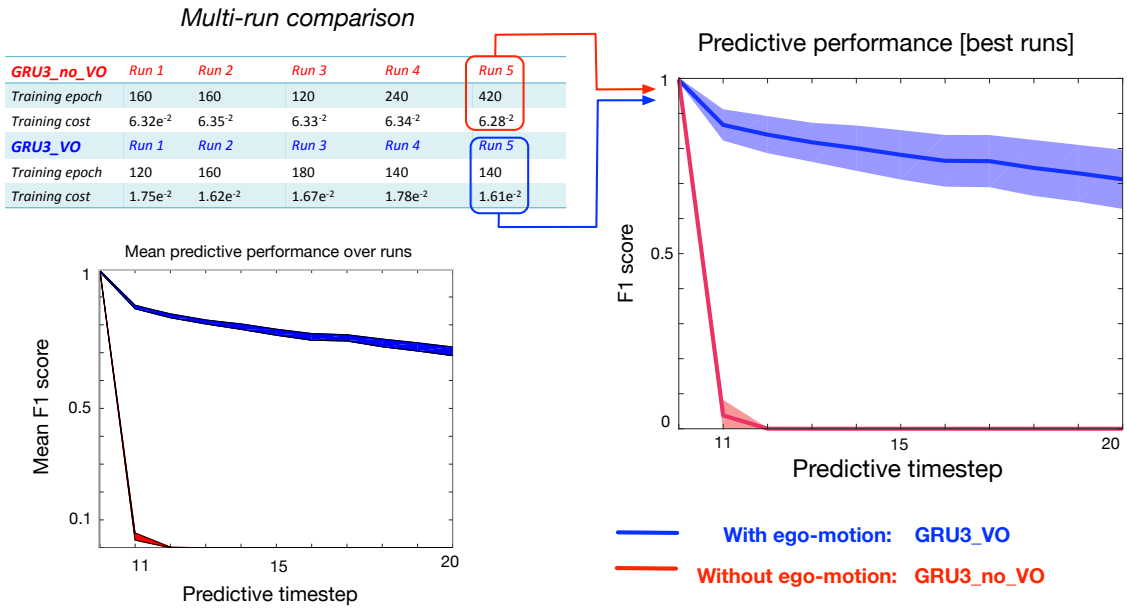


Figure 5.15: Comparison of model performance on **Dataset 2**. Blue: GRU3_VO (ego-motion), Red: GRU3_no_VO (no ego-motion). *Left column*: model performance over five runs with different weight initialisations. Top: training epoch and cost at end of training for each run. The two best models are circled in red and blue. Bottom: mean F1 score on the D_{test} sequences of the test set for all five runs of both models at end of training. Timesteps $t = 11 - 20$ represent full prediction (input dropped out). The shaded region represents the minimum/maximum span of mean F1 performance across runs. *Right column*: predictive performance (F1 mean and one standard deviation) of the best fully trained models. Only the models which are aware of sensor ego-motion (GRU3_VO) are able to correctly predict future occupancy. Best viewed on pdf.

and GRU3_no_VO respectively. Bottom left plots compare the mean F1 score between all runs, and the right plot highlights the predictive mean and standard deviation for the two best models.

We once again find a sharp contrast between the baseline model and GRU3_VO which shows ability to predict future occupancy despite moving objects and moving sensor. As there are moving obstacles entering the grid at unpredictable times it is not possible for the model to predict into the future without some loss of performance compared to the previous experiment. In contrast, the baseline model completely fails to predict future states.

Qualitative analysis We qualitatively compare the performance of the two best models on two sequences from the test set shown in Figure 5.16. Similarly to the

previous experiment, we represent the grids centered on the robot frame and show every other timestep between $t = 8 - 20$. Timesteps $t = 12 - 20$ represent full prediction as the input is dropped out. For both sequences the top row shows the forward translation Δx and yaw $\Delta\theta$ components of the sensor $\mathbb{SE}(2)$ ego-motion transform $T_{t,t+1}$ throughout the sequence.

Second row from the top shows the input, which is shown to the network until timestep $t = 10$, and then dropped out. The two middle rows show the output predictions from the two models, color-coded according to the target output as false positive (blue), true positive (green), and false negative (red). The output predictions are thresholded at 0.5 to calculate this metrics. We additionally add a bottom row showing the semantic ground truth occupancy to get a sense of which objects are dynamic (purple) and which are static (light green). This is for illustrative purposes only; semantic information is not provided to any of the models.

Despite constant motion and rotation from the sensor, and the presence of dynamic objects, the model which incorporates ego-motion predicts perfectly the sequence. The baseline model completely fails in the task. We find that the spatial transformer is an efficient solution for tracking by decoupling sensor ego-motion and world dynamics.

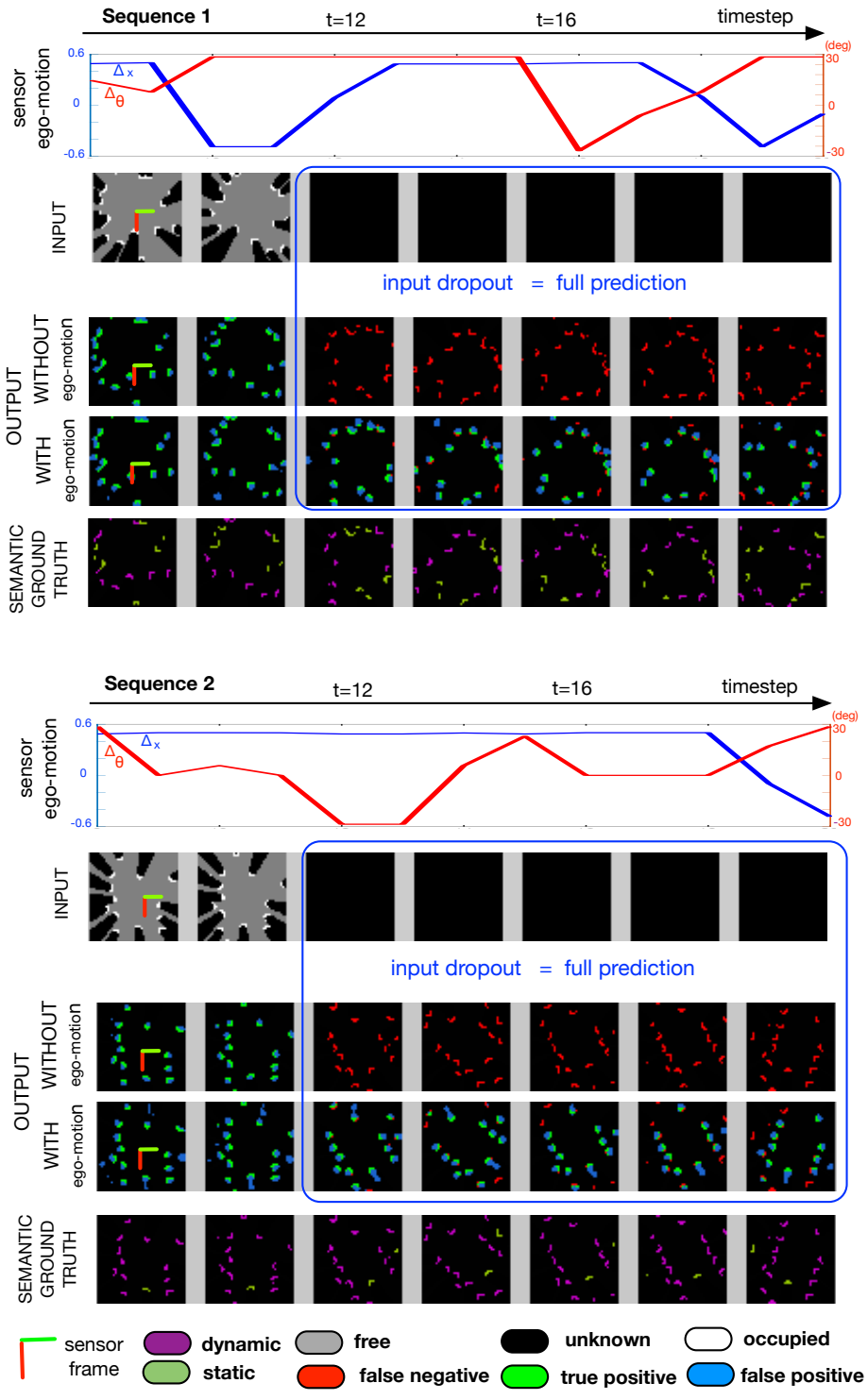


Figure 5.16: Model prediction comparison from $t = 8 - 20$ on typical sequences of **Dataset 2** featuring a moving sensor, and both static and dynamic objects. For both sequences, top to bottomw rows represent: 1) forward translation Δx and yaw $\Delta\theta$ components of the sensor ego-motion; 2) model input, dropped out from $t = 11 - 20$; 3+4) model prediction without and with ego-motion (GRU3_no_VO vs. GRU3_VO); 5) scene semantics for visualisation purposes. Only the network which incorporates ego-motion is able to predict the future scene occupancy. Best viewed on pdf.

5.4.4 Tracking in the real world

In our two previous experiments on synthetic data, we found a clear signal that without incorporating ego-motion the baseline model is not able to track objects from a moving sensor, particularly in the presence of yaw. We also showed that incorporating ego-motion via a spatial transformer module was an efficient way of handling both sensor motion and dynamic objects.

In this third experiment, we evaluate our proposed solution on a vehicle driving in a real-world urban environment. We compare a baseline architecture which does not incorporate ego-motion (GRU3_no_VO) with an equivalent model incorporating the Spatial Transformer module into the hidden belief state (GRU3_VO). We refer to this later model as the *informed* model in the text. We choose one of the best architectures found in our real world experiments of Chapter 4 as a base architecture for our models. It features three layers of 16 gated recurrent units, dilated convolutions, and decodes all the hidden state to the output.

A one-second long forward-prediction takes 13ms on a Titan GeForce GTX NVIDIA GPU with 6GB of RAM, making it applicable to real-time tracking.

Quantitative analysis We compare five runs of both models up to training epoch $k = 90$ in Figure 5.17, left. Following our synthetic analysis, unsurprisingly, the GRU3_VO models offer improvement over the baseline model in all future frames as indicated by the F1 mean. All model runs achieve commensurate performance for each model, with lower cost when incorporating ego-motion information.

We fully train the two best models to epoch 1200 and observe the mean and standard deviation of the F1 measure in Figure 5.17, right. Both models show decreased mean performance and increased variance with increasing time horizon. This is consistent with increasing uncertainty as the belief is propagated and as new parts of the world become observed. The model considering ego-motion achieves higher mean with a lower standard deviation which suggests a positive contribution of the spatial transformer.

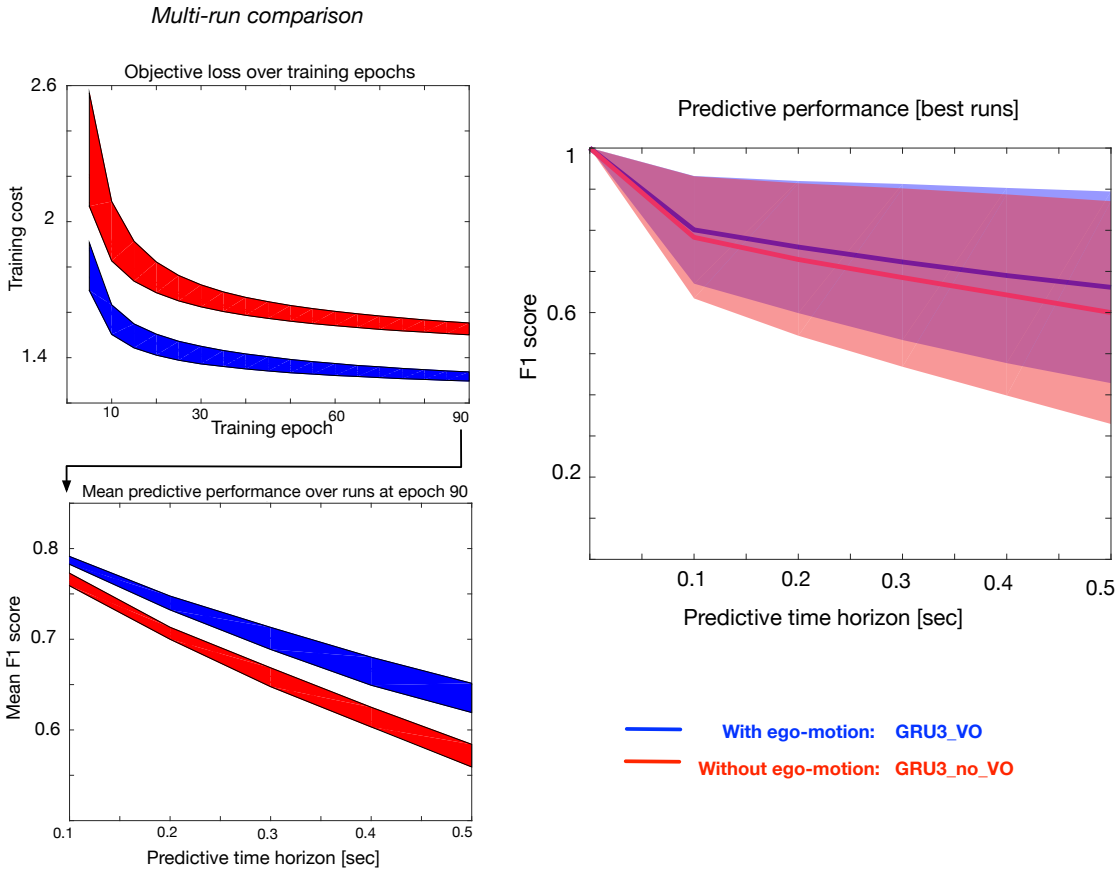


Figure 5.17: Model performance with (blue) and without (red) ego-motion. Left: Comparison of five runs with different weight initialisations. The shaded regions correspond to the minimum/maximum values over the five runs for both models. The top plot shows decreasing training set cost with number of training epochs for all model runs. The bottom plot shows the mean F1 measure for all runs at training epoch $k = 90$ for a predictive time horizon of $t = 0.5$ seconds. We observe favourable performance for GRU3_VO on both metrics. Right: Mean F1 score over the predictive time horizon of $t = 0.5$ sec for the two best fully trained models. The shaded regions represent one standard deviation from the mean. We observe a positive contribution of the spatial transformer both in terms of increased mean and reduced variance. The baseline model does surprisingly well which we attribute to the benign test set where the vehicle mostly evolves at constant velocity down straight roads. We illustrate this hypothesis in Figure 5.23 and investigate the data spread in the text.

The spread of performance (standard deviation) is larger for the model without ego-motion particularly at larger time horizons which is expected. However, surprisingly, it achieves respectable performance. We investigate why this may be by observing prediction performance as a function of vehicle velocity represented in Figure 5.18.

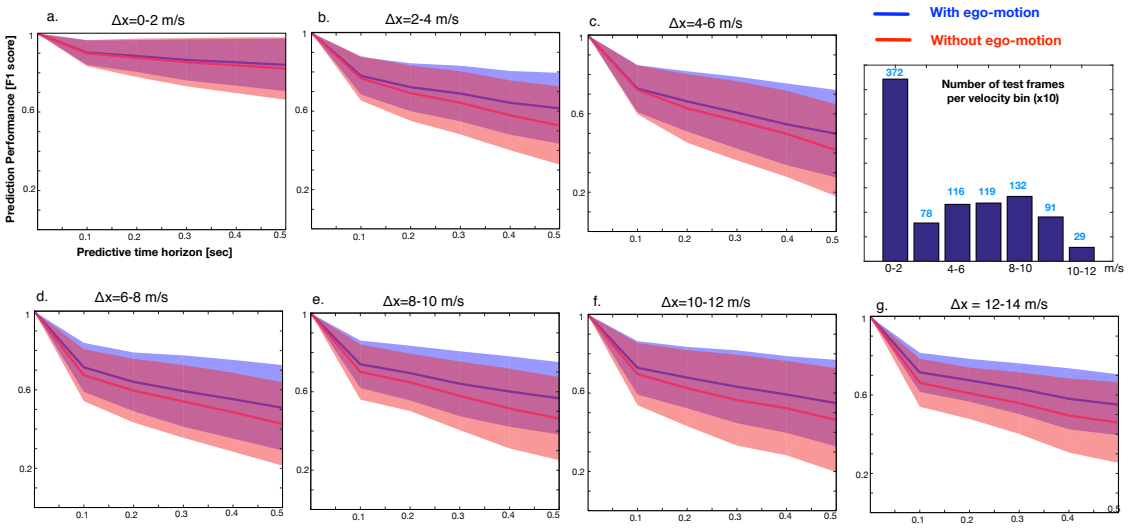


Figure 5.18: Predictive ability of both models as a function of vehicle forward velocity Δx in meters per second. Curves represent mean, and shaded areas one standard deviation from the mean. Figures *a*-*g* represent different velocity bins of width 2m.s^{-1}, ranging from $a : \Delta x = 0 - 2\text{m.s}^{-1}$ to $g : \Delta x = 12 - 14\text{m.s}^{-1}$. The plot in the upper right hand corner represents the number of test frames in each bin. Both models achieve commensurate very high tracking performance when the vehicle speed is close to zero (a). As the velocity of the vehicle increases, the performance of both models decreases, and the performance gap between baseline and informed model increases; the baseline model (red) shows worsening mean and variance compared to the model with ego-motion information (blue). We notice that the largest bin in terms of number of test frames corresponds to 40% of the test set and when the vehicle is travelling at speeds lower than 2m.s^{-1}. This weighs favourably for the baseline model's overall performance.

We present seven bins of increasing velocity from $a : \Delta x = 0 - 2\text{m.s}^{-1}$ top left, to $g : \Delta x = 12 - 14\text{m.s}^{-1}$ bottom right. We consider the data in sequences of length $t = 1 : 10$ corresponding to a full sequence of 5 inputs shown and 5 inputs dropped. For each of these ten timesteps, we bin the data by velocity and compute mean and standard deviation for both models. The number of test samples corresponding to each bin is shown in the top right bar plot. We overall notice that the baseline model achieves worse performance in terms of mean and standard deviation across velocities. Both models achieve very high performance when the vehicle is travelling at less than 2m.s^{-1} suggesting that they have both learned to track from a static platform. Both models show decreasing performance as the velocity of the vehicle increases. However, we notice that the difference in performance between both models increases; that is the mean and standard deviation of the baseline model

degrade further than the model incorporating ego-motion. This reflects a positive contribution of the spatial transformer. We also point out the fact that the first bin *a.* corresponding to speeds between $0 - 2\text{m.s}^{-1}$ carries 40% of the total test frames. This is due to the fact that the dataset presents heavy traffic and red lights which contributes favourably to the overall performance of the baseline model.

We similarly observe model prediction as a function of vehicle yaw, $\Delta\theta$ in Figure 5.19. Similarly to the Figure 5.18, we bin the test frames with respect to angular velocity for every timestep $t = 1 : 10$ of a full on/off sequence. We observe the models' predictive ability within a range of ± 25 deg.s^{-1} and show the number of test frames per bin in the bottom right bar plot. Similarly to the previous graph, we observe a positive contribution of the informed model at all values of yaw in terms of mean. Both models show best performance when the vehicle is travelling in straight lines (bin *c.*). This is reasonable as this corresponds to constant velocity with benign appearance change compared to rotation, which can be learned by the baseline model. Medium angular velocities of $\pm[5 - 15]$ deg.s^{-1} (bins *b.* and *d.*) represent the most striking performance difference between both models.

Finally the two extreme bins reflect commensurate performance of both models suggesting that at these high turn radii other challenges such as new unobserved regions, and angle viewpoint may come into play.

Notice however that the central bin where the vehicle is driving along straight lines contains more than 90% of the data, making it difficult to draw conclusions on the performance difference for the smallest bins of higher yaw.

Observing F1 performance as a function of both yaw and velocity similarly suffers from little data at both high yaw and velocity bins. Nonetheless, across velocity and yaw we find evidence that the spatial transformer contributes favourably to the model's performance.

Qualitative Results As the frequency and duration of natural occlusions in the dataset is not as high as in the static vehicle dataset, we maintain the sequence of 5 on/off and observe the full prediction of the model when the input is dropped out. We observe a selection of sequences where the informed model performs well,

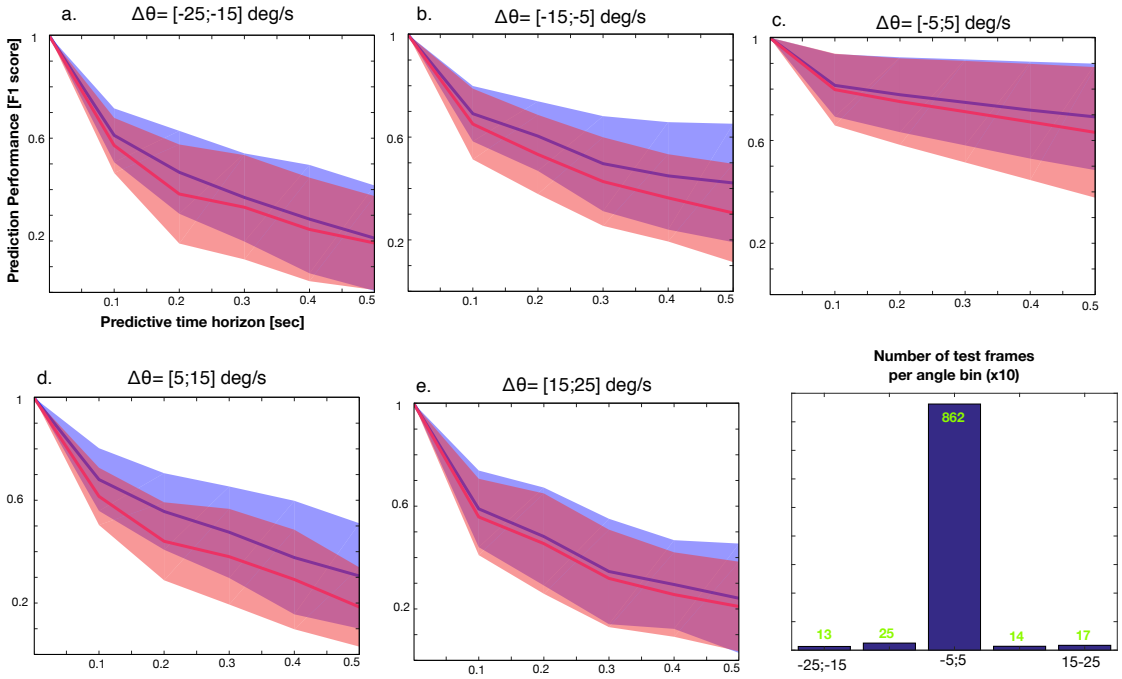


Figure 5.19: Predictive ability of both models as a function of the vehicle yaw $\Delta\theta$ represented in degrees per second. Curves represent mean and shaded areas one standard deviation from the mean. Figures a.-g. represent bins of increasing velocity, with width 10 deg.s^{-1}, ranging from $a : \Delta\theta = [-25; -25]$ to $e : \Delta\theta = [15; 25]$. The plot in the lower right hand corner represents the number of test frames in each bin. The informed model shows highest performance across the range of vehicle yaw. Both models achieve best performance when the vehicle is driving in near straight lines (bin *c.*). The fact that this bin contains more than 90% of the test data makes it difficult to draw general behaviour patterns between both models for the other bins.

and others where it does more poorly. All sequence images are shown from a sensor centric point of view, as indicated by the robot frame at the center of the grids, where X represents the forward facing coordinate.

Tracking when the vehicle is stopped We first observe the model's ability to track when waiting at a red light in Figure 5.20. The sequence of inputs spans 0.6 sec of data, and the figure is to be viewed from bottom to top. The network output (line 2) is registered against the target occupancy (line 3) and projected back into the 3D pointcloud (line 4). The top line 5 shows the forward facing camera view for context. To register the view with the point cloud we circle the static car with a cyan contour in lines 4 and 5 at timestep $t = 0$. Inputs $t = 1 - 5$ are dropped out, resulting in full prediction by the network. We notice that the network is able

to remember the location of static obstacles, as well as correctly track two moving cars as highlighted by the yellow contours. The model has also learned to fill the partially visible cars which is a useful by-product for estimating scene occupancy.

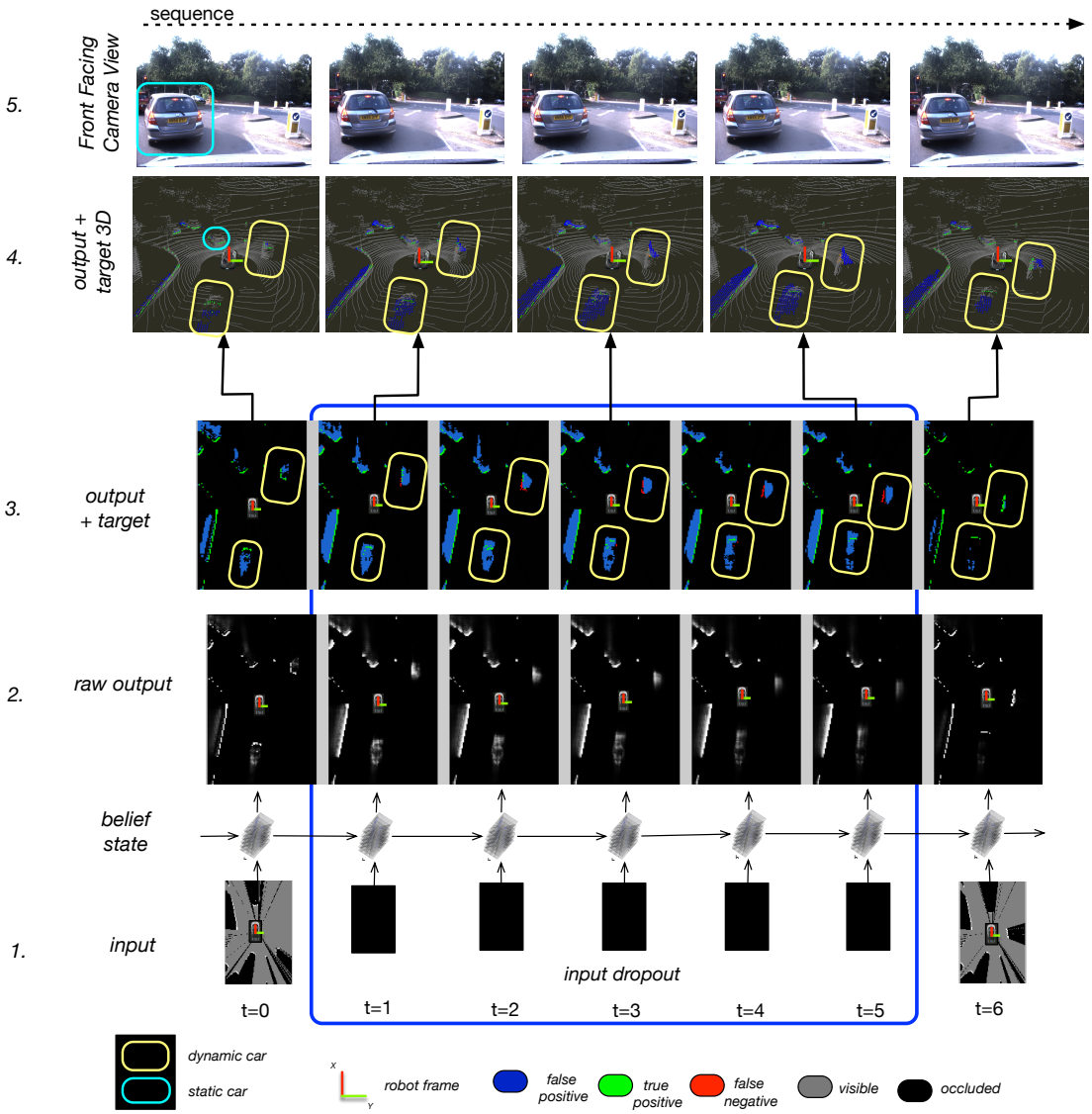


Figure 5.20: Left to Right, Bottom to Top: Example sequence (0.6 sec) of dynamic and static object tracking through occlusion when the vehicle is stopped at a red light. The input is dropped out from $t = 1:5$ meaning that the network output represents full prediction. We highlight the moving vehicles in yellow contours. The output is evaluated against the ground truth in lines 3 and 4, line 4 representing the output projected back into the original 3D point cloud. Line 5 shows the front facing camera view, where we circle the stopped vehicle in cyan and show its corresponding location in the point cloud at $t = 0$. The network is able to both remember the position of static obstacles, and correctly track the moving vehicles. It also learns to fill the cars partially which is a useful addition to the tracker predictions. Best viewed on pdf.

Tracking with constant velocity Figure 5.21 illustrates an example of the informed model accurately tracking both dynamic and static objects through occlusion when travelling along a busy part of Summertown, Oxford. The top line corresponds to robot ego-motion, showing the vehicle travelling at a steady velocity of 20km.h^{-1}. The second line from the top represents the right-hand side camera view for visualisation purposes. As the input is dropped out from $t = 2 - 6$ this represents full prediction. Timesteps $t = 7 - 8$ contain a natural occlusion of the pedestrians (cyan) by the passing car (yellow). In both situations, the model accurately translates the position of the occluded static pedestrians (cyan), as well as accurately captures the dynamics of the vehicle which passes between the pedestrians and the sensor (yellow). It is also able to track the static structure correctly to the left of the vehicle.

Tracking on acceleration Figure 5.22 illustrates a second sequence where the model tracking static and moving objects whilst gradually accelerating along the road. The two forward and right-hand side camera views highlight two dynamic cyclists (pink contours) that the model is able to track through both model input dropout and natural occlusions produced by static vehicles stopped in traffic (yellow contours). We highlight in red contours two predictions that the model cannot predict as they constitute new previously unseen parts of the world as the vehicle travels forward through model dropout ($t = 3$ and $t = 23$).

Improved tracking ability compared to the baseline model Figure 5.23 compares the tracking performance of the model to that of the baseline model which does not consider the vehicle ego-motion. The five frame sequence represents full prediction of the scene, as the input has been blacked out to simulate total occlusion. The vehicle is driving at a constant velocity of 45km.h^{-1}. Highlighted with a pink rectangle is a moving vehicle passing by, and tracking accuracy is compared to that of the visible ground truth. Although the ego-motion model achieves very satisfactory tracking of the moving vehicle (bottom output row), we find that the baseline model gradually fails to produce any prediction overlap with the actual

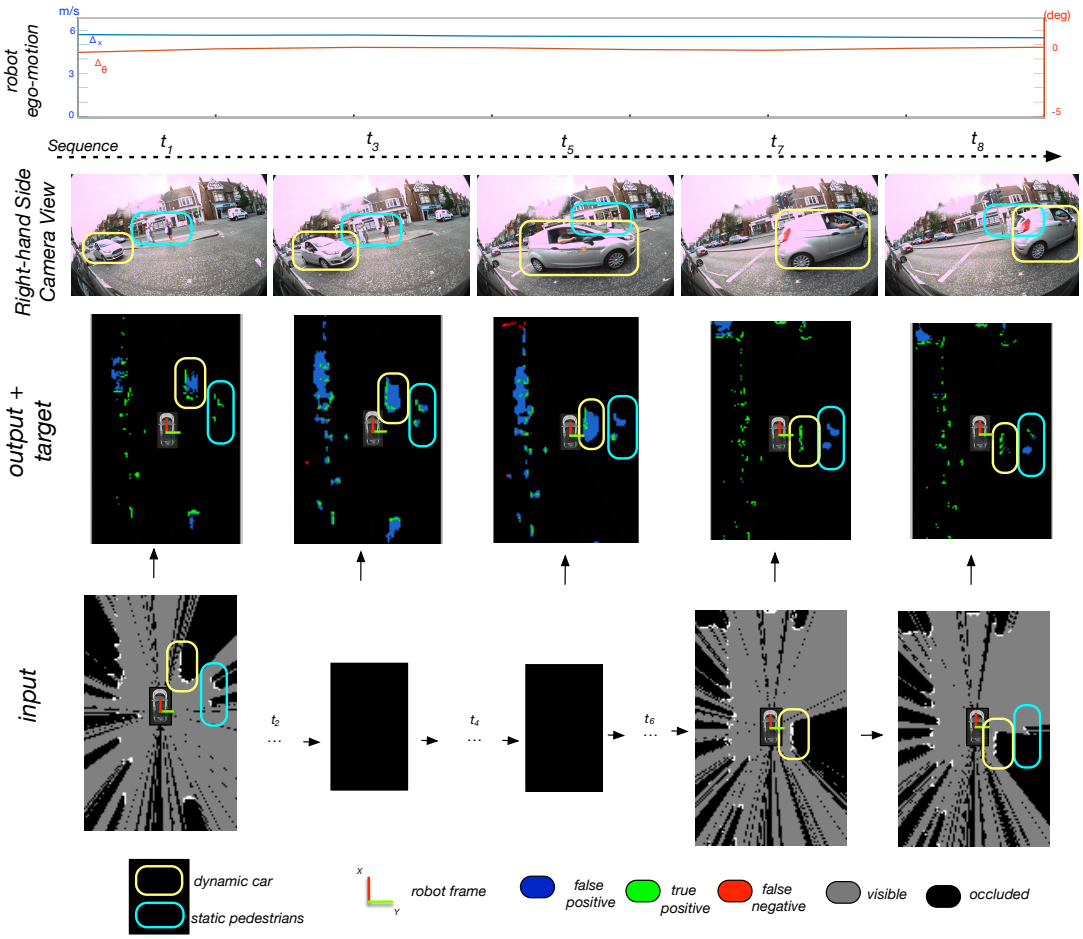


Figure 5.21: Left to Right, Bottom to Top: Example sequence (0.8 seconds) of dynamic and static object tracking through occlusion by the model. The top row shows the camera view to the right side of the vehicle for illustrative purposes. The sensor is mounted on the roof of the vehicle which is travelling at a constant speed of 20km.h^{-1} throughout the sequence. We highlight the occlusion of static pedestrians (cyan contour) caused by a moving vehicle passing between them and the sensor. The occlusion of the moving vehicle (yellow contour) is created by our blanking out of the input from time step $t_{2,\ldots,6}$ included (illustrated by the black network input). The output of the network is evaluated against the visible ground truth. Best viewed on pdf.

position of the passing vehicle at the last frame as visible with the increasing area of false negative predictions (top output row). The static background to the left of the scene is however accurately tracked in both models. This sequence suggests that the baseline model may be translating the scene with respect to a learned vehicle ego-motion. The F1 predictive performance shown in the top right hand corner illustrates the highest performance of the ego-motion model. However, though tracking the dynamic vehicle represents the most important part of the tracking task

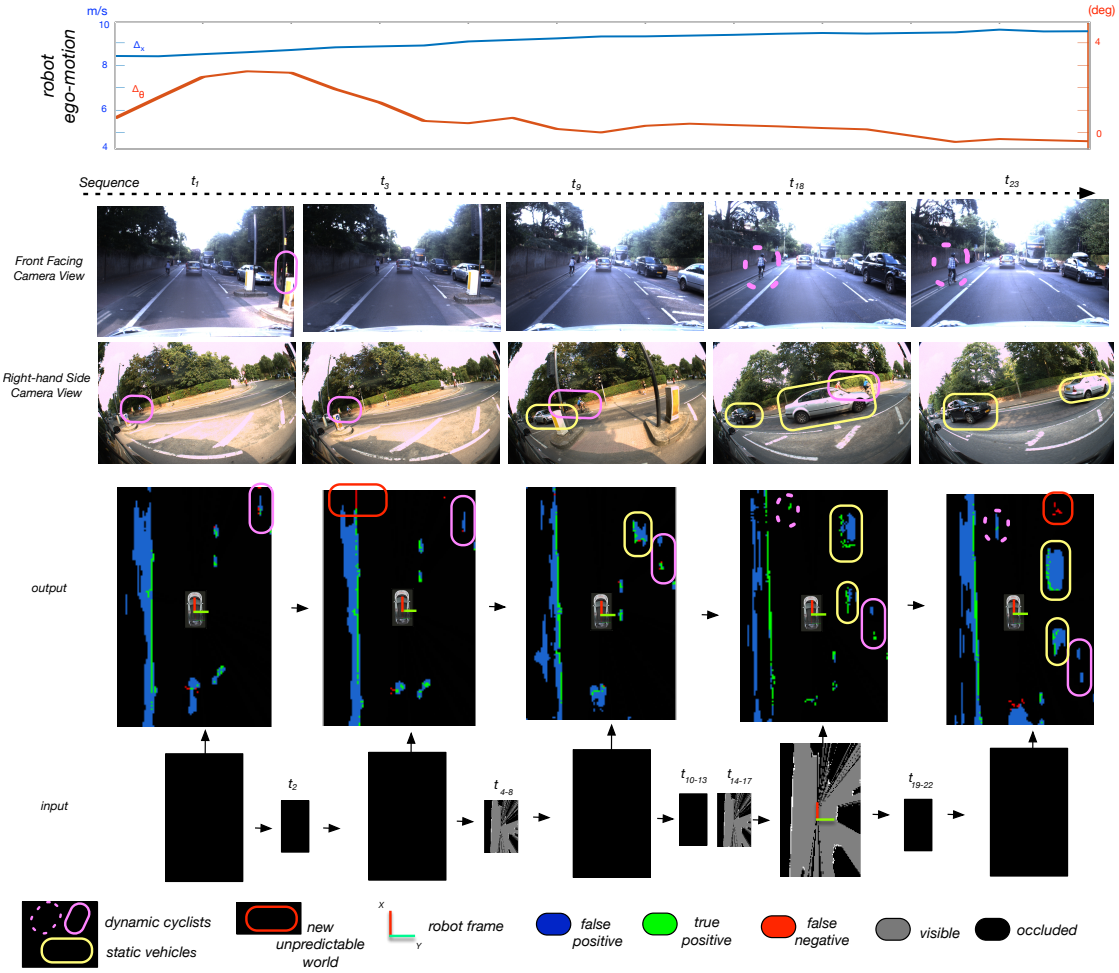


Figure 5.22: Left to Right, Bottom to Top: Example sequence (3 sec) of dynamic and static object tracking through occlusion when accelerating along a road as shown in the top line robot ego-motion. The two camera images show the front and right hand side views where the network is able to track two moving cyclists on either side of the road through occlusion due to drop-out, as well as static vehicles stopped in traffic. The output is evaluated against the visible ground truth. As the vehicle travels forward (along the X axis), it discovers new parts of the world that it cannot predict (red contours). Best viewed on pdf.

in this sequence, it contributes to little F1 measure which is reflected in the relatively small F1 performance observed ($\Delta F1 \sim 0.1$). As the dataset is over-weighted by static background, we hypothesise that this, along with learned ego-motion, may be an explanation for the high F1 measure observed for the baseline model.

Tracking whilst turning To finish this section, we observe a four second long sequence where the vehicle is turning at the Keble triangle, Oxford in Figure 5.24. The top of the figure illustrates vehicle ego-motion and forward camera throughout

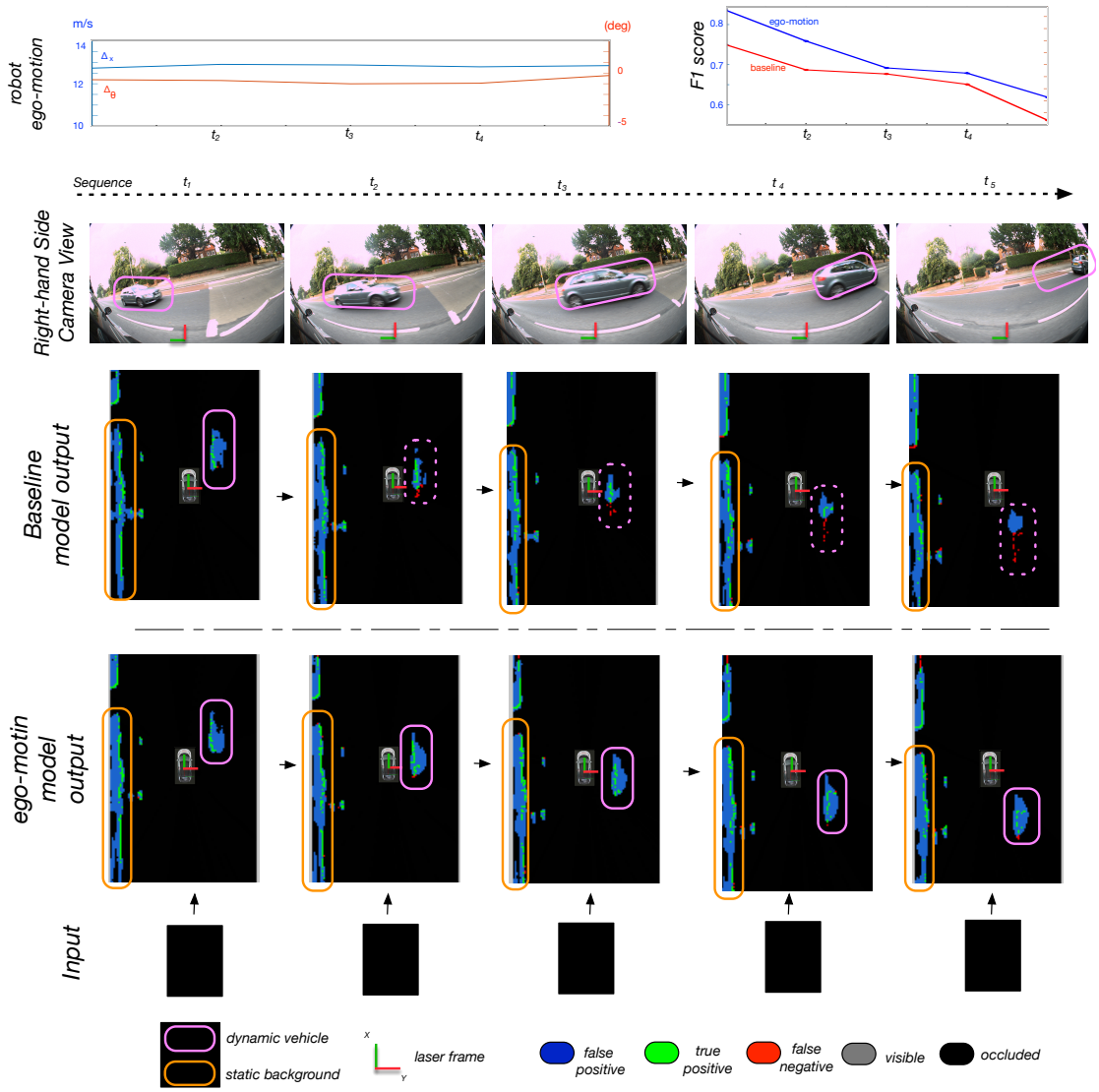


Figure 5.23: Left to Right: output sequence (0.6 seconds) of dynamic and static object tracking through occlusion as predicted by the ego-motion model with spatial transformer (bottom output sequence) and the baseline model (top output sequence). The output predictions are compared to the visible ground truth. In this sequence, the robot is moving at a steady 45km.h^{-1} along a straight road (top left). The input to the network is blacked out for the entire sequence shown and simulates total occlusion of the scene. The top row shows the camera view to the right side of the vehicle for illustrative purposes. Where the ego-motion model accurately tracks a passing vehicle (pink contour), the baseline model fails to capture the dynamics of the vehicle (dashed pink contour). This is particularly visible on the last frame, with complete false negative and prediction and trailing false positive area for the baseline model. Both models accurately capture the position of the static buildings to the left of the vehicle. This suggests that the baseline model may have learned one of several vehicle ego-motions, and highlights the positive contribution of the spatial transformer module in the architecture. Best viewed on pdf.

the sequence, as well as the vehicle trajectory overlapped on Open Street Maps for context. As can be seen, the vehicle is constantly turning at a rate of $10-30$ deg.s^{-1} with an increasing forward velocity as it completes the turn. The network output is shown compared to the visible ground truth (true positive, false positive and false negatives), and projected back into the 3D point cloud. We show the network grid with a white rectangle spanning $Y \times X = 18 \times 24\text{m}^2$ around the robot. Highlighted in pink and cyan are three moving cars, in light green is a moving bus, and in yellow is a static building surface. The bus and cyan vehicle are registered in the forward facing camera for context. Finally, the frames with a red contour represent complete prediction as the input is dropped.

We point out a few features of the predictive sequence. The network is able to correctly track the bus in the first eight frames of the sequence despite the initiation of the turn to the right. Notice that throughout the turn the model is able to correctly track the longitudinal building structure (yellow) along the left side of the street, particularly neatly during complete occlusion. The two pink vehicles are correctly tracked throughout the sequence, even as one of the cars turns right along the same street as the robot in the last eleven frames. The robot is also able to track the cyan vehicle to the front right of it; however we highlight three frames with a dotted cyan contour where the robot fails to track the vehicle through occlusion. It is difficult for the network to predict the sharp right turn of the vehicle as it is not aware of the layout of the road ahead. We suggest that context is here an essential ingredient to the tracking task, as without awareness of the road layout the network cannot be expected to predict evolving turns consistently. We further explore the idea of incorporating context in the tracking framework in the following chapter.

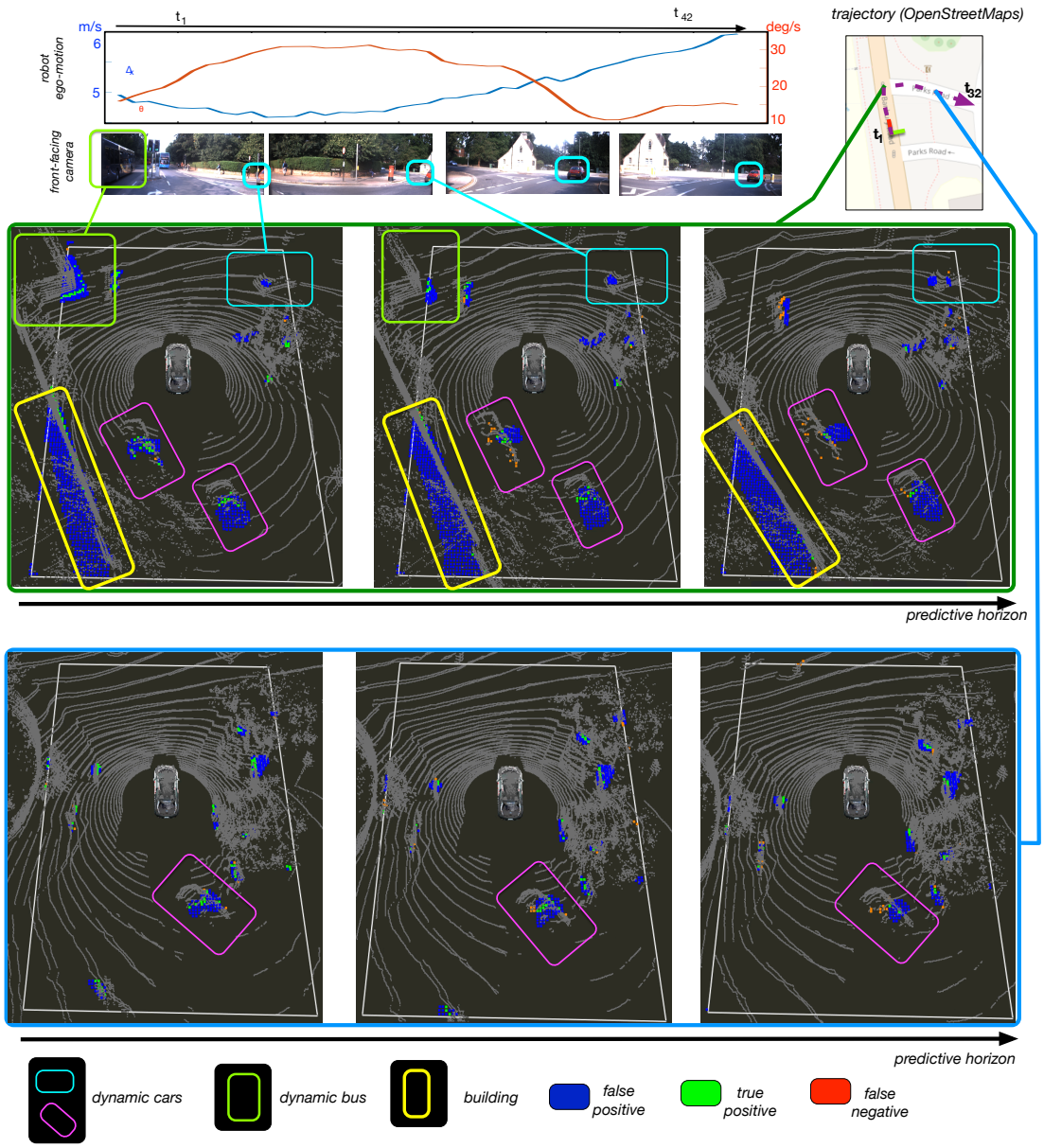


Figure 5.24: Two one-second long network full predictions (input dropped out) projected into the 3D point cloud and visualised from a bird's eye view. The robot is turning throughout the sequence. False positives (blue) represent the network's ability to predict through occlusion, and are therefore valuable. Pixel-wise cost computation produces false negatives (red) though this is actually not a problem here as these are closely bordered by both true positives (green) and false positives (blue). See text for sequence explanation. Best viewed on pdf.

5.5 Transfer of Knowledge

We close off this chapter by revisiting the semantic classification through inductive transfer of Chapter 4 and look at learning to classify the predicted output occupancy into dynamic and static objects on synthetic **Dataset 2**. We extend the learning task to predicting both unoccluded scene occupancy $\mathbf{y}_t$ and scene semantics $\mathbf{c}_t$. We leverage transfer of knowledge by first training the model on the task of tracking with ego-motion knowledge (see experimental Section 5.4.3), and then train a classifier to semantically label the predicted output scene using the hidden representation $\mathbf{h}_t$.

5.5.1 Problem Formulation

We wish to assess the model's ability to semantically classify the scene once it has learned to track. We therefore consider the pre-trained tracking model of Section 5.4.3 which incorporates ego-motion (GRU3_VO) and train a classifier to semantically classify the predicted occupancy output as illustrated in Figure 5.25.

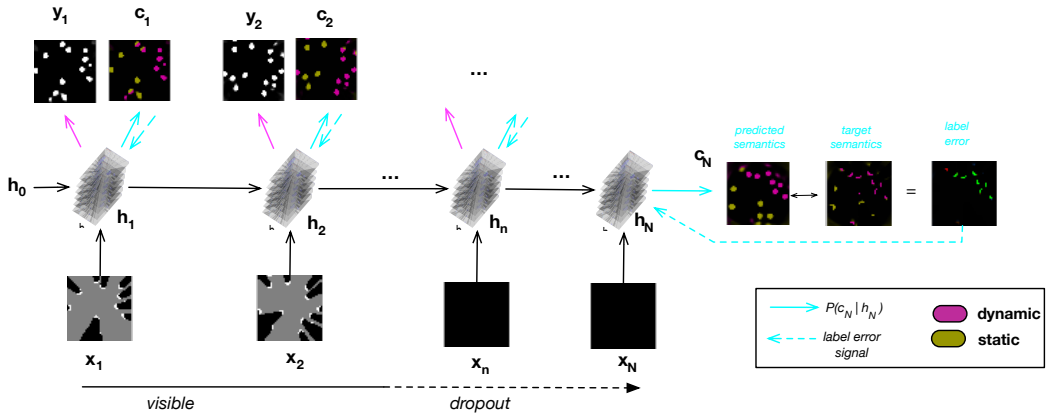


Figure 5.25: Once trained on the task of tracking, the network is trained to predict the semantics of the predicted output occupancy. Labelling errors are only backpropagated through the semantic decoder (dotted cyan line).

We maintain the same training procedure whereby the network is trained to predict future scene semantics $p(\mathbf{c}_{t+n}|\mathbf{x}_{1:t}) = p(\mathbf{c}_{t+n}|\mathbf{h}_{t+n})$ so as to verify that the network is able to use its memory $\mathbf{h}_{t+n}$ to label the scene. At every timestep t, $p(\mathbf{c}_t|\mathbf{h}_t)$ is modelled as a two-layer neural network taking as input the hidden state

and outputting class probability with an output sigmoid non-linearity (Figure 5.26, a). The output pixel-wise class predictions are modelled as independent Bernouilli variables where the positive and negative classes represent dynamic and static objects respectively. The learning objective aims to maximise the following log-likelihood:

$$\log\, p(\mathbf{c}_t|\mathbf{h}_t) = \sum_i c_i\, \log(p_i) + (1 - c_i)\log(1 - p_i), \tag{5.6}$$

where p_i represent the decoder output pixel-wise predictions, and c_i are the corresponding binary target labels.

We only wish to assess the model's ability to label the predicted scene semantics rather than penalise the network on its ability to predict scene occupancy in the first place. We therefore mask the output cost with the tracker output predictions as illustrated in Figure 5.26, b.

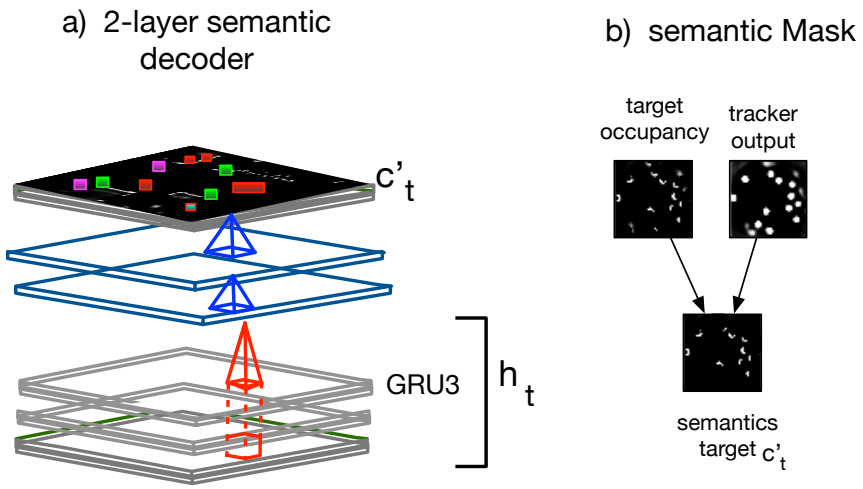


Figure 5.26: The semantic decoder consists in a two-layer network with sigmoid activations (a). The output predictions are compared to $\mathbf{c}'_t$ which correspond to the target labels $\mathbf{c}_t$ masked by the visibility grid and tracker output predictions (b).

5.5.2 Experimental Evaluation

We consider **Dataset 2** which figures a moving sensor, and both dynamic and static obstacles. As labels are readily accessible, we train on the entire dataset of $D_{train} = 800$ sequences, and test on the $D_{test} = 200$ sequences. We consider the pretrained tracker GRU3_VO of Section 5.4.3. We then train the classifier to semantically classify the predicted tracker output. The loss is only backpropagated through the classifier decoder meaning that the weights of the tracker are fixed. Training consists in sequences of $N = 20$ where the first 10 inputs are shown, and the last 10 inputs are dropped out.

We use the Adagrad optimizer with an initial learning rate parameter of 0.01 and train for 200 epochs with early stopping on an NVIDIA Titan X GPU. We do not use any weighting for the classes as the proportion of dynamic vs. static is relatively balanced (1.5) and we did not observe any issue during training.

Quantitative We observe the mean and standard deviation of the F1 measure over the entire sequence length of the test set in Figure 5.27. Surprisingly, the mean initially increases during the first $t = 1 : 5$ timesteps of the sequence when the input is shown. As illustrated in the sequences shown below, we suggest this be due to the fact that the model requires a few timesteps of seeing the input data before understanding the dynamics of the scene. At timestep $t = 1$, the hidden state is blank and the model "sees" the scene for the first time so it cannot know which objects are dynamic or static. As the hidden representation captures the scene with new incoming input, it becomes capable of accurately predicting scene dynamics as reflected in the high F1 measure. As the input is dropped out, there is almost no loss of performance reflecting the model's ability to correctly capture scene dynamics. We observe some variance of the data; this is in part due to the fact that some sequences only contain static obstacles, for which the F1 measure will return 0. We observe several predictive sequences in the following section.

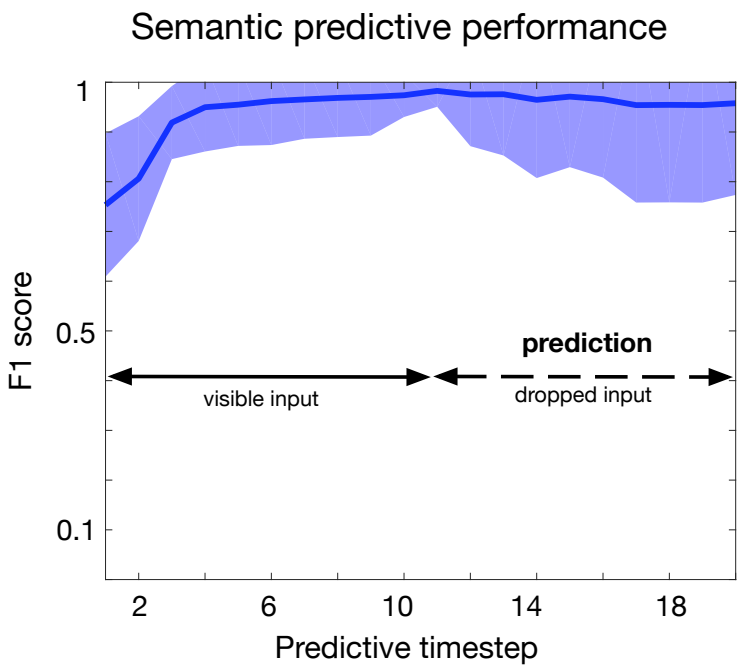


Figure 5.27: F1 mean and standard deviation on semantic classification using a pre-trained model on the task of tracking on **Dataset 2**. Timesteps $t = 11 - 20$ correspond to prediction into the future as the input is dropped out. The curve initially increases with time as the model gradually captures the dynamics of the scene.

Qualitative Figure 5.28 shows an example sequence from the test set. The input (d.) is only visible from timesteps 1-10 (bottom line). The network outputs (c.) both scene label (pink), and scene occupancy (cyan). The output predictions are masked by the predicted occupancy and compared to the target labels (a.). Target dynamic objects are colored purple, static objects are colored bronze. The resulting predicted labels with target is shown in line b as false positive, false negative, and true positive for the dynamic class.

We observe that the model is able to accurately predict the dynamic objects from $t = 4$ as seen in line b. At timestep $t = 12$, we circle dynamic and static objects with bronze and purple contours respectively (lines a,b,c) to highlight the model's ability to tell the two classes apart in the predicted occupancy output. In the predicted labels output of line c., we see that the model has learned to mask out the static objects (black regions).

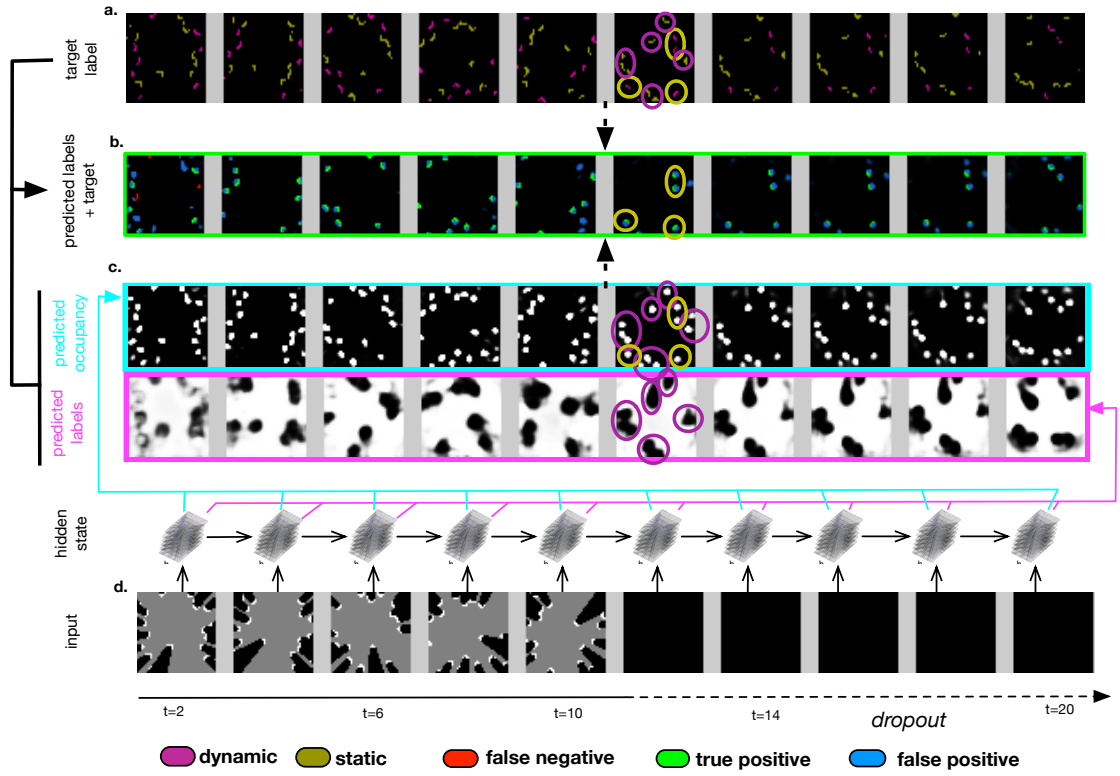


Figure 5.28: Semantic labelling prediction for a moving sensor viewing both dynamic and static objects, with (green) and without (green) pre-training on the task of tracking. In this context, false positives (FP) reflect an incorrect prediction. The model that was not pre-trained on the task of tracking completely fails at capturing scene semantics and so is not shown here. Best viewed on pdf.

We look at two further sequences in Figure 5.29. These two sequences are only shown from $t = 12 - 20$ which corresponds to full prediction as the input is dropped out. The top sequence no longer contains dynamic obstacles after $t = 12$, while the bottom sequence shows a mix of dynamic and static obstacles. In both cases, the network is able to accurately tell apart both classes. We circle dynamic (purple) and static (bronze) objects in both sequences for clarity.

5.6 Conclusion and Future Work

In this chapter, we proposed an approach to performing object tracking for a mobile robot travelling in urban environments. We find that transforming the hidden state according to the estimated ego-motion with the use of a spatial transformer module is an effective way of decoupling robot motion from world dynamics and enabling

tracking in motion. Results on synthetic data shows compelling performance. Deployment on real world data shows favourable results compared to our baseline model both quantitatively and qualitatively. We also find that once the network has learned to track, it can semantically label the scene into static and dynamic objects. This is a valuable feature for a robotics system, which would be interesting to test on real world data as future work.

The current implementation presents a number of limitations that could be investigated in future work.

On the one hand, the trained network has not been tested on data outside of the central Oxford area and it is not known how well it would perform in a different urban environment. A city such as San Francisco has similar high level structure and motion patterns, yet presents important differences such as larger avenues, steeper hills, tram lines, different vehicle and pedestrian dynamics and right-hand traffic rules. It would be interesting to observe which aspects of scene occupancy and semantics an Oxford-trained model deployed on a similar sensor-vehicle setup would do well at. We hypothesise that vehicle pitch would need to be considered and that the model may either require fine-tuning of the weights, complete retraining, or there may be the option of transferring knowledge of general tracking patterns to a different scene context whilst avoiding catastrophic forgetting, as demonstrated in sequential task learning work Rusu et al. [2016]. In the next chapter we investigate introducing scene context as a conditioning input to adapting network dynamics to the scene at hand.

Further, our implementation is completely model-free. We wished to investigate what an end-to-end system can learn by simply watching the world evolve when deployed in an area with no prior knowledge. This is valuable when observing an environment or system whose dynamics and interactions are either unknown, complex, or require parameter tuning and approximations. It further does not require prior scene segmentation, a priori object knowledge and object classification. There is value however in incorporating models when they are known and allow restriction of the search space: vehicles in urban environments do not disappear,

they remain on roads and follow driving rules with constrained dynamics. We extend our system in this direction in the next chapter where we consider scene context to guide predicted motion patterns. Further work could for example additionally introduce models via banks of pre-designed filters representing expected dynamics of pedestrians, vehicles, and bicycles.

Finally, current model predictions extend one second into the future which is shorter than the ten second horizon typically required by a planner. The reasons for this are threefold. Firstly, we are interested in exploring the capacity of the tracking framework to handle vehicle ego-motion, and are thus interested in introducing architectural changes rather than focusing on providing an output readily usable by a planner. Secondly, in the current implementation where the world is represented as a Cartesian grid around the robot, the greater the prediction horizon, the greater the unknown region ahead of the robot. Predicting further into the future therefore requires considering a larger grid and longer input sequences, which increases computing memory and training time linearly. A ten second prediction would roughly require a 2.5 times increase in both sequence length ($N = 100$ frames) and grid size ahead of the robot for a minimum horizon of 50 meters which exceeds the memory capacity of a 6Gb GPU. The current architecture would need to be reconsidered by, for example, moving away from the spatial grid and reducing the dimensionality of the hidden state with fully connected layers or spatial stride and considering higher cost weighting of future frames for efficient learning. Radar may further need to be considered in order to provide a visual horizon of up to 200 meters. Lastly, it is unreasonable to predict 10 seconds into the future without consideration of object interactions with scene context to capture multi-modal possible futures. In the next chapter we work towards improving predictive horizon by introducing road layout. This additional information enables the network to adapt predictions to the underlying scene context and offers the option of predicting multi-modal paths at intersections.

With some of these limitations in mind we consider incorporating scene context to guide tracker predictions. Context can take the form of camera imagery, or a

2.5D input with laser scan at varying heights. In the next chapter we consider a bird's eye-view of road layout as context information.

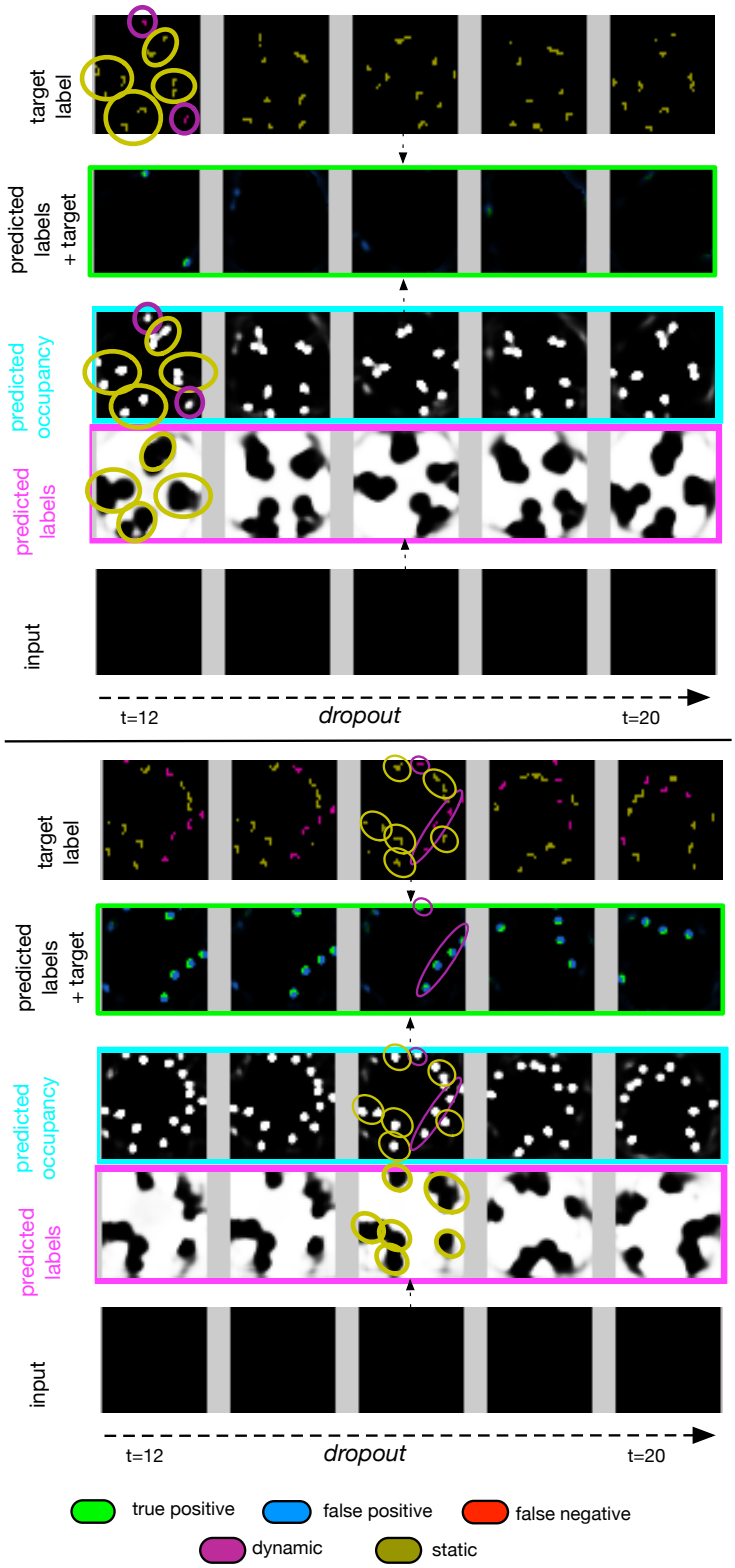


Figure 5.29: Full prediction of semantic labelling for two sequences. The top sequence presents no dynamic objets after $t = 12$ which is correctly captured by the network. The bottom sequence shows a balanced distribution which is also correctly classified. Best viewed on pdf.

"For me context is the key – from that comes the understanding of everything."

— Kenneth Noland

6

Learning to Predict the World State in Context

In the previous chapter, we extended our tracking system to a moving platform by incorporating a spatial transformer module to update the hidden state according to vehicle ego-motion. We showed that the proposed solution was able to decouple sensor motion from world motion on synthetic data and demonstrated favourable performances on real world data. In this chapter, we further aim to adapt our tracking system to real world environments by incorporating scene context in the tracking models as we argue that context is an essential component of situational awareness. Objects in the world behave differently depending on their surroundings. Particularly, in the context of urban driving, vehicles follow road networks in a very structured way. We propose an extended architecture able to adapt its motion models to the underlying road layout. We showcase promising results on real world overhead maps.

6.1 Motivation

Looking at the static image of a scene, humans are very good at understanding the context and possible interactions between objects present. They can also easily infer the chain of dynamic events that are likely to happen in the near

future, in the perceived setting. In urban environments, a driver can effortlessly predict the general traffic patterns at an upcoming intersection, based on visual cues from the road layout, road signs, other drivers' behaviour etc. In Figure 6.1 we illustrate how at a roundabout, strong priors lead drivers to predict that vehicles will be driving clockwise/anti-clockwise and that they may exit or enter at different points depending on the lane they are observed to be in. Similarly, at a T-junction, a car is expected to turn along with the road even if their current motion pattern is a straight line.

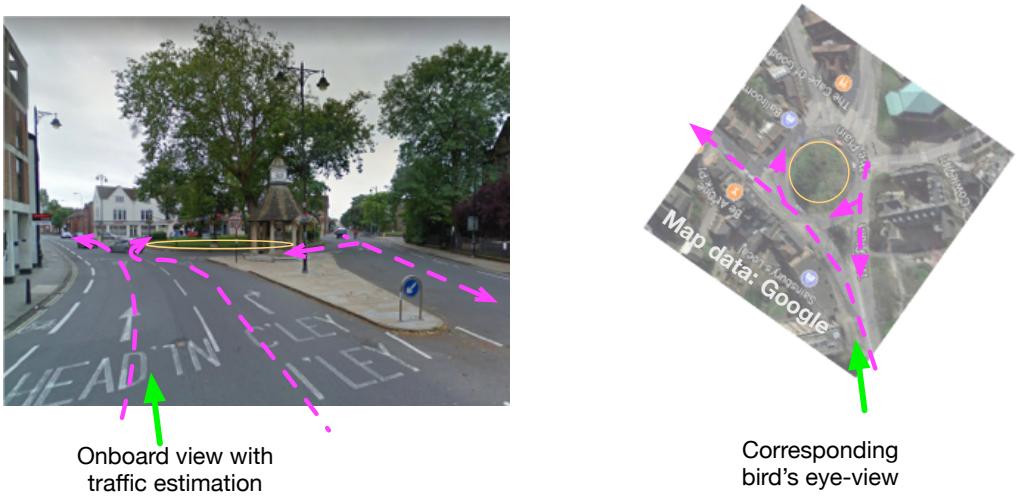


Figure 6.1: When coming up to a roundabout (left image), human drivers can effortlessly predict possible motion patterns (pink dotted lines) of vehicles entering and exiting the roundabout (orange oval). To the right is represented the corresponding bird's eye-view extracted from GoogleMaps.

This strong coupling between scene context and motion models has been explored in traditional tracking pipelines. Some bodies of work suggest motion models that are class or environment specific, whilst others learn to predict traffic patterns from road layouts, though are not directly connected to a real-time tracking system.

We consider scene layouts as a proxy for scene context in the context of urban autonomous driving. We introduce *dynamic filters* in our network to condition motion models to the local scene and show improved tracking performance on both synthetic and real-world layouts.

6.1.1 Contributions

In this chapter we extend our tracking framework to adapt its motion models to urban road patterns. We observe the limitations of our baseline model in predicting pertinent dynamics at road intersections and roundabouts, and investigate two mechanisms for adapting network predictions appropriately. Inspired by our findings when incorporating static biases in the hidden state in Chapter 4 and by the recent work of [De Brabandere et al., 2016] on *Dynamic Filter Networks* we propose conditioning network dynamics on the scene layout.
We conduct a proof-of-concept of synthetic scenarios and show that we are able to incorporate scene context that generalises to unseen layouts. We then extend the analysis to real-world layouts and link usability to real-world scenarios in a setting where GPS positioning is available.

In doing so we make the following contributions:

- we propose an end-to-end trainable framework for predicting motion according to a scene layout, without the need for annotated ground truth or pre-processing;

- we demonstrate the effectiveness of dynamic filters for tayloring motion models to the scene at hand;

- we propose an extension to real-world applications using GPS localisation in overhead maps.

To the best of our knowledge, at the time of conducting this research ours was the first work to consider scene layout as a prior in a deep tracking framework.

We present our extended architecture and improved results on a synthetic dataset in Section 6.2. We extend our findings for deployment in real-world scenarios in Section 6.5.

6.2 Tracking According to Scene Layout

In this section we introduce the mechanisms chosen to incorporate scene context in our tracking framework and showcase the ability of the extended framework to track with context on synthetic layouts. We set this work in the context of predicting motion according to an underlying road layout, and assume that only local information is required to explain the perceived motion. We also assume that the underlying road layout is complete and readily available. We address these limitations and propose extensions in our discussion and future work.

6.2.1 Problem Formulation

Our tracking objective remains that of predicting the future unoccluded state of the world $\mathbf{y}_{t+n}$ at time $t+n$, given a sequence of occluded observations of the world $\mathbf{x}_{1:t}$. Empty inputs $\mathbf{x}_{t+1:t+n}$ are provided to force the network to iteratively update and use its belief $\mathbf{h}_t$ to infer future state occupancy. We model individual output occupancy grid predictions as independent Bernouilli variables:

$$p(\mathbf{y}_{t+n}|\mathbf{h}_{t+n}) = \prod_i (p^i_{t+n})^{y^i_{t+n}} (1 - p^i_{t+n})^{(1-y^i_{t+n})} \tag{6.1}$$

We train our network to maximise this distribution on the visible available ground truth parts of the grid. Gradients are backpropagated through the dynamic filter CNN module similarly to any feed forward network. We refer the reader to Section 2.2.5 for more details.

6.2.2 Network architecture

The original tracking architecture considers traditional convolutional filters which consist in a multiplicative kernel W and added bias b applied to a given input feature map $\mathbf{x}$ to produce an output $\mathbf{y}$:

$$\mathbf{y} = W * \mathbf{x} + b$$

In Chapter 4 we adapted these filters to produce pixel-wise biases $b_{i,j}$ which learned place-specific information, rather than a bias b shared across the entire

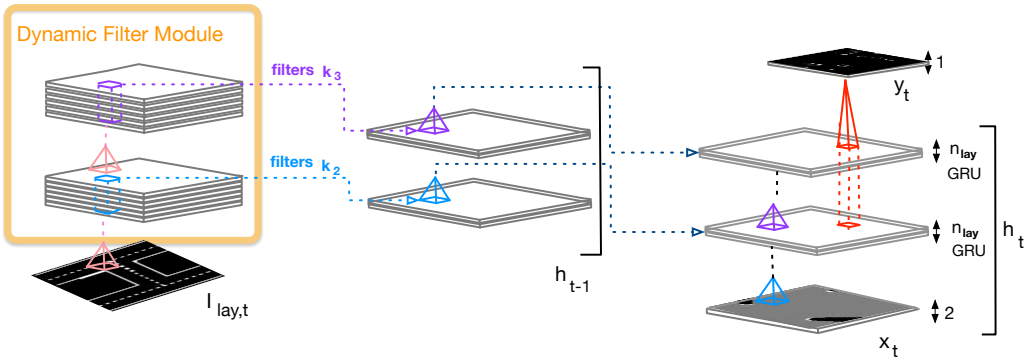


Figure 6.2: Our extended architecture for tracking in context features a dynamic filter module which conditions network filters on the layout at time t. The conditioned filters are used to update the hidden state accordingly. They can be consist in dynamic kernel weights, dynamic biases, or a combination of both.

feature map. Kernel weights W remained shared across the feature map and both W and $b_{i,j}$ values were learned at training time and fixed at test time.

Based on these findings, to adapt motion patterns to the layout at time t, we give the network the capacity to *dynamically* produce both pixel-wise biases $b_{I_{lay},i,j}$ and local kernel weights $W_{I_{lay},i,j}$ by conditioning their values on the scene layout I_{lay}. In doing so, we draw inspiration from the dynamic filter module developed in De Brabandere et al. [2016] to adapt network convolutions to a given input. We refer the reader to the related works Section 2.1 for a detailed presentation of the dynamic filter module.

Figure 6.2 represents our updated architecture incorporating dynamic filters. We consider a road layout I_{lay} which reflects a two-way road system where lanes are separated by dotted lines. In our baseline three-layer model, a convolutional network (CNN) produces three sets of dynamic filters k_1, k_2, k_3 at every grid location, applied to the three layers of the hidden state update from $t-1$ to t. This is different to the baseline architecture where filters are shared across pixel locations, learned at training time, and fixed at test time.

These dynamic filters can apply to the convolutional filter kernel weights (Figure 6.3 b, $\mathbf{W}_{xhf_{\{ij\}}}$ in equation below), to the filter biases (Figure 6.3 a, $\mathbf{b}_{z\{ij\}}$

in equation below) or both. Crucially they are produced for every pixel location i, j by conditioning their values on the local layout.

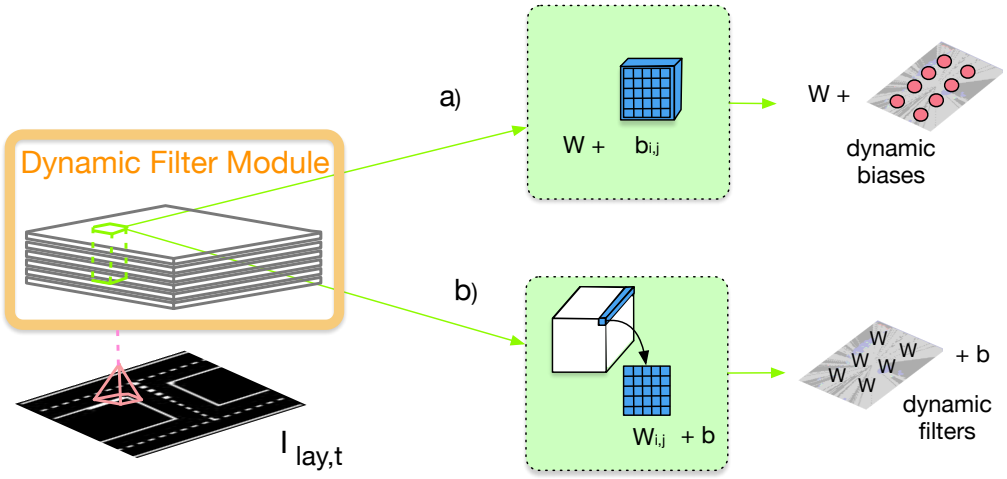


Figure 6.3: At time t, the layout can either condition pixel-wise dynamic biases (a), or pixel-wise dynamic filters (b). Both dynamic biases and filters can be used in one or all of the convolutional operations of the gated recurrent unit (GRU) equations. Simple CNNs suffice to convert the layout into relevant information during the hidden state update.

Dynamic convolutional filters can be used for either/all of the gated recurrent unit reset ($\mathbf{r}_t$) and forget ($\mathbf{f}_t$) gates, and candidate memory state ($\bar{\mathbf{h}}_t$) update equations:

$$\mathbf{f}_t = \sigma(\mathbf{W}_{xhf\{ij\}} * (\mathbf{x}_t + \mathbf{h}_{t-1}) + \mathbf{b}_{z\{ij\}}) \ , \tag{6.2}$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_{xhr\{ij\}} * (\mathbf{x}_t + \mathbf{h}_{t-1}) + \mathbf{b}_{r\{ij\}}) \ , \tag{6.3}$$

$$\bar{\mathbf{h}}_t = \tanh(\mathbf{W}_{xh\{ij\}} * (\mathbf{x}_t + \mathbf{r}_t \circ \mathbf{h}_{t-1}) + \mathbf{b}_{h\{ij\}}) \ . \tag{6.4}$$

Here $*$ denotes dilated convolution described earlier and $\circ$ denotes element-wise multiplication. The subscripts $\{ij\}$ indicate separate bias values and kernel weights for every pixel i, j of the feature map they are applied to.

In our experimental section, we compare the tracking performance when producing local dynamic filter weights ($\mathbf{W}_{xhf\{ij\}}$), local dynamic filter biases ($\mathbf{b}_{r\{ij\}}$) or both. We refer to models which have access to the underlying layout as *informed* models. We find that dynamic filters provide an elegant and efficient way of conditioning network parameters on the scene layout to produce relevant dynamics.

This is in contrast to the traditional convolutional filters and pixel-wise static biases which fail at this task as we show in our experimental evaluation, and is our extension of choice for incorporating scene layout in the tracking system.

6.3 Datasets

We test the capacity of our extended network to track the world according to the underlying road layout on two datasets. A first *RoadLayouts* consists in a selection of layouts resembling that of simple urban road patterns with single lanes and turn options. This dataset is used to highlight the current model's limitations at tracking according to context, and as proof of concept for the dynamic filters' efficacy in adapting its predictions to the local layout. A second *GoogleStreetMaps* dataset consists of synthetic paths drawn on real world Google Map overhead views of urban streets. Unlike the first dataset it contains multimodal intersections and roundabouts which provides a more realistic and challenging setting for deployment in real-world scenarios. A third dataset consists of synthetic paths produced on the road network provided by Open Street Maps (OSM) [Haklay and Weber, 2008], in the Oxford city centre, UK. It is similar in nature to the *GoogleStreetMaps* dataset as it corresponds to real world road networks with multimodal intersections and one-way streets. It however is open access, and can provide greater applicability to real world usage as a robot in the real world can localise in the overhead map, access the underlying map topology for context, and learn or use motion priors relevant to its location.

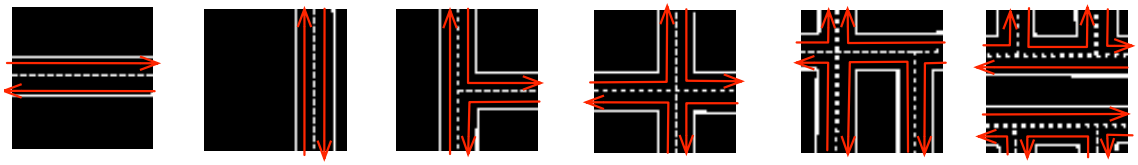
We provide an overview of all three datasets in Table 6.1 and detail them in the following sections.

Table 6.1: Characteristics of the three datasets considered, including number of training and test frames.

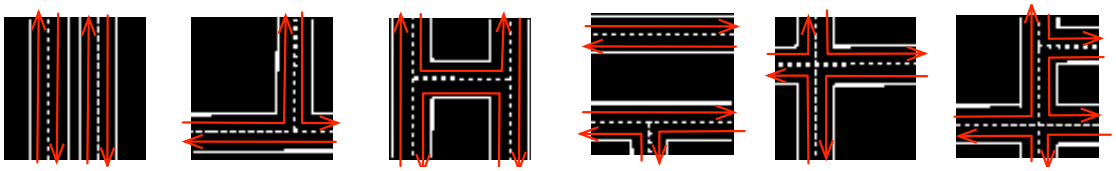
Dataset	Training Frames	Test Frames	Characteristics
RoadLayouts	81600	8000	25 synthetic layouts
GoogleStreetMaps	62000	8000	41 real world layouts
OpenStreetMaps	267360	66840	1670 OSM segments

6.3.1 Road Layouts

We consider 25 layouts split into 17 for training, and 8 for testing which we illustrate in Figure 6.4. Layouts are represented in white on black, with dashed lines representing lane separation. The motion of objects along the road layouts are represented with red arrows for visualisation purposes. On such layouts, motion patterns are explained by the local road curvature, which we aim to capture to adapt predictions accordingly.



(a) Training set layout samples



(b) Test set layout samples

Figure 6.4: Examples of synthetic road layouts with motion direction (red arrow). The layouts represent urban road patterns where objects move on the left hand side (UK driving rules). Layouts are located at random positions in the scene and present turns, but they do not have intersections. Training (top row) and Test (bottom row) layouts are different from one another.

In keeping with past chapters, we produce circular objects which move along the layouts in a deterministic way and constant velocity, and which are observed by a range sensor situated at the bottom of the grid as illustrated in Figure 6.5 on the next page. We perform ray tracing on the obstacles and produce a 1441 laser scan with angular step of 0.25 degrees. Both layout and occupancy grids are of size 63×63.

Our training and test set consist of $D_{train} = 2040$ and $D_{test} = 400$ sequences of size $N = 40$ and equally distributed over the 17 and 8 layouts of the training and test set respectively.

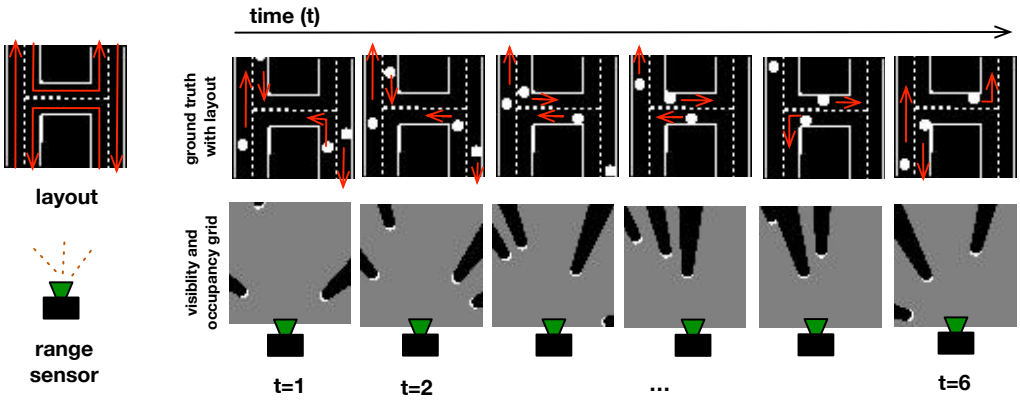


Figure 6.5: Typical sequence of objects moving along the synthetic road layouts. Objects follow the direction of motion (illustrated with red arrows), keeping straight lines to the left and dashed lines to the right. Ground truth (top row) occupancy is only partially observable from a range sensor (bottom row). White represents occupied space, gray visible free space, and black is occluded unknown space. Test layouts i) and ii) are either/and incomplete and smaller scale than the ones in the training set.

6.3.2 Google Street Maps Layouts

Our *GoogleStreetMaps* dataset consists of 41 layouts extracted from the Oxford city centre, UK, as provided by the GoogleMaps API. The data is grayscale, contains building structure, and is split into a training set of 31 layouts, and a test set of 10 layouts. A selection is shown in Figure 6.6.

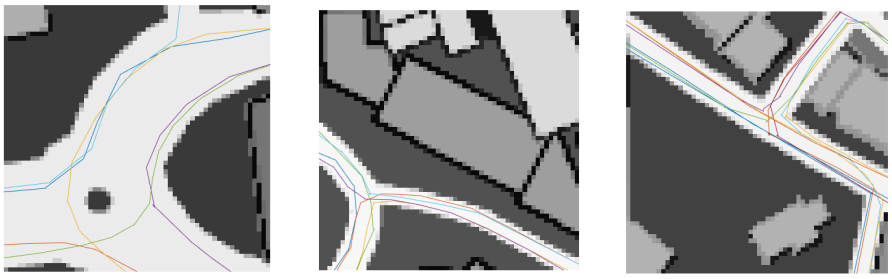


Figure 6.6: Examples of real-world GoogleMap road layouts with overlapping synthetic paths (coloured lines). Both training and test sets contain roundabouts and multi-modal intersections. All layouts are different and extracted from the Oxford city centre in the UK.

Both training and test sets contain roundabouts and multi-modal intersections with roads of different widths. Paths are manually drawn on these layouts (Figure 6.6) as a succession of connected segments. They follow the left hand side

driving rules of the UK and connect to whichever feasible next segment present at intersections and roundabouts. Similarly to the previous section, circular objects move along the layouts and are observed by a range sensor situated just outside of the grid which is of size 60×60. We perform ray tracing on the obstacles and produce a 1441 long laser scan with angular step of 0.25 degrees.

Our training and test sets consist of $D_{train} = 1550$ and $D_{test} = 200$ sequences of length $N = 40$ and distributed equally over the 31 and 10 layouts of the training and test set respectively.

6.3.3 Open Street Maps Layouts

Our *Open Street Maps* dataset consists in a collection of synthetic paths produced on the Open Street Maps (OSM) [Haklay and Weber, 2008] network. For consistency with the Oxford RobotCar Dataset[1] we consider the fraction of OSM covering the central Oxford area as shown in Figure 6.7.

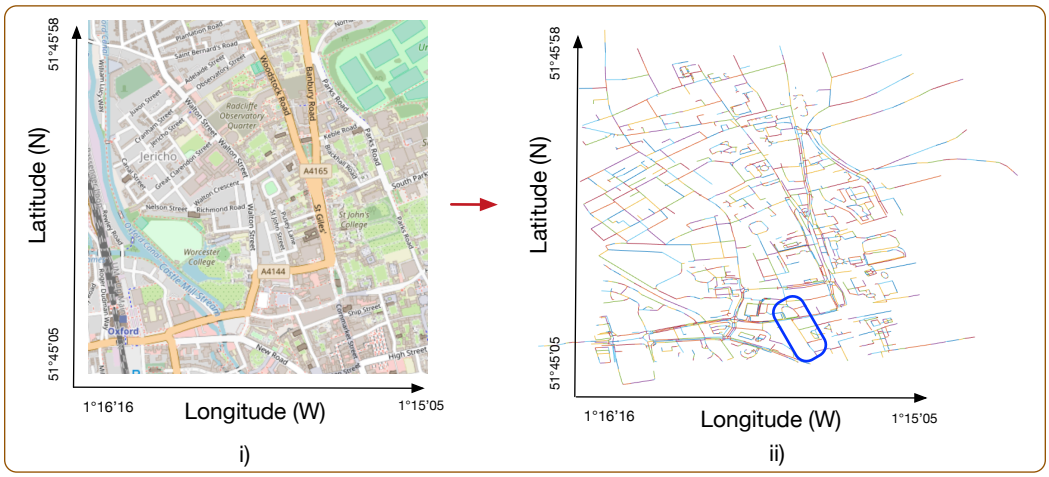


Figure 6.7: Open Street Map layout of Central Oxford (i), and corresponding road network (ii). The coloured segments (right) represent main roads and service paths, bicycle lanes and foot paths, which are connected at node points. OSM provides segment information such as length, type, direction and number of lanes. The dark blue rectangle highlights an area for which we provide an example layout and network structure in Figure 6.8.

From the OSM html file[2] we extract segment information such as road length, direction of travel, number of lanes, and road type. We choose to only retain main

[1] `http://robotcar-dataset.robots.ox.ac.uk/`
[2] `https://www.openstreetmap.org`

roads and service lanes accessible by vehicles. For a given segment of the extracted map which we refer to as the *reference* segment, we compose the surrounding network of roads. These are represented as green segments in the blue box to the left of Figure 6.8. The represented layout corresponds to the blue rectangle of Figure 6.7 ii). The start of the segment is represented with a filled blue circle, as visible on the zoomed view to the right of Figure 6.8.

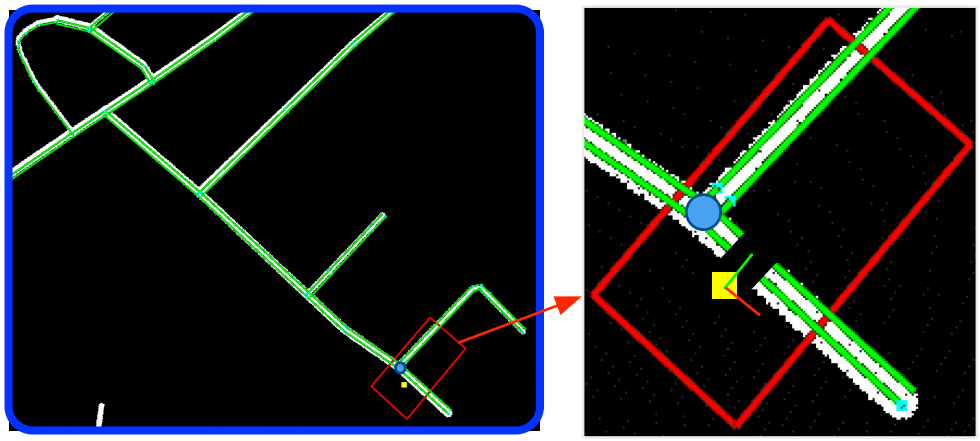


Figure 6.8: Example of a local road network extracted from OSM, and centered on a chosen reference segment (left). We build a local grid (red rectangle) around a reference frame (yellow square) located close to the beginning of the reference segment (filled blue circle) as shown in the right image of the figure.

Road segments that are bi-directional are represented with two green segments/lanes of opposite direction of travel, one-way roads with one lane with appropriate direction of travel. Lanes are shifted 1m to the left of the OSM segment coordinates along the direction of travel to allow spatial separation of lanes within the road width. We use MapBox[3] to create a black and white rendering of the OSM layout where only drivable roads are represented. This layout is overlapped to the road network and visible in white underneath the road segments of Figure6.8.

Data generation To produce the data, we place a reference frame in a random position offset from the road and close to the beginning of the reference segment. The orientation in yaw of the reference frame is randomly selected between $+/-30°$ of the segment bearing. This reference frame is visible on the yellow square of

[3] `https://www.mapbox.com`

Figure 6.8, with forward axis X in green and right axis Y in red. The red rectangle represents a local grid centered on the reference frame which extends from $[-10 : 30]$ meters along the X axis, and $[-10 : 10]$ meters along the Y axis.

Objects are produced as circles moving along the lane segments and observed by a range sensor positioned at the coordinate frame just described. Only the part of the world within the local grid is retained for our sequences.

An example view of observation by the range sensor is shown in Figure 6.9.

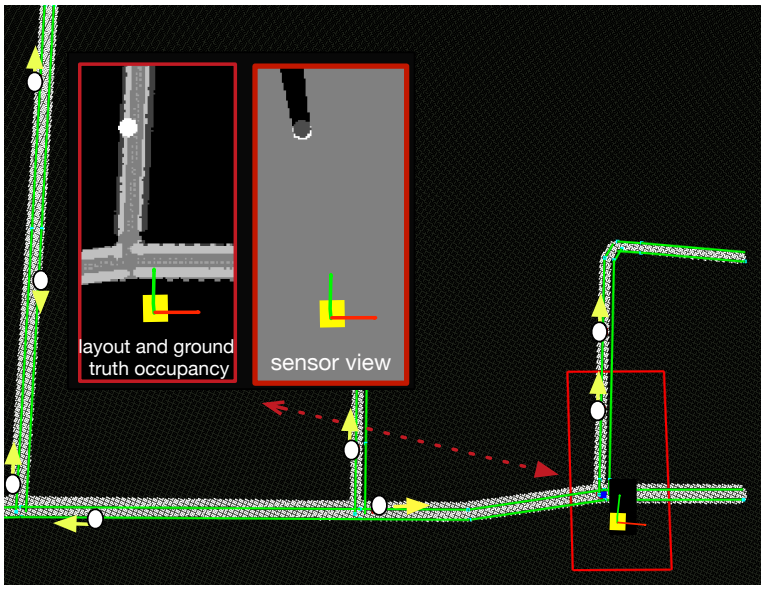


Figure 6.9: Synthetic objects (white circles) move along the road network according to the UK left-hand side rules. Direction of travel is represented with yellow arrows. The range sensor is positioned at the coordinate frame and observes only the part of the world located within the rectangular grid.

The ground truth occupancy and view from the sensor frame are shown in the two left rectangular frames. In the ground truth view we show the underlying OSM road map in dark gray for visualisation purposes. The lanes and lane separation represented in full and dotted light grey lines respectively, is the layout chosen for context input to the network. As per our previous conventions, visible occupied space is shown in white, free space in gray, and unknown occluded space in black. When objects approach an intersection they randomly turn onto one of the connected segments. To enable smoother turns the objects "cut corners" when turning onto a new segment. If there is no connected segment (dead end), the object performs a u-turn and returns on the opposite lane.

Dataset we produce 5 batches of $N = 40$ frame sequences for $D_{tot} = 1670$ segments according to the previously described procedure which represents about 4km of different road layouts. For every segment we save both the MapBox layout and road segment layout. We perform an 80/20 split for training and test set which results in $D_{train} = 6684$ training sequences and $D_{test} = 1671$ test sequences. We consider different segments for the training and test sets.

6.4 Experimental Evaluation

In the following section, we present and discuss a number of experiments conducted to address a few key questions with respect to our proposed solution: can the baseline handle tracking according to layout? can the proposed dynamic filters provide an effective solution to predicting motion according to a given layout? can the proposed solution deal with multimodal intersections and real world layouts without explicit lane separation? We also explicit the training methodology followed across experiments.

6.4.1 Methodology

In this section we clarify our data split and sequence length choices, our training procedure, and we outline the different models compared in the results section.

Input data

Training consists in dropping out inputs regularly within each sequence, to force the network to learn to predict into the future. For both the *RoadLayout* and *GoogleStreetMaps* datasets, for every 20 frames of each sequence, we show the 3 first frames and dropout the 17 next frames. We hence have two periods of dropout per sequence of $N = 40$. The choice of 3/17 split (rather than 10/10) reflects our wish to test the network's ability to predict far into the future according to the road layout. We found that reducing the initial shown input to three frames was sufficient to provide initial context, and that seventeen frames provided a sufficiently long predictive horizon to test for objects following the underlying layout and crossing

at intersections. We wish to test the ability of the network to predict into the future according to an underlying layout. We do not however want to penalise the network for not being able to predict objects coming into the frame during periods of dropout as these are random events. To focus on the former objective, new objects are only generated during the first 3 frames resulting in most objects only entering the grid during the first visible input frames. Timesteps $t = 4 - 20$ and $t = 24 - 40$ therefore represent complete prediction.

For the *Open Street Maps* dataset, as segments are smaller, we show the 3 first frames and black out the 7 subsequent frames. We hence have four periods of blackout per sequence of $N = 40$. As objects can enter the frame at any given timestep, when we black out the frames we keep an outer edge of 5 pixels visible for the network to know that objects are entering. This prevents penalising the network for not being able to predict the unpredictable and is coherent with our objective to investigate if the model is able to predict motion according to layout.

Training schedule

At every epoch of training, the training set indices are randomly shuffled and the model is trained on mini-batches of 1 sequence. We use the Adagrad optimizer with an initial learning rate of 0.01. We train models until convergence whilst monitoring the loss on the test set, and performing early stopping in the event of overfitting on the training set.

Evaluation

A threshold value of 0.5 is chosen to determine if a cell is predicted to be occupied or empty and to evaluate the F1 performance on the D_{test} sequences of the test set for all models compared.

Models

For all experiments on the *RoadLayouts* and *GoogleStreet Maps* datasets we consider a two-layer hidden state of 9 GRUs each where spatial convolutions have kernel size of 5×5. We found this overall receptive field of 9×9 for the second layer of

filters was sufficient to capture the local layout for conditioning dynamic filters. For the *OpenStreetMaps* dataset we consider a three layer hidden state as we found results improved with this increased capacity. The entire hidden state is decoded to the output with a spatial convolution of kernel size 7×7.

Experiments aim to verify that the proposed solution is able to capture scene layout and adapt its predictions accordingly. We therefore consider the following models:

- *Baseline*: our baseline model which features traditional convolutions learned at training time and fixed at test time. This model does not have access to the underlying road layout.

- *Dynamic Biases*: at every sequence timestep t, the layout I_{lay} is incorporated as individual local biases into the hidden state via the convolutional operations of the GRU module r_t, f_t and $\bar{\mathbf{h}}_t$ (see Equations 6.2).

- *Static Biases*: this model features individual static biases incorporated similarly to the *Dynamic Biases* model though they are not conditioned on the layout. Rather, they are learned at training time and fixed at test time, similarly to the static memory explored in Chapter 4.

- *Dynamic Kernels*: at every sequence timestep t, the layout I_{lay} is incorporated as pixel-wise convolutional kernel weights for updating the hidden state $\bar{\mathbf{h}}_t$. The two forget ($\mathbf{f}_t$) and reset ($\mathbf{r}_t$) gates are computed using traditional convolutions on the stacked input $\mathbf{x}_t$ and hidden state $\mathbf{h}_{t-1}$ tensors as presented in the GRU equations 6.2. There are no dynamic or static biases in this model.

Dynamic Biases and *Dynamic Kernels* are referred to as *informed* models through the text. During training, we observed that the dynamic filter weights of the *Dynamic Kernels* model reached one order of magnitude higher values than those of the traditional filters of the network. We therefore initialise the weights of the dynamic filters to 1/50 of that of the other weights of the network to reduce this effect. We regroup the different model characteristics in Table 6.2.

Model	Local Biases	Local Kernels
Baseline	none	none
Static Biases	static	none
Dynamic Biases	dynamic	none
Dynamic Kernels	none	dynamic on $\bar{\mathbf{h}}_t$

Table 6.2: Characteristics of the convolutional biases and kernels for the four models compared in their ability to track according to scene layout.

6.4.2 Tracking according to uni-modal road layouts

We first seek to answer the following question: can the proposed dynamic filters predict motion according to a given layout compared to the baseline architecture? We consider the *RoadLayouts* dataset as it features clear tracks with unimodal intersections, and we demonstrate that our proposed solution is able to turn accordingly at intersections.

Baseline model

To justify the proposed extended architecture, we first assess the *Baseline* model's limitations and observe qualitatively in Figure 6.10 two sequences of predictions on two different layouts which contain a turn. For clarity of analysis, these sequences only contain **one** object moving through the scene and we observe a selection of timesteps within the a timestep window. They are produced similarly to the sequences of the test set. We then quantitatively evaluate our baseline model and compare performance to that of the informed models.

For each layout, we represent the sequence in four ways. From bottom to top we show: 1) the input occupancy and visibility grid as seen by the network. It is only shown during the first 3 timesteps, and then dropped for 17 timesteps; 2) the raw output occupancy prediction from the *Baseline* model; 3) the same output compared to the target occupancy as true positives (green), false positives (blue), and false negatives (red); 4) the underlying ground truth occupancy with overlapping layout for visualisation purposes.

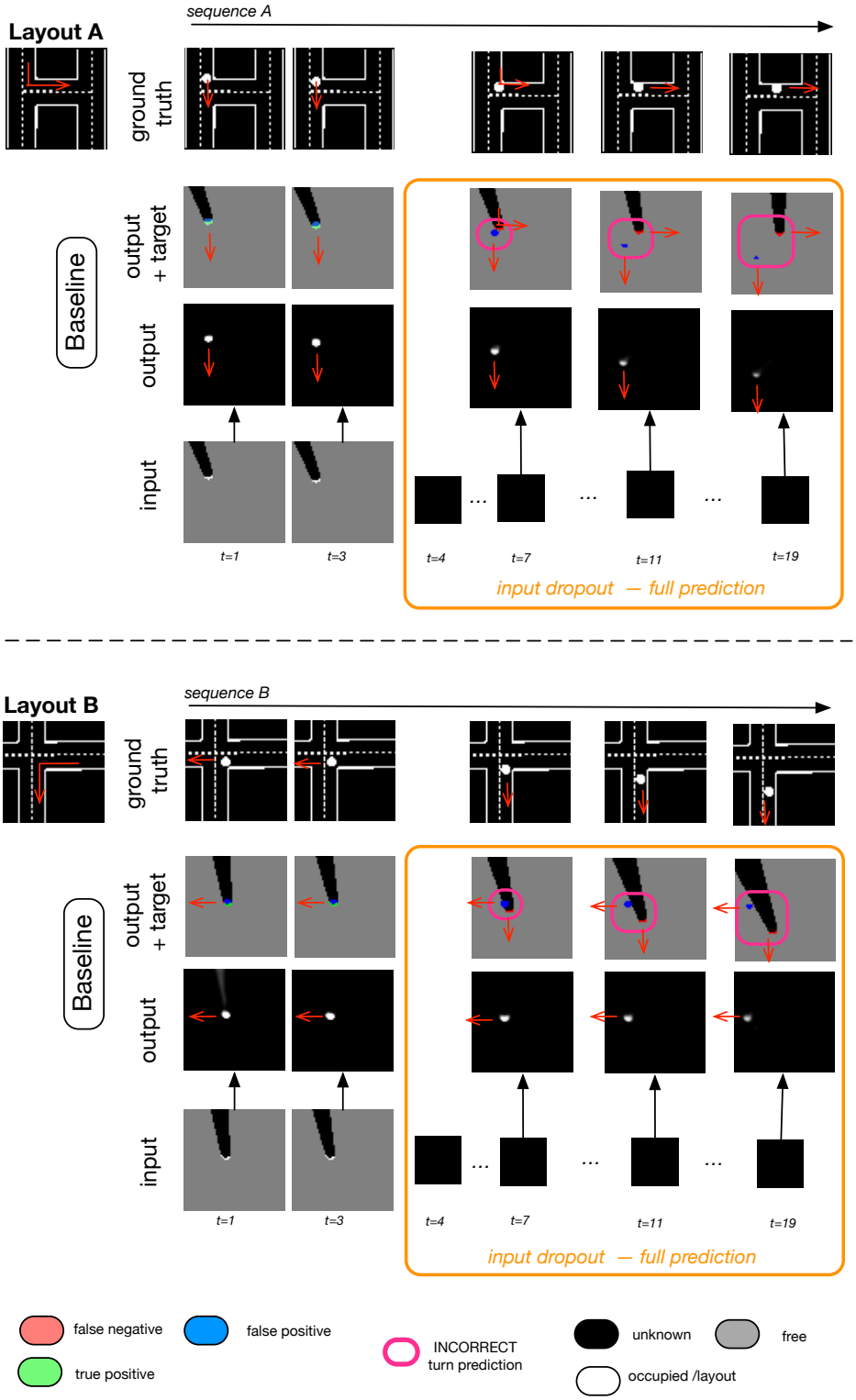


Figure 6.10: Failure of the *Baseline* model to predict the turn at an intersection on two different layouts. In both scenarios the model predicts incorrect linear motion across the intersection as indicated by the red false negative predictions and highlighted by the magenta contours. Predictions also weaken over time as indicated by the shrinking false positive predictions in blue. We highlight the motion of both ground truth and predicted output with red arrows for clearer visualisation. Best viewed on pdf.

To visualise the entire colour-coded output prediction, a threshold of 0 rather than 0.5 is chosen here. False positives are measured with respect to the target occupancy (i.e visible part of the ground truth). Consequently, any false positive prediction located adjacent to either a false negative or true positive prediction is a good thing, as it represents the model hallucinating occupancy in occlusion. Inversely, any false positive predicted far away from a false negative or true positive prediction is incorrect. We indicate the direction of travel of the object with red arrows to aid visualisation.

The model is able to predict linear motion. It is however unable to predict the left turn at the corners of both layouts as highlighted by the magenta contours and discrepancy increases with the predictive time horizon. False positives here represent incorrect predictions rather than positive hallucinations in occlusion. The model is only able to learn to predict straight lines which is due to the fact that turns only represent a small fraction of the entire data. The model can therefore achieve best performance by at least learning to predict linear motion. We however observe weakening predictions over time. We suggest this is due to the fact that because the model does not have access to the layout and that layouts are not always straight lines, it is forced to produce weak predictions in order to incur the least cost during training.

Dynamic Filters

We now observe the performance of the two informed models *Dynamic Biases* and *Dynamic Kernels* on the same task of predicting turns at intersections. We first observe a single object per sequence, and then show performance on multi-object sequences from the test set.

Single Object Figure 6.11 compares the performance of all three models on the same Layout A sequence as that shown at the top of Figure 6.10 in the previous section. We represent a selection of frames from the 20 timestep sequence where $t = 4 - 20$ represents full prediction as the input is dropped out. Where the *Baseline* model is unable to predict the turn as previously observed and as highlighted in

magenta contours (i), the two informed models (ii, iii) accurately predict the left turn according to the layout (green contours).

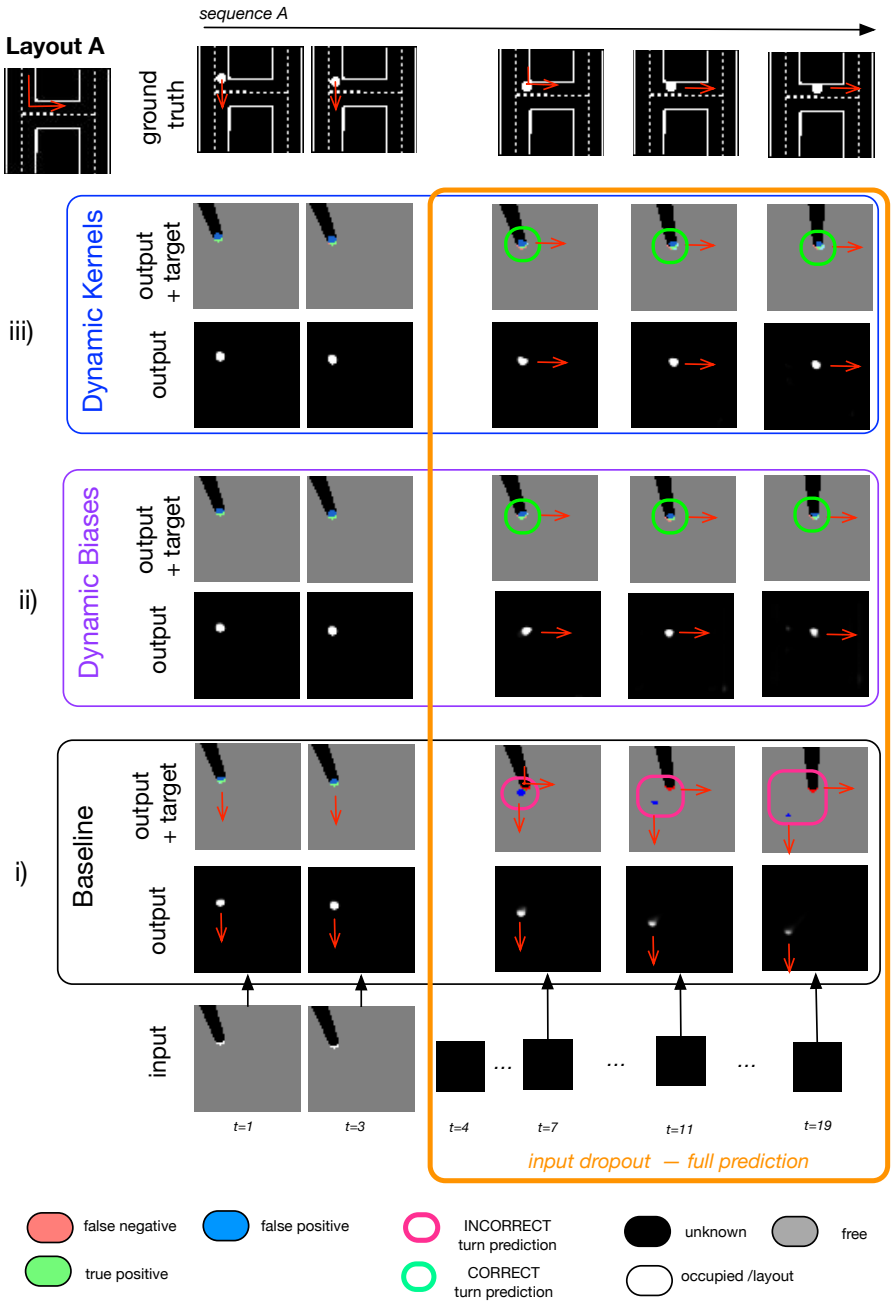


Figure 6.11: Sequence prediction on a layout presenting turns for a selection of models. The *Baseline* model (i) is unable to predict the turn and predicts linear motion throughout as highlighted in the pink squares. On the contrary, the two models which have access to the underlying layout (ii) and (iii) are able to confidently predict the turn according to the layout as highlighted in the green squares.

We further compare all three models on a simple layout which does not contain any turns in Figure 6.12. This is to compare performance on a task that the baseline model is able to learn.

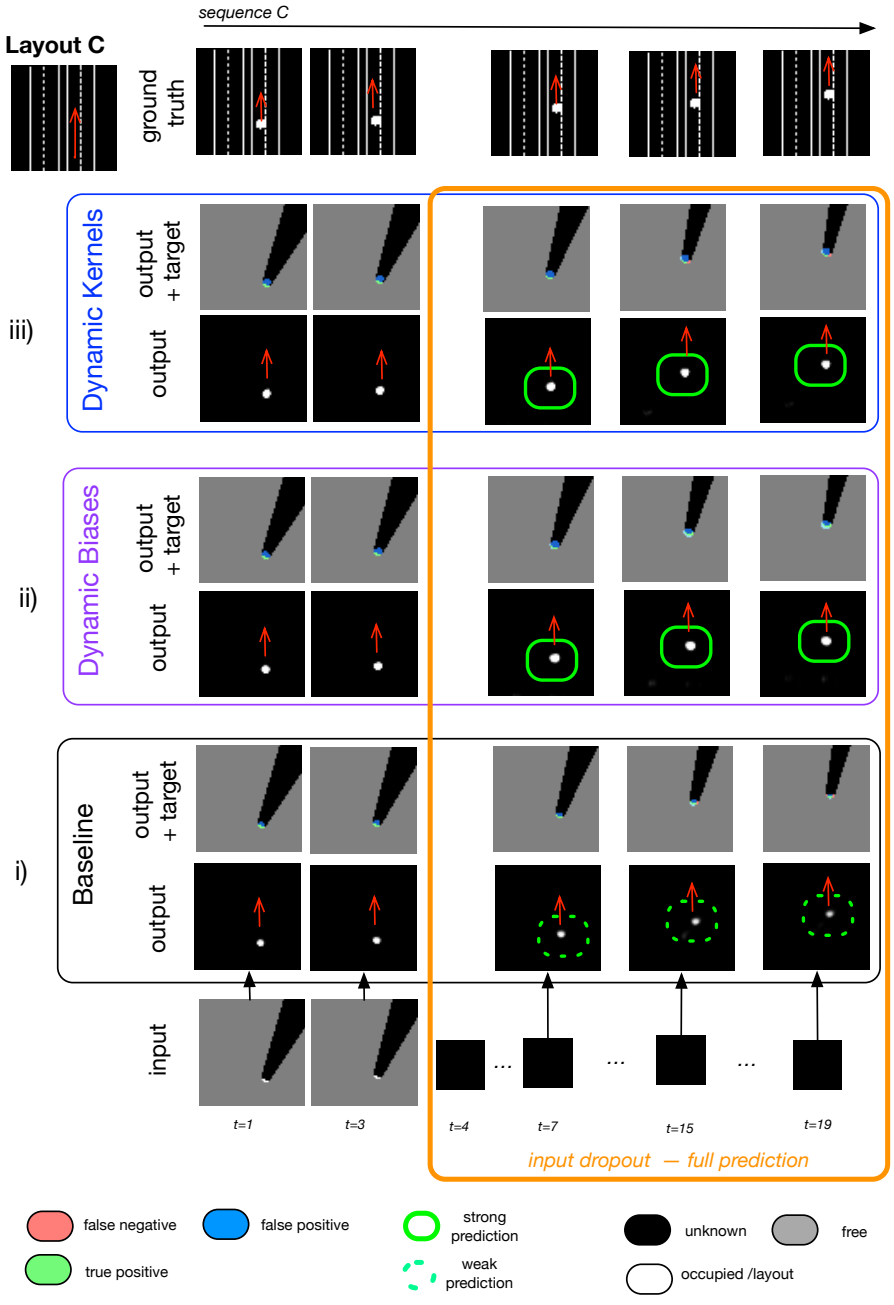


Figure 6.12: Sequence prediction on a simple layout containing two straight road layouts. The *Baseline* model (i) achieves good performance as there are no turns involved. Predictions are however weaker compared to the two informed models which have access to the underlying layout (ii) and (iii).

All models accurately predict the linear motion into the future. In comparison with the two informed models however, the *Baseline* model predictions are weaker as highlighted with the dotted green contours towards the end of the sequence. Similarly to what was observed in Figure 6.10 we hypothesise that the *Baseline* model is forced to produce less confident predictions as it does not have access to the underlying layout.

Multiple Objects We further compare both informed models on test set sequences which contain multiple objects. Figure 6.13 shows the performance of *Dynamic Biases*. The model accurately predicts turns at intersections and produces overall strong signal output across the layouts. The only false negative observed (lay ii) corresponds to an object coming into the grid just as the input is dropped out, which is too late for the model to notice.

Figure 6.14 shows the performance of the *Dynamic Kernels* model on the same selection of sequences. Similarly to *Dynamic Biases*, the model accurately predicts turns. One exception is that highlighted on layer iii. at timesteps $t = 17 : 19$. As the object approaches the turn it slowly disappears. We also observe some weak predictions towards the end of sequences iii. and iv. As the model performs otherwise correctly we suggest that in view of the early overfitting of the model, better weight regularisation could enable the network to learn more consistently. We do not observe this limitation in our other experiments where the data is more complex and less overfitting is observed.

Quantitative Analysis

We conduct three runs for each of the four models and stop training early to prevent overfitting. Results are collected in Figure 6.15. Timesteps $t = 4 - 20$ represent complete prediction into the future as the input is dropped out. All model descriptions are detailed in Table 6.2. We introduce the *Static Biases* model to compare to *Dynamic Biases*. This is to verify that it is the *conditioning* of the individual biases on the layout and not the individual biases themselves that account for the *Dynamic Biases* model's ability to track according to the layout.

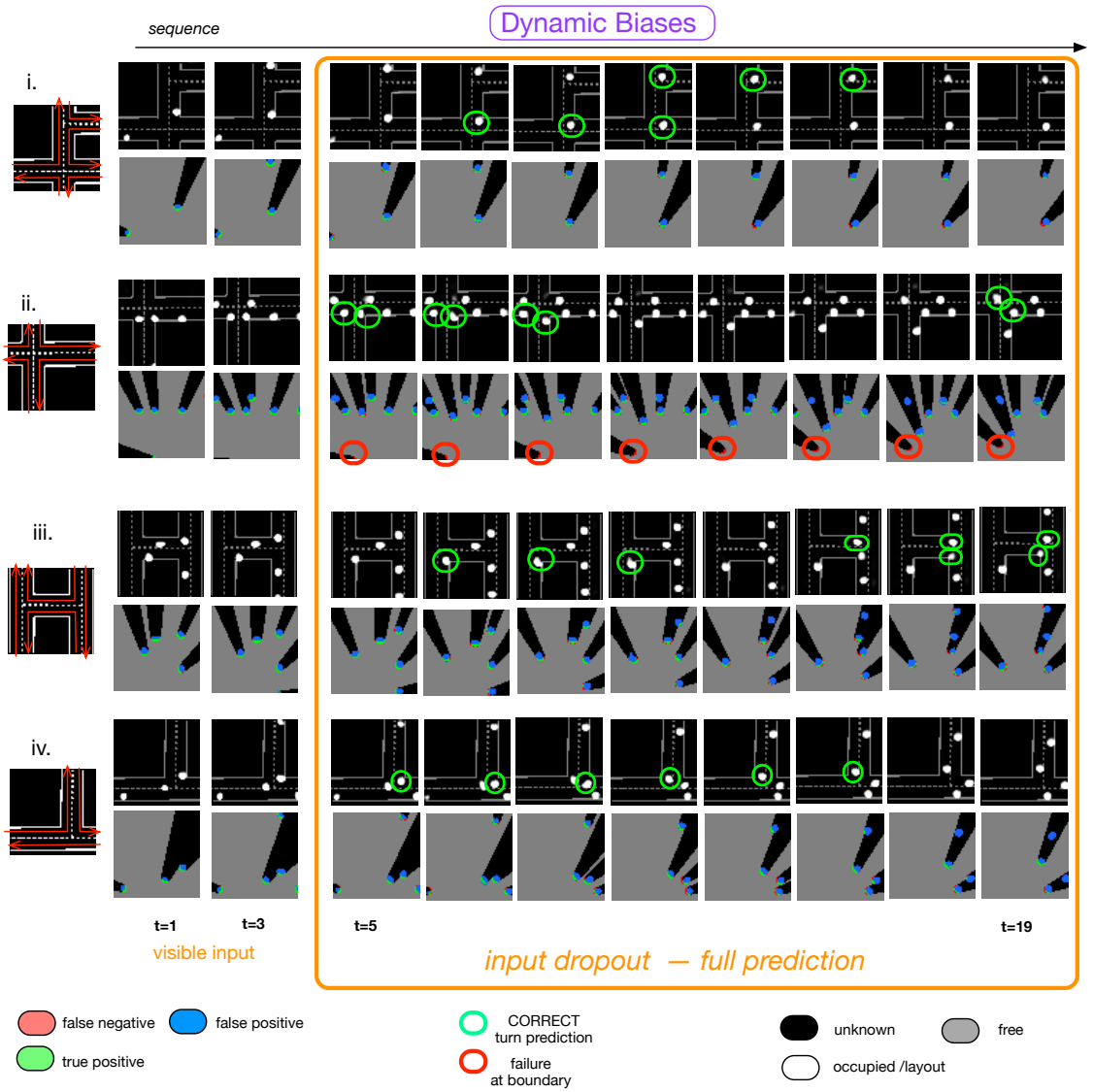


Figure 6.13: Selection of sequences from the Test set and *DynamicBiases* model predictions. The model is able to confidently predict turns on the test layouts. The only false negative is observed for layer (ii) and is due to an object barely coming in just as the input is dropped out which makes it challenging for the network to notice.

Each model run achieves commensurate performance as observed in the mean F1 measure to the right of the figure. We highlight the best runs for each model with corresponding colour-coded contours.

Figure 6.16 compares the mean predictive F1 measure of the best four model runs.

The *Dynamic Biases* model achieves best F1 performance as observed in Figure 6.16 (Green curve). The performance of *Dynamic Kernels* is nearly commensurate to that of our best model *Dynamic Biases*. As observed qualitatively,

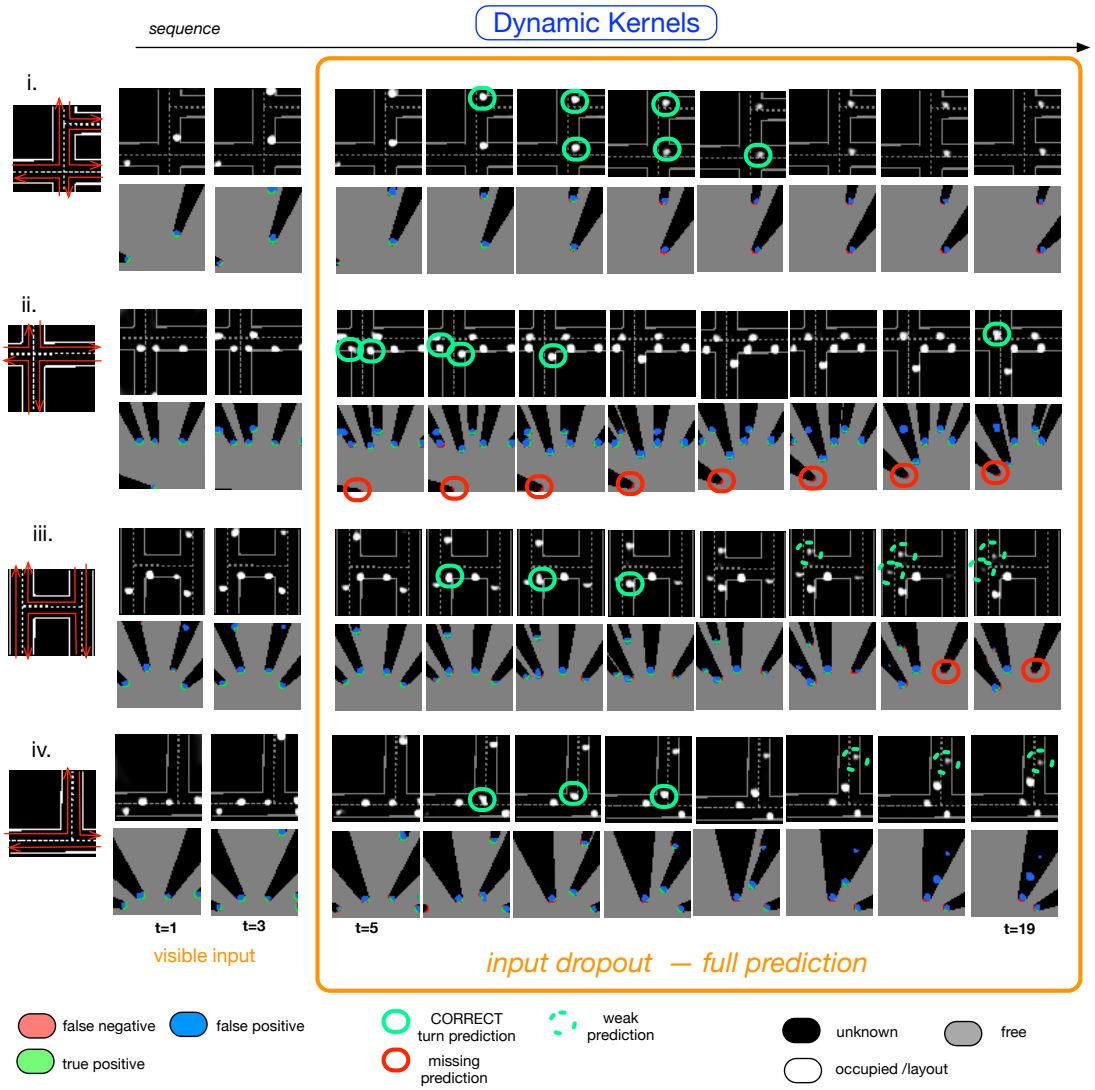


Figure 6.14: Selection of sequences from the Test set and *Dynamic Kernels* model predictions. The model is able to confidently predict turns on the test layouts to the exception of timestep $t = 17 - 19$ for layout iii. The false negatives observed on layout (ii) are due to the same challenging sequence as observed in Figure 6.13. The object enters just as the input is dropped out making it challenging for the network to notice. At the end of the sequence on layout (iii) the model misses the left turn of an object and shows some weak predictions towards the end of sequences on layouts (iii) and (iv). We suggest these mixed results are due to the observed overfitting of the model early into training, requiring early stopping. As we do not observe similar issues when training the model on the other more complex datasets, we suggest that further weight regularisation could aid model performance. We further discuss this in the text.

we hypothesise that better performance could be achieved with additional weight regularisation on the *Dynamic Kernels* model.

We explain the surprisingly high F1 measure of the *Baseline model* by the fact

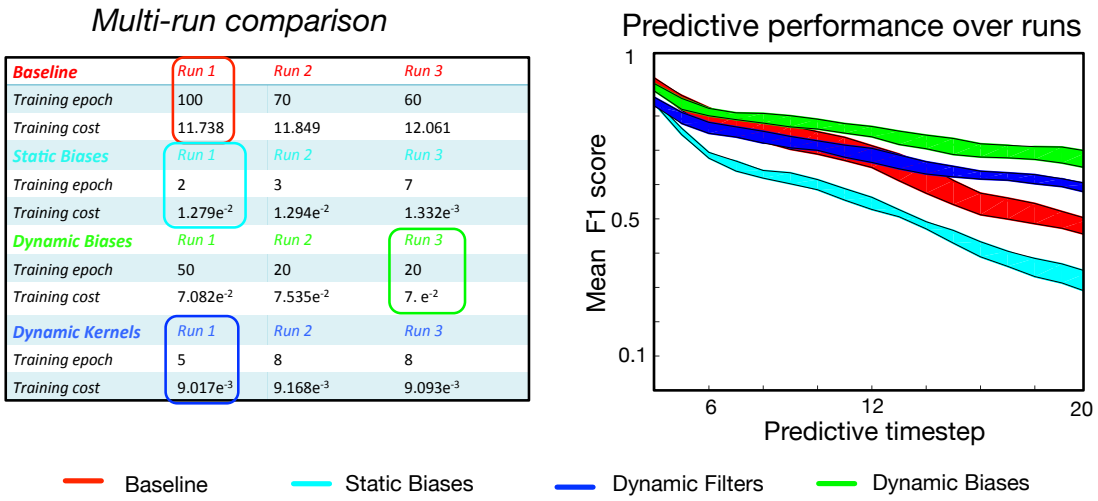


Figure 6.15: Predictive performance of multi-runs for the four models compared. The training epochs and costs at end of training are reported in the left table, with the best runs circled in colour. The corresponding mean F1 performance (averaged over test set sequences) for all three runs of each model is plotted to the right in the relevant colour. The shaded regions represent the minimum/maximum span of the mean F1 performance over the three runs for each model.

that the dataset is relatively benign, with fewer turns compared to straight lines in all layouts. This results in an F1 measure being much influenced by performance on straight lines, which the baseline model does well. This is reinforced by the wider performance difference when considering only Layout B which contains numerous turns, to the right of Figure 6.16.

Finally, we find that *Static Biases* achieves worst performance of all, below that of the *Baseline* model. This model contains static biases which are learned at training time and fixed at test time. We argue that at training time, the model can only learn to track according to the layout by memorising the layouts in its static biases. At test time, it tries to fit the perceived motion to one of the learned layouts which fails to be helpful as test layouts are different. This is coherent with the qualitative evaluation of Figure 6.17 which compares the *Static Biases*'s ability to predict scene dynamics on one layout from the **training** set (Layout Z), and one layout from the **test** set (Layout B).

The **test** sequence layout(top half) presents a turn that the model completely fails to capture as highlighted with the pink circles. Inversely, the model predicts

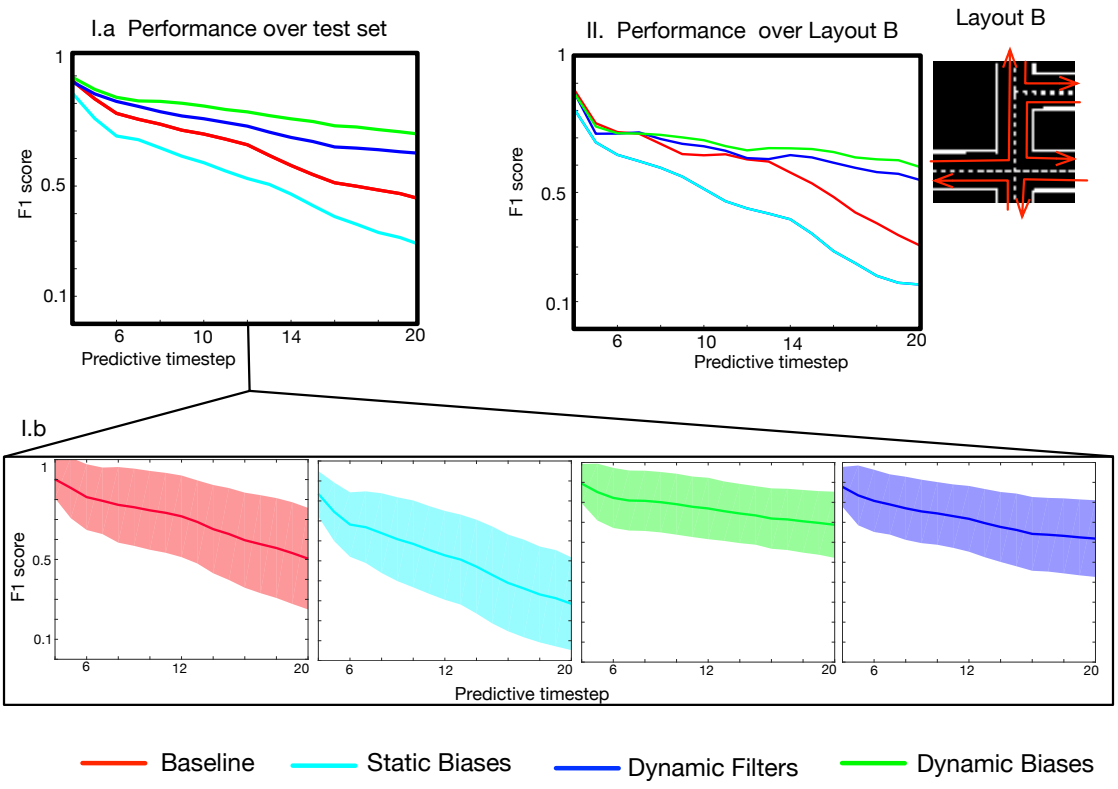


Figure 6.16: Mean F1 measure (averaged over test set sequences) for a variety of models on predicting scene occupancy given a layout. The plots correspond to the performance of the best model runs circled in the left table of Figure 6.15. *Dynamic Biases* and *Dynamic Kernels* which respectively condition local biases and filters on the layout achieve highest performance (Green and Dark Blue curves). The *Baseline* model which does not incorporate layout information (Red) achieves reasonable prediction ability due to the fact that there are only few turns in the sequences, but it only learns a linear model and cannot account for the layout turns. The model which fixed static biases (*Static Biases*, Cyan) overfits to the training and does poorly on new unseen layouts. The performance difference between informed and uninformed models is greater when considering Layout B with contains multiple turns (II, right). Informed models present lower variance across sequences (I.b).

perfectly the turns of Layout Z of the **training** set (bottom half). Notice the difference in strength of the output prediction on both training/test set compared to that of the *Baseline* model which also highlights the overfitting confidence. Figure 6.18 compares the F1 performance measures for the **test** set sequence (Blue curve), and for the **training** set sequence (Red curve) and showing clear overfitting to the training set.

We find that only the informed models are able to predict the scene occupancy robustly according to layouts featuring uni-modal intersections. The *Baseline*

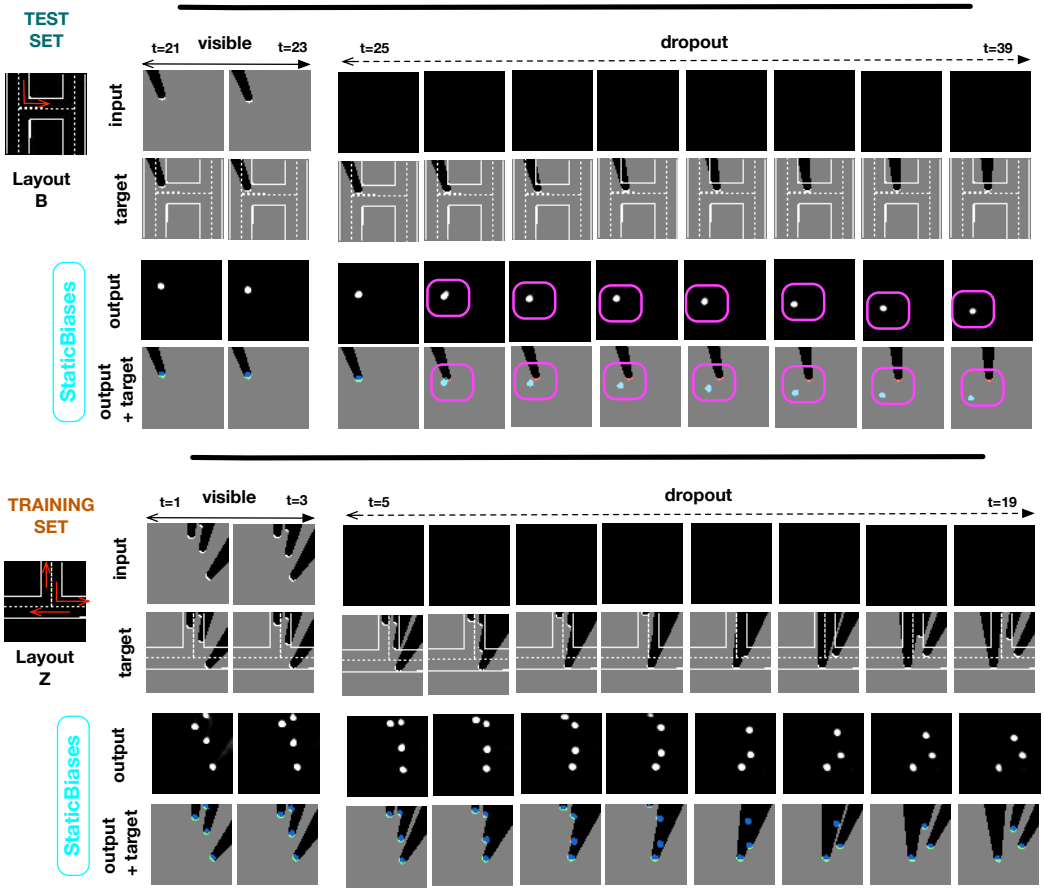


Figure 6.17: Output predictions of the *Static Biases* model on a sequence from the Test set (top) and Training set (bottom). The model fails to predict the turn accurately on the Test set sequence but predicts the training sequence flawlessly suggesting strong overfitting on the training set.

model fails in this task and a static memory is not able to generalise predictions to new unseen layouts. We find that both dynamic kernels and biases are able to conditioning network dynamics on the layout is an effective way and that dynamic kernels produce particularly crisp outputs. We provide an intuition as to why that may be in the discuss section, and further investigate both solutions on real-world layouts featuring multimodal intersections.

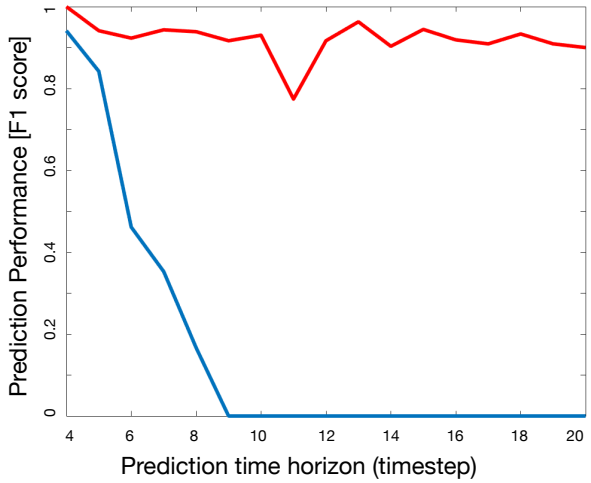


Figure 6.18: F1 measure for **Test** set layout B (Blue curve) and **Training** set layout Z (Red curve) corresponding to sequence of Figure 6.17. The model is unable to predict the turn in layout B showing clear overfitting to the training set.

6.4.3 Tracking according to real world multimodal layouts

In the previous experiment we showed that incorporating dynamic filters under the form of biases or kernels was an effective means of extending the network's ability to track according to synthetic layouts featuring uni-modal intersections. In this experiment we aim to investigate if our proposed solution is able to address multi-modal intersections on real world layouts which do not feature lane separation and straight angles. We focus on comparing our two informed models to the baseline: *Baseline*, *Dynamic Biases*, and *Dynamic Kernels*.

Quantitative Evaluation We conduct three runs for each model and perform early stopping to prevent overfitting. Performance results are collected in Figure 6.19. All models show decreased performance as the prediction horizon increases which is expected. Both informed models (Blue and Green) achieve commensurate performance and suggest clear improvement over the *Baseline* model (Red). The performance increase is larger than in our synthetic dataset. We suggest this is due to the fact that the synthetic layouts presented few turns compared to many straight roads as well as single option intersections which made it easier overall to predict (higher overall F1) and easier for the *Baseline* model to do well with linear motion. On the Google Maps layouts which present natural curvy roads, the *Baseline* model completely fails at predicting far into the future (F1 of nearly 0 at timestep 20). We circle in relevant coloured contours the best runs of each model, though performance is commensurate across runs.

We find it takes respectively four times and ten times as many iterations for the *Dynamic Biases* and *Baseline* models to train compared to the *Dynamic Kernels* model, with no added performance value. This suggests that dynamic kernels are an efficient way of extracting and utilising the layout.

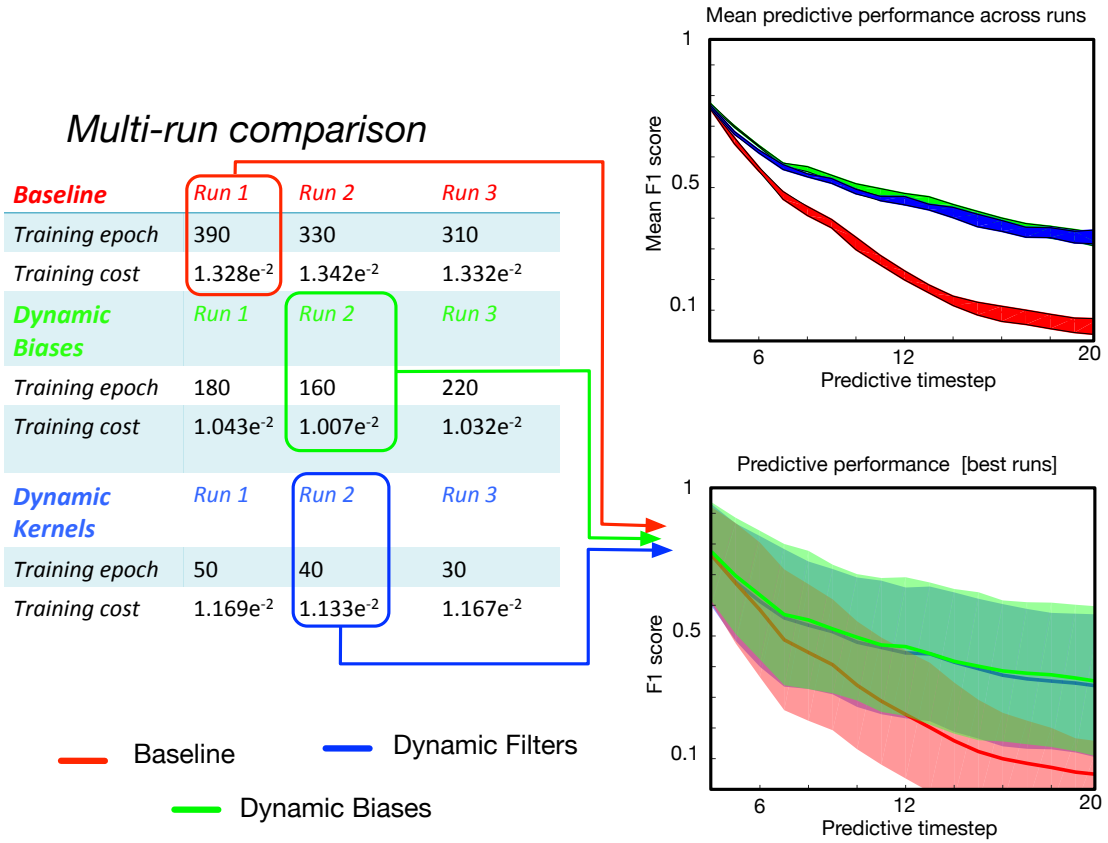


Figure 6.19: Predictive performance of multi-runs for the three models compared. The training epochs and costs at end of training are reported in the left table. The corresponding mean F1 performance (averaged over test set sequences) for all three runs of each model is plotted to the top right in the corresponding colour. The shaded regions represent the minimum/maximum span of the mean F1 performance over the three runs for each model. The best runs are circled in the table in the corresponding colour, and their F1 performance mean and standard deviations represented bottom right. All models show decreasing ability to predict into the future with timesteps which is expected. The model which does not incorporate layout information (Red) is unable to predict into the future. Both informed models achieve commensurate performance and clear improvement over the baseline. Multi-modal intersections contribute to the observed variance.

Qualitative Evaluation Figures 6.20 and 6.21 compare the models' ability to predict into the future on two relatively simple layouts. LayoutA contains a single road slightly curving towards the bottom of the frame. On this simple layout, the *Baseline* model has difficulty producing a clean and accurate prediction in the last 4 frames shown, and objects. In comparison, the two informed models cleanly predict the two objects crossing as highlighted in the green contours.

Layout B present a T-junction where only the informed models are able to predict a pertinent turn. The *Baseline* model predicts a linear trajectory at the

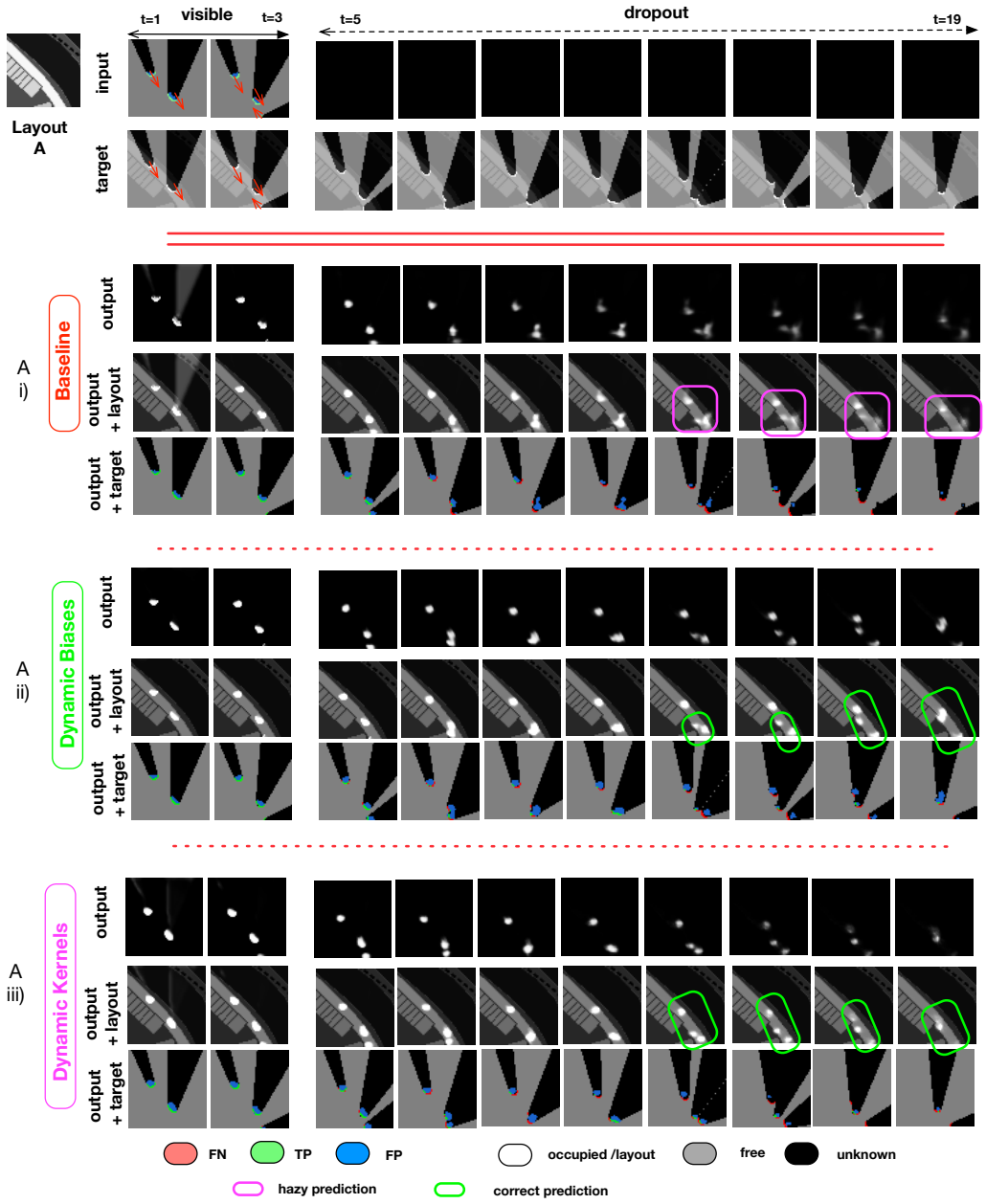


Figure 6.20: Example sequence on an easy one-road layout for our selection of models. The red arrows show motion direction for the objects visible at timesteps 1-3. The *Baseline* model achieves poor performance in the last few frames of the sequence whereas informed models are able to correctly predict the objects crossing on the road. Best viewed on pdf.

intersection (pink circles). Notably, the *Dynamic Kernel* model pertinently predicts the two possible directions the object can take at the intersection (highlighted in full green contours).

Layout C in Figure 6.22 presents a more challenging roundabout layout. Once

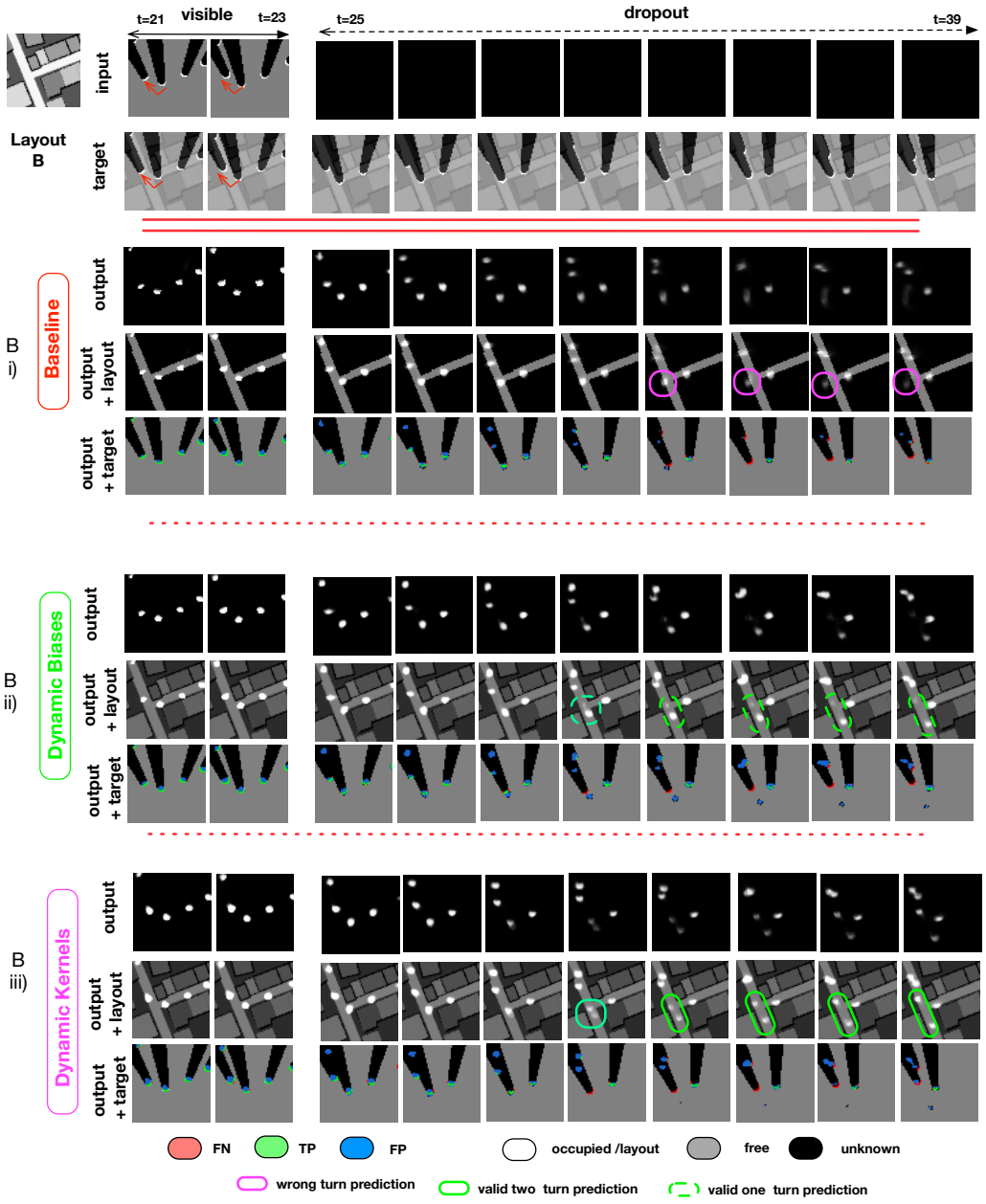


Figure 6.21: Example sequence on a T-junction GoogleMap layout for our selection of models. As expected, the *Baseline* model fails to predict any turn at the intersection and predicts a weak linear motion as highlighted in the pink circle. Conversely, the informed models are able to correctly predict a multi-modal turn at the intersection as highlighted in full green circles. The *Dynamic Kernels* particularly does well at predicting both possible turns at the intersection. Best viewed on pdf.

again the *Baseline* model does poorly and predicts weak, evanescent tracks. Quite remarkably, the *Dynamic Kernels* model is able to predict a correct bimodal path for the object located at the roundabout (green circles in the pink box). The object

is both predicted to exit and to carry on around the roundabout clockwise. The *Dynamic Biases* model also show signs of multi-modal paths at the roundabout, though with less confidence, and incorrectly predicts circling both clockwise and anti-clockwise around the roundabout centre (highlighted in dotted green circles in the green rectangle). A close-up of the green, blue and pink boxes is provided in Figure 6.23 for better visualisation.

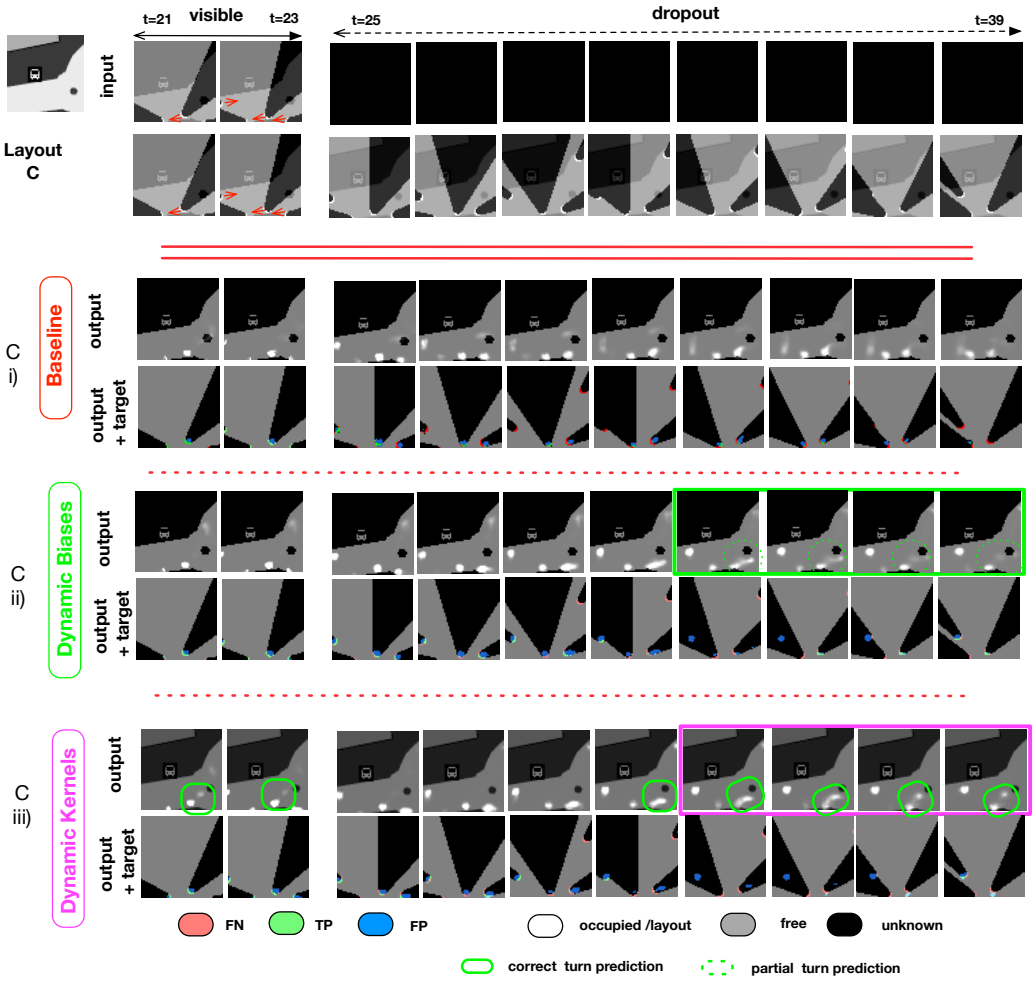


Figure 6.22: Example sequence at a roundabout GoogleMap layout for our selection of models. The *Baseline* model fails to predict consistant and clear occupancy. The informed models are able to predict paths according to the layout, and achieve mixed performance on the roundabout. The *Dynamic Kernels* model actually produces an accurate multi-modal prediction at the roundabout (green full contours). The *Dynamic Biases* model predicts a less clear prediction around the roundabout. We provide a zoom in to the green and magenta boxes in Figure 6.23. Best viewed on pdf.

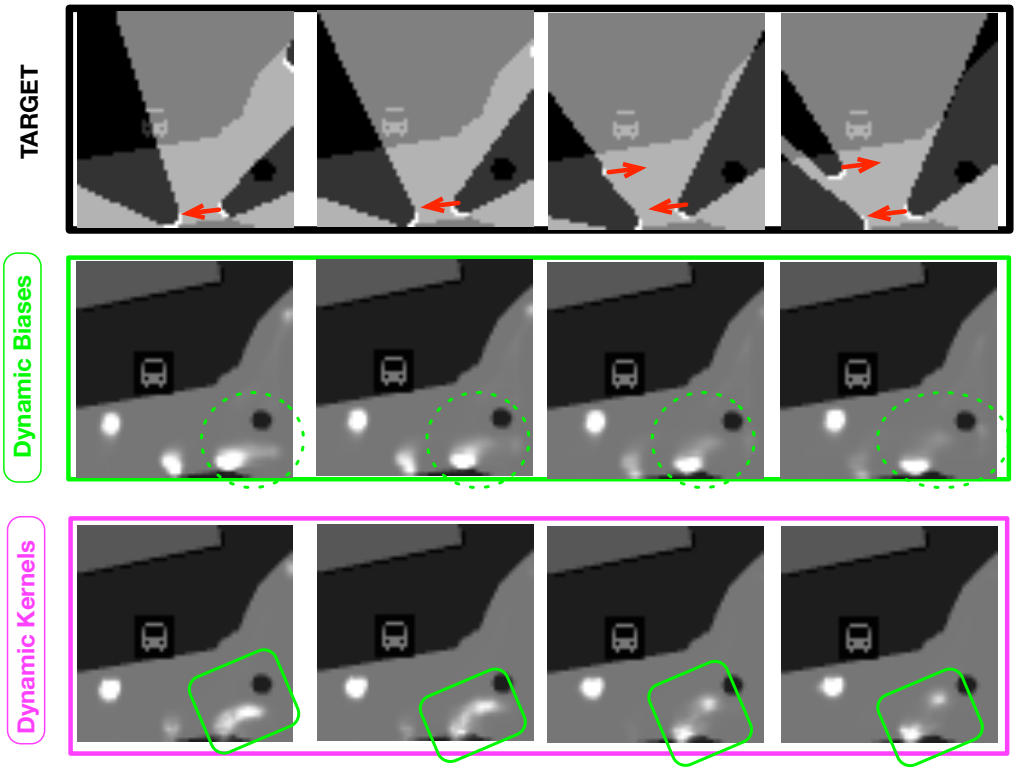


Figure 6.23: Close-in on the roundabout prediction for two informed models from the sequence shown in Figure 6.22. The *Dynamic Biases* model (Green box) predicts weak multi paths around the centre of the roundabout which is partly incorrect as objects should circle the roundabout from the left only (highlighted with the dotted green circles). The *Dynamic Kernels* model however accurately predicts the two possible paths at the roundabout as highlighted by the full green rectangles. Best viewed on pdf.

6.4.4 Discussion

We find that both *Dynamic Kernels* and *Dynamic Biases* are able to capture underlying scene layout and predict dynamics according to simple road layouts compared to the original framework.

The *Dynamic Kernels* model however appears to better handle multi-modal intersections of real-world layouts. It overall converges quicker and produces more "crisp" output making it a promising mechanism for context extraction as we further observe in the next section. We hypothesise that this may be due to the fact that the dynamic kernels exercise a more direct influence on the hidden state update compared to the dynamic biases. *Dynamic Biases* provide an offset to the convolutional operation, and are able to move the response above or below the sigmoid threshold. The convolutional kernels remain however generically learned

and not specific to the layout. On the other hand, the dynamic kernels are produced specifically for the layout at hand, and directly provide a new candidate hidden state. This mechanism appears more "pro-active" than the dynamic biases, and more directly comparable to traditional trackers updating object position according to the chosen motion model. This could explain the somewhat cleaner results obtained. We also find it is a more elegant and efficient (in terms of scalability) approach to incorporating context.

We did not find improved performance when combining both dynamic biases and kernels for context integration. Rather, we suggest that better performance could be achieved by increasing the receptive field at the input to increase context understanding, investigate including the hidden state in the conditioning input alongside the road layout, and train on a bigger dataset such as the one produce on GoogleMaps layouts.

6.5 Towards Real-World Tracking in Context

In this final section, we present our idea of bridging tracking in context to real world applicability and present preliminary results on learning a prior of vehicle motion from surrounding road layout for a moving vehicle in an urban environment. Can we obtain a prediction similar to that depicted in our opening Figure 6.1? We consider our *Open Street Maps* dataset and learn to track along the segment layouts similarly to previous experiments. We then consider a car driving in the Oxford city centre equipped with a GPS system to localise in the overhead OSM map. We test our learned model on the surrounding road segment and qualitatively visualise predictions in the vehicle forward facing camera image. We show that the model is able to predict a vehicle turning at an intersection, as seen from the vehicle frame.

We first visualise qualitatively the network prediction on a test frame from a vehicle driving in Oxford cite centre, and then present our network performance on the OSM dataset.

6.5.1 Motion priors onboard a vehicle

To visualise predictions in a real world setting, we consider GPS coordinates collected in the area of Jericho, Oxford, from a NISSAN LEAF vehicle equipped with a BumblebeeXB3 camera and NOVATEL GPS. From the collected GPS coordinates we localise in OSM using the Hidden Markov Model approach of Newson and Krumm [2009]. Figure 6.24 represents a fraction of GPS traces localised in the overhead OSM network.

Once we register the GPS tracks in the OSM, we project GPS traces onto the matched OSM segment and produce a coordinate frame at the location of the robot frame facing as illustrated in Figure 6.25. From this reference frame we produce synthetic paths according to the underlying road network similarly to the data synthesis presented in the previous section.

We produce sequences of $N = 40$ for 450 GPS positions and test our *Dynamic Biases* model pretrained on the OSM dataset. We select a test point where we observe a plausible prediction and project the tracker output in the forward-facing

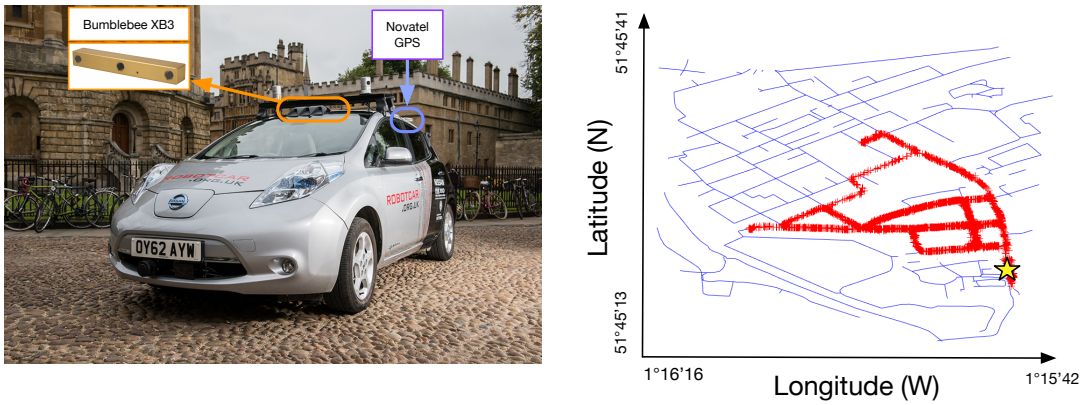


Figure 6.24: GPS tracks (red asterix) localised on the road segments of OSM with map matching Newson and Krumm [2009] and collected from a NISSAN LEAF vehicle in the area of Jericho, Oxford, UK. Only drivable road segments are represented (no walkways). The yellow star represents the example location shown in Figure 6.25.

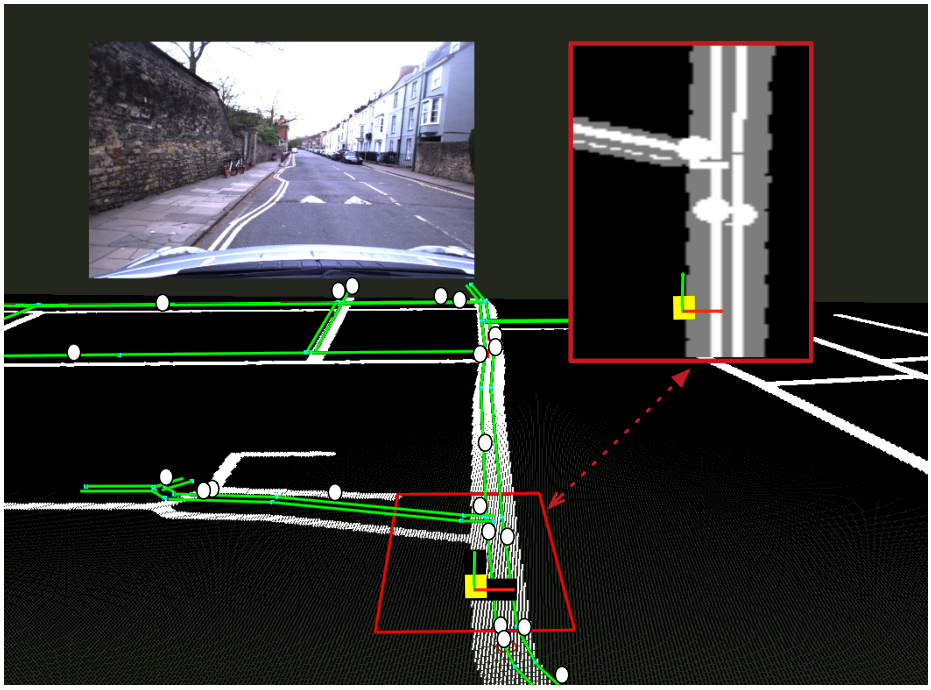


Figure 6.25: Registration in OSM and data generation in the vehicle frame. The red box represents the network grid. The robot frame is located on the yellow square and objects move along the OSM network. The corresponding location in the OSM map of Figure 6.24 is shown with a yellow star.

camera. We observe the predicted tracks and projected predictions into the camera frame in Figure 6.26.

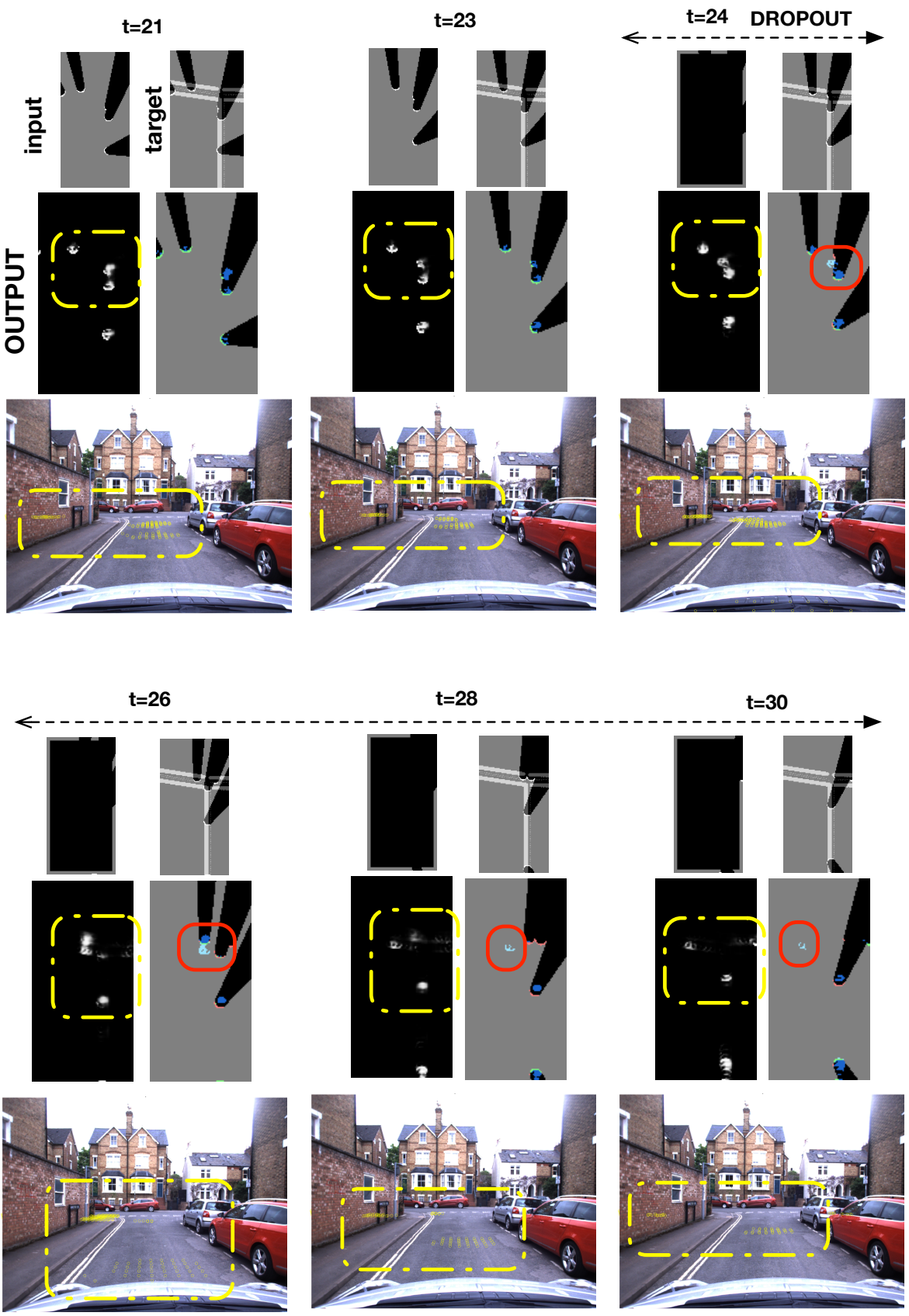


Figure 6.26: Example prediction at a point along the driven path. Objects turn to the right at the intersection but the network predicts a left-hand turn. Though this is incorrect with respect to the ground truth it constitutes a plausible outcome and correct prior. Best viewed on pdf.

In the figure, we represent a sequence of network inputs and output prediction, viewed in different ways. The top two rows of every timestep represent network input and output. From top left to bottom right we show: network input, target with overlapping layout for visualisation, raw output predictions, and output predictions colour-coded with the target output. The colour-coding remains the same as previous experiments, with red, blue and green representing false negatives, false positives, and true positives respectively. The output predictions are projected into the camera frame as pixelised yellow dots. We highlight with a dotted yellow contour the predictions in both network output grid and camera imagery to aid interpretation and registration between the two views. Timesteps $t = 24 : 30$ correspond to input dropout. The output therefore represents complete prediction by the network.

We find that the network succesfully predicts a left turn at the intersection(red circle) for the object moving along the forward facing X axis. Though this left turn is incorrect compared to the actual ground truth, it represents a plausible output with respect to the underlying layout.

In a real-world deployment one could use more accurate cm-precision registration in overhead maps for systematic accurate registration in the robot frame. Alternatively, a robot could localise in a semantic map under frameworks such as Experience-Based Navigation [Churchill and Newman, 2013]. The tracking ouput predictions could then provide a context-relevant prior of object dynamics directly into the robot frame.

6.5.2 Quantitative Evaluation

Quantitatively we compare the predictive ability of our three models on the OSM dataset with the predictive F1 mean in Figure 6.27. As only the $t = 1 : 3$ first inputs are shown to the network timesteps $t = 4 - 10$ represent network prediction within the 5 pixel edge of the grid. As previously observed, the informed models achieve better performance than the *Baseline* model.

The initial F1 measure when the input is first occluded (timestep 4) is very low for all models. This is due to the F1 measure not accounting for True Negatives

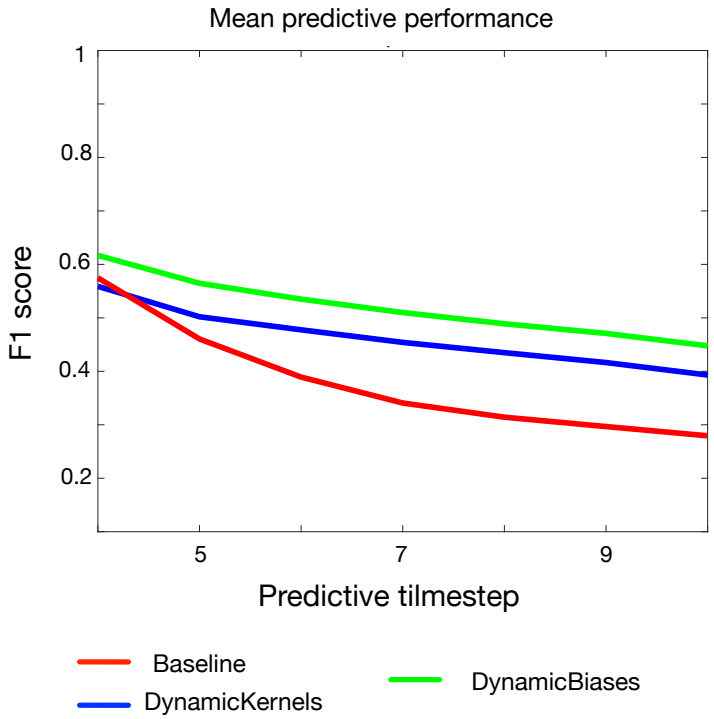


Figure 6.27: Mean F1 score (averaged over test set sequences) for the informed models compared to the *Baseline* when predicting motion according to OSM layouts in Oxford city centre.

(correctly predicting that no object is present in the scene). If no object is present in the scene, as is the case for some parts of our sequences, the F1 measure will produce 0 even if the network correctly predicts that no object is present. This results in degrading the overall F1 metric. As this effect occurs for all models in the same way and as we are interested in model performance comparison we do not find this to be a limiting factor for this study.

Qualitative Figure 6.28 shows an example sequence where the layout presents an intersection. The Baseline model is unable to predict an accurate path at the intersection. The *Dynamic Biases* model on the other hand is able to produce a multi-modal path corresponding to the intersection layout and produces overall stronger predictions than the *Baseline*. The *Dynamic Kernels* predicts only one of the two possible modes.

Figure 6.29 shows more disappointing results of the *DynamicBiases* model on a

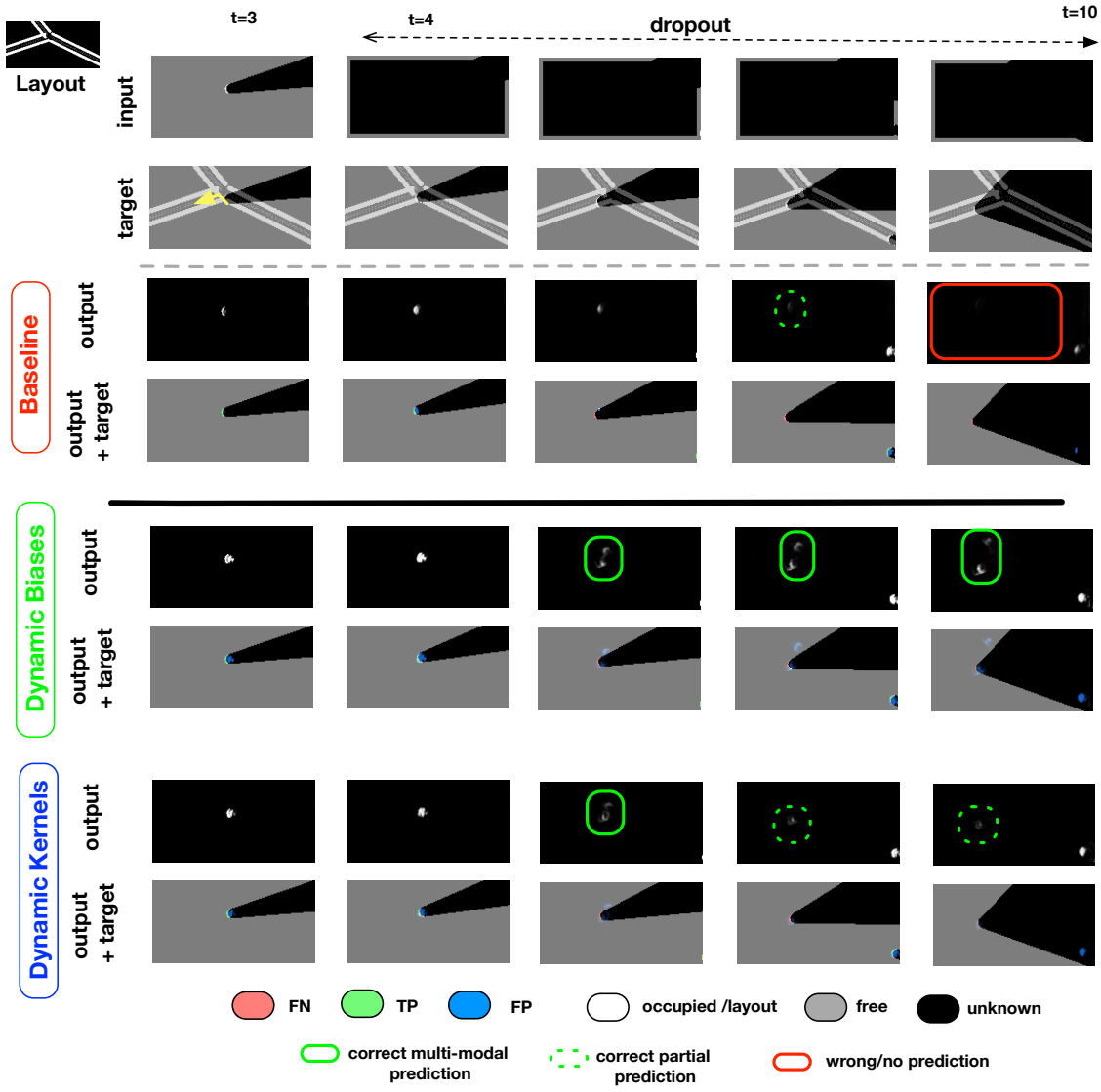


Figure 6.28: Example sequence of prediction by the *Baseline* and *DynamicBiases* models. The yellow arrow in the target sequence indicates that the object turns left at the intersection. At the intersection, the *DynamicBiases* model is able to predict a multi-modal path which is relevant with the underlying T-junction layout. The Baseline model on the other hand is unable to predict the intersection accurately, and overall produces a weaker prediction.

selection of sequences containing an intersection. Although the model produces hints of multi-modal prediction, these are very weak. The model tends to predict one path coherent with the intersection layout, which can be a turn. We hypothesise that the data could be improved with more objects (to increase the signal during training), a weighted objective, and with more intersections. We leave this for future work.

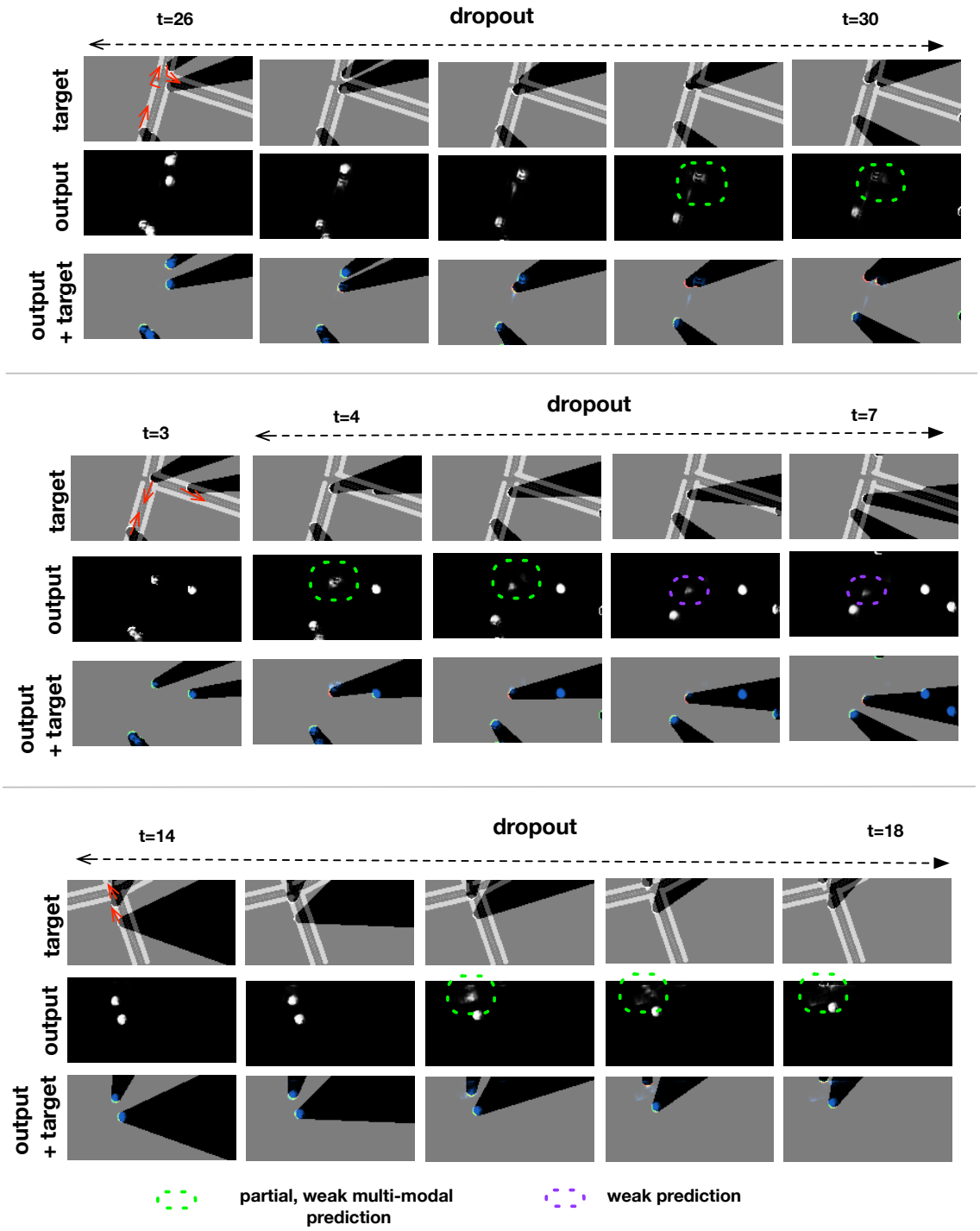


Figure 6.29: Example sequence of prediction by the *DynamicBiases* model at intersections. The red arrows in the target sequence indicate which way objects are moving. Although the model shows hints of multi-modal prediction, they are very weak. We hypothesise that the training set could be enhanced to incorporate more intersections and more objects for more signal.

6.6 Conclusion and Future Work

In this chapter we considered the limitations of a framework tracking objects out of context. We proposed dynamic filters as a mechanism for adapting network dynamics to an underlying scene in a self-supervised way. We find that the proposed solution is an effective means of adapting motion models to road layout on synthetic data, particularly when incorporating dynamic kernels with appropriate receptive field and weight regularisation. Experiments on OSM layouts showed mixed results, which we suggest would be improved with smoother paths, and greater proportion of objects at intersections. We also suggest that better context could be captured by learning a compact encoded representation of the layout, which in combination with a coordinate grid could be input to the network filters. This would provide wider context, and extend applicability to learning interactions and capturing distant context beyond the receptive field of the convolutional filters.

Future work could further explore tracking in context using onboard sensors for extracting scene layout. This would remove the need for localising in a map, and in combination with the proposed extension of tracking from a moving platform in Chapter 5, would enable learning in context from collected real-world data.

-

7 Conclusion and Future Work

Robots will need to be equipped with systems able to build high level interpretations of the data provided by theirs sensors and move away from *instantaneous* sensing, towards *predictive* sensing. We summarise our contributions and suggest several avenues of future work building on our findings.

7.1 Summary of contributions

In this thesis we explored the use of latent models for a localisation and perception system to perform practical inference in the context of field robotics, with the aim of increasing robot situational awareness. Our contributions are as follows:

1. **Predicting localisation performance by learning an appearance-based model of the localisation system** (Chapter 3): we addressed the limitations of a localisation system providing instantaneous localisation information and motivated the importance of developing a robot's capacity to predict *a priori* its ability to localise in a given environment. In the context of Teach and Repeat we proposed a novel appearance-based approach to estimating the likely localisation envelope around a taught trajectory. We leveraged the effectiveness of Gaussian Processes for learning a model of the localisation system in a given environment, and demonstrated that the proposed solution

is able to predict localisation envelopes for similar yet geographically distant, new trajectories in the real world.

2. **Improving a Perception System's Situational Awareness by Learning Real World State Estimators** (Chapter 4): we addressed the limitations of a perception system relying on occluded sensor data and motivated the importance of developing frameworks able to learn rich state representations of real world environments, from experience. We motivated a number of architectural changes to the *Deep Tracking* [Ondruska and Posner, 2016] framework and deployed the proposed solution to predicting unoccluded future scene occupancy in a real world urban setting. We demonstrated the effectiveness of recurrent neural networks for learning to track dynamic scenes through occlusion in a self-supervised manner, and compared favourably to a state-of-the art model-free tracking solution. We also showed how the rich state learned on the task of tracking could be used to learn the additional task of semantically labelling the scene with little additional data, using Transfer of Knowledge.

3. **Extending State Estimation to a Moving Vehicle for Practical Real-World Deployment**(Chapter 5): we acknowledged the limitation of the deep tracking framework developed for static platforms and motivated the need for deep methods to be able to operate on moving platforms. We propose what we believe is a novel use of deep networks for state estimation from a moving source and adapt the estimated state update to the local robot frame with the use of ego-motion estimates. We demonstrated the effectiveness of our proposed solution on synthetic data, and showed improved performance when tracking from a moving vehicle in an urban city centre. We also showed that the extended architecture is able to differentiate between static and dynamic obstacles.

4. **Improving Place-Specific State Estimation by Incorporating Scene Context** (Chapter 6): we considered the limitations of the tracking framework to observing objects outside of scene context, and proposed an extended solution for adapting motion models to the scene layout without the need for annotated ground truth. We demonstrated on synthetic data that the use of dynamic filters could be an effective means of tayloring motion models to the underlying scene layout and suggested promising preliminary results linking usability to real world applications by localising in overhead maps.

7.2 Future Work

There are many avenues to be explored in the areas of predictive sensing for system and world state estimation in field robotics. First and foremost it would be interesting to combine both localisation corridors and future state occupancy predictions as cost maps input to a planner operating in space and time. This would combine the three pillars of robotics in a common predictive framework. We also suggest future directions of research related to individual aspects of the work presented in this thesis below.

Predicting Localisation Performance for a Range of Environmental Conditions

Chapter 3 considered appearance features for generalising localisation performance to new routes. Just as vision-based localisation itself, this approach relies on the matching of features which are prone to change due to environmental conditions such as location, lighting or occlusion. To alleviate this shortcoming, the method could be improved by implicitly learning a number of representations for dissimilar circumstances such as location, time or weather. This is similar in spirit to experience-based navigation as introduced in Churchill and Newman [2013] and Linegar et al. [2016b].

Tracking in Motion with Context

A number of directions could be envisaged to combine and extend the work presented in Chapters 4 to 6.

Multi-sensor context aggregation for tracking in motion A natural extension to the work presented in Chapters 5 and 6 would be to integrate context into the moving deep tracking architecture using a combination of sensor modalities. Within an experience-based navigation system [Churchill and Newman, 2013] context could take the form of prior maps in which the robot is able to localise. Alternatively, context and road layout could be extracted from live images or 3D laser data. In the case of a single forward facing camera mounted on a vehicle, imagery and robot motion information could be converted into a full 360° layout context around the robot as illustrated in Figure 7.1.

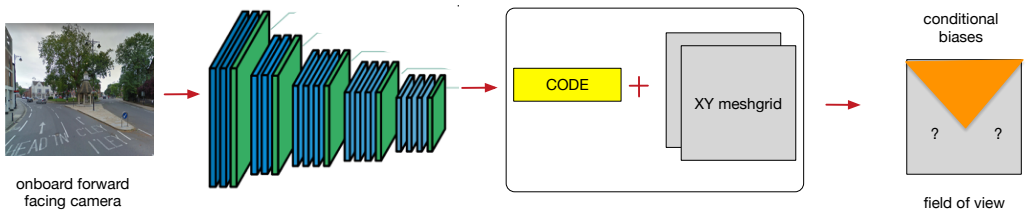


Figure 7.1: The forward facing camera could be used to generate spatial biases in the network at the corresponding birds' eye-view projection. Using ego-motion information, this layout could be extrapolated to the full 360° grid around the robot.

Alternatively, with a full camera and 3D laser scanner sensor configuration, birds eye view representations could be constructed live and used to learn and produce motion models in context as illustrated in Figure 7.2.

Finally, forward facing imagery and 3D laser scans could be composed to form a 2.5D grid for input and context, similarly to the work of [Chen et al., 2017].

Incorporating motion awareness Although our proposed extension for tracking from a moving vehicle incorporates ego-motion in the belief state update, the network does not "see" the robot ego-motion as input and cannot therefore learn how its own actions in the world might influence how the robot perceives the world

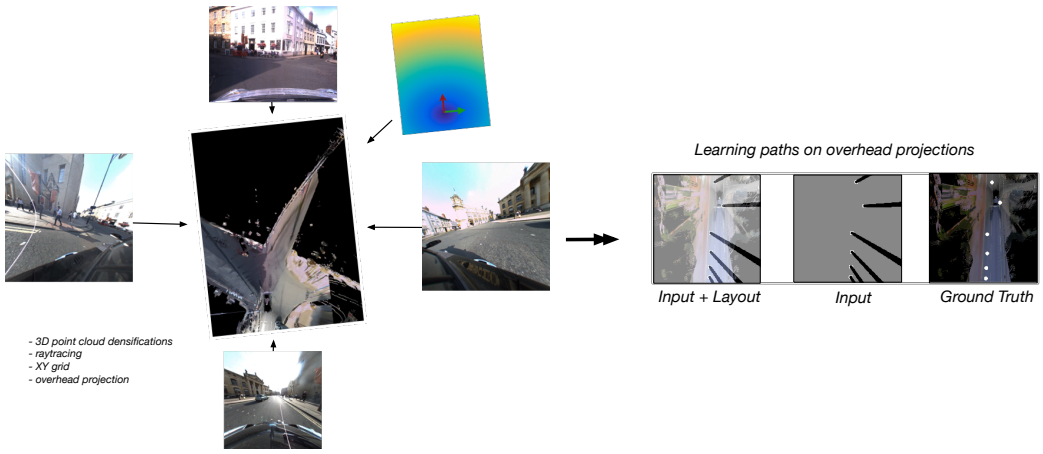


Figure 7.2: Cameras providing a 360° view could be combined with a densified 3D point cloud to form an overhead projection around the vehicle. Registering the projection with the network grid could provide context.

and how world dynamics react to the robot motion. Ego-motion could be integrated similarly to layout and condition dynamic filters for belief state update, or tracking could be incorporated within a wider planning system where the robot could learn to move and track the world in a fully end-to-end framework.

Extracting state information Though the deep tracking framework resembles traditional trackers in that it tracks and predicts future object locations, it does not explicitly provide data association, bounding boxes, or object state descriptors. Transfer of knowledge could be envisaged to extract this information.

Improving overhead birds' eye view

In the spirit of predicting perception and seeing through natural occlusion, the deep tracking framework spirit could be used to learn to produce better birds' eye viewpoints. We believe that true understanding of the world requires robots to learn by moving through their environment.

Onboard Layout Extraction Our work in Chapter 6 assumed full view of the road layout extracted from an overhead map. To avoid the need for registering in a prior map, road layout could be directly inferred from onboard forward facing camera imagery. The road layout could first be extracted using road boundary

detection [Suleymanov et al., 2018] and path proposals [Barnes et al., 2017] such as illustrated in Figure 7.3.

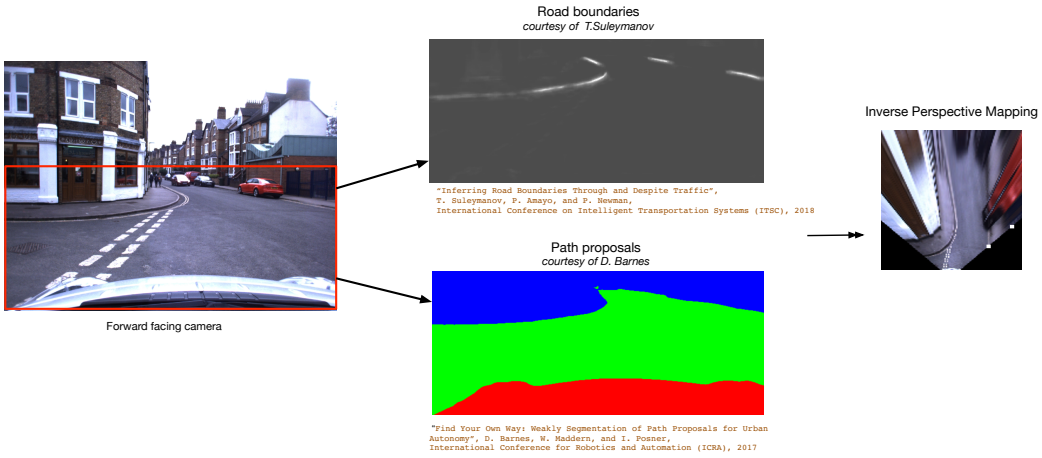
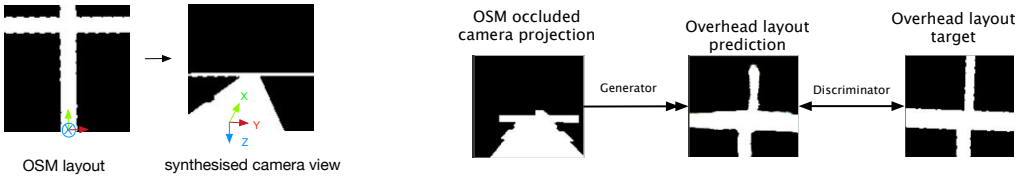


Figure 7.3: The road and road boundaries can be extracted from a forward facing camera and projected into a bird's eye view with Inverse Perspective Mapping. Courtesy images from D. Barnes related to [Barnes et al., 2017] and T.Suleymanov [Suleymanov et al., 2018].

The segmented road can then be projected into a bird's eye view using Inverse Perspective Mapping. This however produces incomplete layouts which cannot provide clear information of the upcoming intersection due to the limited view of the road bends and object clutter such as parked cars. This projection can be improved by learning an image translation from forward facing layout to overhead layout using OSM data.

OSM road layouts can be projected into the camera frame producing pairs of images from both camera and bird's eye view. An image translation network based on Conditional Generative Adversarial Networks such as Pix2Pix [Isola et al., 2017] could then be trained to produce overhead road layout from BB3 path imagery. As real world road layouts in the camera frame are not fully observed when reaching an intersection, the image translation task can be trained on partially observable camera view layouts as illustrated in Figure 7.4.



a) OSM projection into camera frame

b) Image translation from camera view to OSM layout with Pix2Pix

Figure 7.4: OSM layouts can be projected into the camera frame as if the robot were moving along the OSM road network (a). An image translation network such as Pix2Pix can then be trained to produce complete overhead layouts, from occluded OSM camera projections (b).

Finally a common representation could be learned between real image road segmentation and OSM projected frames to apply the learned translation to onboard camera data.

7.3 Concluding Remarks

In this thesis we have extended data-driven frameworks to enable robotic systems to build higher level understanding of what they are sensing and how they are performing in the field. Prediction is a major component of human intelligence, reflecting a deep understanding of the world that surrounds us. We hope that this work will be valuable for developing more capable robots able to operate in real-world environments; from autonomous cars on Earth to exploratory rovers on Mars.

Bibliography

Agarwal, A. and Suryavanshi, S. (2017). Real-Time* Multiple Object Tracking (MOT) for Autonomous Navigation. Technical report.

Arras, K. O., Mozos, O. M., and Burgard, W. (2007). Using boosted features for the detection of people in 2D range data. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3402–3407. IEEE.

Arroyo, R., Alcantarilla, P. F., Bergasa, L. M., and Romera, E. (2015). Towards life-long visual localization using an efficient matching of binary sequences from images. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 6328–6335. IEEE.

Ballan, L., Castaldo, F., Alahi, A., Palmieri, F., and Savarese, S. (2016). Knowledge transfer for scene-specific motion prediction. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9905 LNCS, pages 697–713.

Ballardini, A. L., Cattaneo, D., Fontana, S., and Sorrenti, D. G. (2017). An online probabilistic road intersection detector. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 239–246.

Barnes, D., Maddern, W., and Posner, I. (2017). Find Your Own Way: Weakly-Supervised Segmentation of Path Proposals for Urban Autonomy. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Singapore.

Bay, H., Tuytelaars, T., and Van Gool, L. (2006). Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer.

Bengio, Y. et al. (2009). Learning deep architectures for AI. *Foundations and trends® in Machine Learning*, 2(1):1–127.

Bishop, C. M. (2006). Pattern recognition and machine learning.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer.

Calonder, M., Lepetit, V., Ozuysal, M., Trzcinski, T., Strecha, C., and Fua, P. (2012). BRIEF: Computing a local binary descriptor very fast. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*.

Caruana, R. (1995). Learning many related tasks at the same time with backpropagation. *Advances in neural information processing systems*, pages 657–664.

Casas, S., Luo, W., and Urtasun, R. (2018). Intentnet: Learning to predict intention from raw sensor data. In *Conference on Robot Learning*, pages 947–956.

Chen, X., Ma, H., Wan, J., Li, B., and Xia, T. (2017). Multi-view 3d object detection network for autonomous driving. In *IEEE CVPR*, volume 1.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078, 2014*.

Choi, S., Lee, K., and Oh, S. (2016). Robust modeling and prediction in dynamic environments using recurrent flow networks. *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 1737–1742.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Churchill, W. (2012). *Experience based navigation: Theory, practice and implementation*. PhD thesis, University of Oxford.

Churchill, W. and Newman, P. (2013). Experience-based navigation for long-term localisation. *The International Journal of Robotics Research*, 32(14):1645–1661.

Churchill, W., Tong, C. H., Gurău, C., Posner, I., and Newman, P. (2015). Know your limits: Embedding localiser performance models in teach and repeat maps. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Cummins, M. and Newman, P. (2011). Appearance-only slam at large scale with fab-map 2.0. *The International Journal of Robotics Research (IJRR)*.

Dahl, G. E., Yu, D., Deng, L., and Acero, A. (2012). Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):30–42.

Dayoub, F. and Duckett, T. (2008). An adaptive appearance-based map for long-term topological localization of mobile robots. In *Intelligent Robots and Systems (IROS). IEEE/RSJ International Conference on*.

De Brabandere, B., Jia, X., Tuytelaars, T., and Van Gool, L. (2016). Dynamic filter networks.

Dequaire, J., Ondrúška, P., Rao, D., Wang, D., and Posner, I. (2018). Deep tracking in the wild: End-to-end tracking using recurrent neural networks. *The International Journal of Robotics Research*, 37(4-5):492–512.

Dequaire, J., Tong, C. H., Churchill, W., and Posner, I. (2016). Off the beaten track: Predicting localisation performance in visual teach and repeat. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 795–800. IEEE.

Djuric, N., Radosavljevic, V., Cui, H., Nguyen, T., Chou, F.-C., Lin, T.-H., and Schneider, J. (2018). Motion prediction of traffic actors for autonomous driving using deep convolutional networks. *arXiv preprint arXiv:1808.05819*.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

Ehrhardt, S., Monszpart, A., Vedaldi, A., and Mitra, N. (2017). Learning to represent mechanics via long-term extrapolation and interpolation. *arXiv preprint arXiv:1706.02179*.

Ekman, M., Kok, P., and de Lange, F. P. (2017). Time-compressed preplay of anticipated events in human primary visual cortex. *Nature Communications*, 8:15276.

Fuchs, F. B., Groth, O., Kosoriek, A. R., Bewley, A., Wulfmeier, M., Vedaldi, A., and Posner, I. (2018). Neural stethoscopes: Unifying analytic, auxiliary and adversarial network probing. *arXiv preprint arXiv:1806.05502*.

Furgale, P. and Barfoot, T. D. (2010). Visual teach and repeat for long-range rover autonomy. *Journal of Field Robotics (JFR)*.

Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059.

Geiger, A., Lauer, M., and Urtasun, R. (2011). A generative model for 3d urban scene understanding from movable platforms. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1945–1952. IEEE.

Gordon, N., Salmond, D., and Smith, A. (1993). Novel approach to nonlinear/non-gaussian bayesian state estimation. *IEEE Proceedings F, Radar and Signal Processing*, 140:107–113.

Gurău, C., Rao, D., Tong, C. H., and Posner, I. (2017). Learn from experience: probabilistic prediction of perception performance to avoid failure. *The International Journal of Robotics Research*, page 0278364917730603.

Gurău, C., Tong, C. H., and Posner, I. (2016). Fit for purpose? predicting perception performance based on past experience. In *International Symposium on Experimental Robotics*, pages 454–464. Springer.

Haklay, M. and Weber, P. (2008). Openstreetmap: User-generated street maps. *Ieee Pervas Comput*, 7(4):12–18.

Hawkins, J. and Blakeslee, S. (2007). *On intelligence: How a new understanding of the brain will lead to the creation of truly intelligent machines*. Macmillan.

Held, R. and Hein, A. (1963). Movement-produced stimulation in the development of visually guided behavior. *Journal of comparative and physiological psychology*, 56(5):872.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hoermann, S., Henzler, P., Bach, M., and Dietmayer, K. (2018). Object Detection on Dynamic Occupancy Grid Maps Using Deep Learning and Automatic Label Generation.

Hu, H. and Kantor, G. (2017). Introspective evaluation of perception performance for parameter tuning without ground truth. *Proceedings of Robotics: Science and Systems, Boston, USA*.

Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5967–5976. IEEE.

Israelsen, B. W. and Ahmed, N. R. (2017). " dave... i can assure you... that it's going to be all right..."–a definition, case for, and survey of algorithmic assurances in human-autonomy trust relationships. *arXiv preprint arXiv:1711.03846*.

Jaderberg, M., Simonyan, K., Zisserman, A., et al. (2015). Spatial transformer networks. In *Advances in neural information processing systems*, pages 2017–2025.

Jayaraman, D. and Grauman, K. (2015). Learning image representations tied to ego-motion. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1413–1421.

Julier, S. J., Uhlmann, J. K., and Durrant-Whyte, H. F. (1995). A new approach for filtering nonlinear systems. In *American Control Conference, Proceedings of the 1995*, volume 3, pages 1628–1632. IEEE.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45.

Kitani, K. M., Ziebart, B. D., Bagnell, J. A., and Hebert, M. (2012). Activity forecasting. In *European Conference on Computer Vision*, pages 201–214. Springer.

Konolige, K., Bowman, J., Chen, J., Mihelich, P., Calonder, M., Lepetit, V., and Fua, P. (2010). View-based maps. *The International Journal of Robotics Research (IJRR)*.

Krekelberg, B., Dannenberg, S., Hoffmann, K.-P., Bremmer, F., and Ross, J. (2003). Neural correlates of implied motion. *Nature*, 424(6949):674.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Krüsi, P., Bücheler, B., Pomerleau, F., Schwesinger, U., Siegwart, R., and Furgale, P. (2014). Lighting-invariant adaptive route following using iterative closest point matching. *Journal of Field Robotics (JFR)*.

Le, Q. V. (2013). 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.

Lee, G. H., Pollefeys, M., and Fraundorfer, F. (2014). Relative pose estimation for a multi-camera system with known vertical direction. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Linegar, C. (2016). *Vision-Only Localisation Under Extreme Appearance Change*. PhD thesis, University of Oxford, Oxford, United Kingdom.

Linegar, C., Churchill, W., and Newman, P. (2015). Work smart, not hard: Recalling relevant experiences for vast-scale but time-constrained localisation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Linegar, C., Churchill, W., and Newman, P. (2016a). Made to measure: Bespoke landmarks for 24-hour, all-weather localisation with a camera. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 787–794. IEEE.

Linegar, C., Churchill, W., and Newman, P. (2016b). Made to measure: Bespoke landmarks for 24-hour, all-weather localisation with a camera. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Stockholm, Sweden.

Lotter, W., Kreiman, G., and Cox, D. (2016). Deep predictive coding networks for video prediction and unsupervised learning. *arXiv preprint arXiv:1605.08104, 2016*.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110.

Luo, W., Yang, B., and Urtasun, R. (2018). Fast and furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3569–3577.

Maddern, W., Pascoe, G., Linegar, C., and Newman, P. (2017). 1 Year, 1000km: The Oxford RobotCar Dataset. *The International Journal of Robotics Research (IJRR)*, 36(1):3–15.

Martin, S. and Corke, P. (2014). Long-term exploration & tours for energy constrained robots with online proprioceptive traversability estimation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Martins, R. (2017). *Direct visual odometry and dense large-scale environment mapping from panoramic RGB-D images*. PhD thesis, PSL Research University.

Matthies, L. (1989). *Dynamic stereo vision*. PhD thesis, Carnegie Mellon University USA.

Medsker, L. and Jain, L. (2001). Recurrent neural networks. *Design and Applications*.

Merzić, H., Stumm, E., Dymczyk, M., Siegwart, R., and Gilitschenski, I. (2017). Map quality evaluation for visual localization. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3200–3206. IEEE.

Milan, A., Rezatofighi, S. H., Dick, A. R., Reid, I. D., and Schindler, K. (2017). Online multi-target tracking using recurrent neural networks. In *AAAI*, volume 2, page 4.

Mitchell, T. M. and Thrun, S. B. (1993). Explanation-based neural network learning for robot control. *Advances in neural information processing systems*, pages 287–287.

Muehlfellner, P., Furgale, P., Derendarz, W., and Philippsen, R. (2013). Evaluation of Fisheye-Camera Based Visual Multi-Session Localization in a Real-World Scenario. In *IEEE Intelligent Vehicles Symposium (IV)*.

Musoff, H. and Zarchan, P. (2009). *Fundamentals of Kalman filtering: a practical approach*. American Institute of Aeronautics and Astronautics.

Newson, P. and Krumm, J. (2009). Hidden markov map matching through noise and sparseness. In *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*, pages 336–343. ACM.

Ondruska, P., Dequaire, J., Zeng Wang, D., and Posner, I. (2016). End-to-End Tracking and Semantic Segmentation Using Recurrent Neural Networks. In *Robotics: Science and Systems, Workshop on Limits and Potentials of Deep Learning in Robotics*.

Ondruska, P., Gurau, C., Marchegiani, L., Tong, C. H., and Posner, I. (2015). Scheduled perception for energy-efficient path following. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Ondruska, P. and Posner, I. (2016). Deep Tracking: Seeing Beyond Seeing Using Recurrent Neural Networks. *Proceedings of the 30th Conference on Artificial Intelligence (AAAI 2016)*, pages 3361–3367.

Ostafew, C. J., Schoellig, A. P., and Barfoot, T. D. (2013). Visual teach and repeat, repeat, repeat: Iterative learning control to improve mobile robot path tracking in challenging outdoor environments. In *Intelligent Robots and Systems (IROS). IEEE/RSJ International Conference on*.

Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. In *IEEE Transactions on Knowledge and Data Engineering*, volume 22, pages 1345–1359, Los Alamitos, CA, USA. IEEE Computer Society.

Pascanu, R., Mikolov, T., and Bengio, Y. (2012). On the difficulty of training recurrent neural networks. *arXiv preprint arXiv:1211.5063, 2012*.

Pathak, D., Girshick, R. B., Dollár, P., Darrell, T., and Hariharan, B. (2017). Learning features by watching objects move. In *CVPR*, volume 1, page 7.

Patraucean, V., Handa, A., and Cipolla, R. (2015). Spatio-temporal video autoencoder with differentiable memory. *arXiv preprint arXiv:1511.06309, 2015*.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, 12:1532–1543.

Petrovskaya, A. and Thrun, S. (2009). Model based vehicle detection and tracking for autonomous urban driving. *Autonomous Robots*, 26(2):123–139.

Rasmussen, C. E. and Williams, K. I. (2012). Gaussian processes in machine learning. The MIT Press.

Rosten, E., Reitmayr, G., and Drummond, T. (2005). Real-time video annotations for augmented reality. *International Symposium on Visual Computing (ISVC)*.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016). Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

Särkkä, S. (2013). *Bayesian filtering and smoothing*, volume 3. Cambridge University Press.

Sibley, G., Mei, C., Reid, I., and Newman, P. (2010). Planes, trains and automobiles - autonomy for the modern robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Sivic, J. and Zisserman, A. (2003). Video Google: A text retrieval approach to object matching in videos. In *Proceedings of the International Conference on Computer Vision (ICCV)*.

Stein, M. L. (2012). *Interpolation of spatial data: some theory for kriging*. Springer Science & Business Media.

Strasdat, H., Davison, A. J., Montiel, J., and Konolige, K. (2011). Double window optimisation for constant time visual slam. In *Computer Vision (ICCV). IEEE International Conference on*.

Suleymanov, T., Amayo, P., and Newman, P. (2018). Inferring road boundaries through and despite traffic. In *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, Maui, Hawaii, USA.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic robotics*. MIT press.

Velez, J., Hemann, G., Huang, A. S., Posner, I., and Roy, N. (2011). Planning to perceive: Exploiting mobility for robust object detection. In *International Conference on Automated Planning and Scheduling*.

Vu, T.-D., Aycard, O., and Appenrodt, N. (2007). Online localization and mapping with moving object tracking in dnamic outdoor environments. pages 190–195.

Vysotska, O. and Stachniss, C. (2017). Improving SLAM by exploiting building information from publicly available maps and localization priors. *PFG–Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, 85(1):53–65.

Walker, J., Gupta, A., and Hebert, M. (2014). Patch to the future: Unsupervised visual prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3302–3309.

Wang, D. Z., Posner, I., and Newman, P. (2015). Model-free detection and tracking of dynamic objects with 2d lidar. *The International Journal of Robotics Research*, 34(7):1039–1063.

Wang, S., Clark, R., Wen, H., and Trigoni, N. (2017). Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2043–2050. IEEE.

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

Watters, N., Tacchetti, A., Weber, T., Pascanu, R., Battaglia, P., and Zoran, D. (2017). Visual interaction networks. *arXiv preprint arXiv:1706.01433*.

Weiss, K., Khoshgoftaar, T. M., and Wang, D. (2016). A survey of transfer learning. *Journal of Big Data*, 3(1):9.

Xingjian, S., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-k., and WOO, W.-c. (2015). Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *Advances in Neural Information Processing Systems (NIPS), Montreal, Canada, December 7-12, 2015*, pages 802–810, Red Hook, NY, USA. Curran Associates Inc.

Xue, T., Wu, J., Bouman, K. L., and Freeman, W. T. (2016). Visual Dynamics: Probabilistic Future Frame Synthesis via Cross Convolutional Networks. pages 1–11.

Yang, M., Wu, Y., and Hua, G. (2009). Context-aware visual tracking. *IEEE transactions on pattern analysis and machine intelligence*, 31(7):1195–1209.

Yang, S.-W. and Wang, C.-C. (2011). Simultaneous egomotion estimation, segmentation, and moving object detection. *Journal of Field Robotics*, 28(4):565–588.

Yu, F. and Koltun, V. (2015). Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122, 2015*.

Yuen, J. and Torralba, A. (2010). A data-driven approach for event prediction. In *European Conference on Computer Vision*, pages 707–720. Springer.

Zentall, T. R. (2005). Animals may not be stuck in time. *Learning and Motivation*, 36(2):208–225.

Zhang, H., Geiger, A., and Urtasun, R. (2013). Understanding high-level semantics by modeling traffic patterns. In *Proceedings of the IEEE international conference on computer vision*, pages 3056–3063.

Zhao, L. and Thorpe, C. (1998). Qualitative and quantitative car tracking from a range image sequence.