Technische Universität Hamburg-Harburg

Indoor Human Tracking based on Dynamic Models from Convolutional Neural Networks

Master Thesis

by

Liangcheng Fu

Start date:	01 August 2017
End date:	30 January 2018
First Examiner:	Prof. Alexander Schläfer
Second Examiner:	Prof. Dr.-Ing. habil. Udo Zölzer
Supervisor:	Johannes Döllinger

BOSCH

Abstract

A short abstract of the thesis in English.

Statement

I assure the single handed composition of this master's thesis only supported by declared resources.

Hamburg, 30. January 2018

(Liangcheng Fu)

Foreword

blabla...

Hamburg, 19. December 2016

Contents

Chapter 1

Introduction

In recent years, more and more robots are deployed in not only industrial environments but also human populated areas. In order to fulfill their tasks, it is necessary for robots to interact and even cooperate with people. Therefore, tracking the locations of people in those environments has gained enormous attentions in robotics community. Besides, with the increasing popularity of artificial intelligence, robots are also expected to be intelligent enough to predict possible future locations of walking human even when encountered with occlusions or missing of sensor data. Enabled with an accurate tracking and prediction of human movements, robots are able to have a better understanding of the environment, which will facilitate interactions between robots and human.

The very fist requirement for a robot to operate is to model the environment. A simple yet effective way is to use occupancy grid map representation of the environment [12]. It decomposes the environment into cells with a predefined resolution, and the state of each cell is a random variable with values of either *occupied* or *not occupied*. This representation has been extensively used in many different kind of robot tasks like simultaneous localization and mapping (SLAM). The classical grid map representation treats the environment as static, which is not always the case in real scenarios. In other words, the environment can be dynamic since objects can move around in the environment. Object tracking addresses dynamics in environment explicitly, since it tracks and predicts how objects move. If there are more than one object involved at the same time, the tracking problem becomes multiple object tracking (MOT). In people tracking systems, the dynamics in environment refer to people location changes along the time horizon (i.e., trajectories).

Commonly, a tracking system is implemented as a multiple stage pipeline, which consists of object detection, data association, motion modeling and occupancy generation. When deal with a multiple object tracking problem, data association becomes very tricky. The classical way to perform data association is to maintain a list of known objects, and associates new observations with those objects. The main difficulty of this approach is to deal with *birth* (whether observation is from a new object) and *death* (whether a maintained object should

be deleted) of tracks explicitly [14].

To address the data association problem, Coué et al. [9] proposed *Bayesian occupancy filter* (BOF), which is an object tracking algorithm based on grid map representation of environment. It essentially avoids data association step in the tracking pipeline, since concepts of *objects* and *tracks* dose not exist in BOF framework. Rather than treat tracking problem from an *object* point of view, BOF addresses tracking from a *cell* perspective. That is to say, tracks are replaced by transitions of occupancies between cells over time. Meanwhile, BOF is robust to object occlusion thanks to the combination of *prediction* and *estimation* steps. This two-step mechanism is well suited for handling the consistency between occupancy predictions and new observations, therefore uncertainties incurred by occlusion is naturally handled in a probabilistic way. Over the past few years, there has been some extensions over BOF proposed in literature [15, 5, 29].

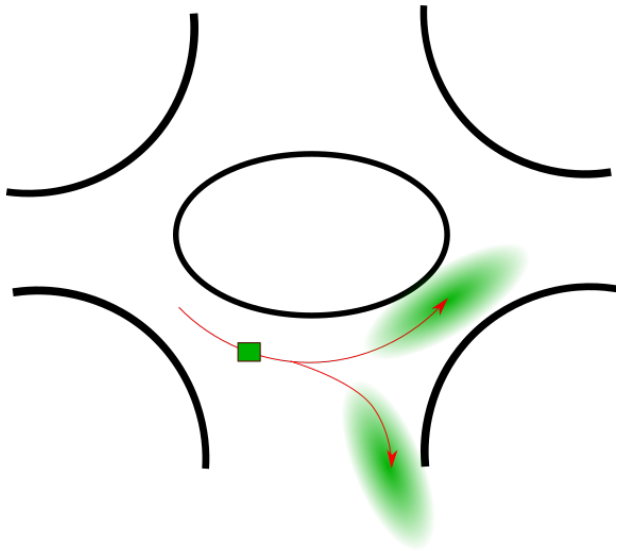

Figure 1.1: *This shows an illustration of a round-about. Cars can only drive in directions indicated by red arrows. The green rectangle represents a car at that location. Since we know that the car must drive in the specified directions, the possible locations of the car after a few time steps are indicated by the two green eclipses. This prior knowledge helps us track the dynamic objects since we have better chances to locate the car according to the motion pattern.*

A tracking system usually predicts the state of the world (e.g., object location and velocity) based on past observations. Many tracking algorithms also incorporate motion models of the objects being tracked, since this allows us to utilize prior knowledge of object's motion cues. As an intuitive example, let us consider a round-about as shown in Figure 1.1, where cars can only drive in one direction. To track a car driving in a round-about, prediction of next possible locations of the car should always be on the side which allowed driving direction indicates, since we know that the car must follow that direction. Likewise, human

tracking in indoor environments can also benefit from human motion patterns. For BOF and its variants, the objects being tracked are assumed to perform linear motion, which indicates that objects always move in the same direction as in the last time step. Although Gindele et al. [15] proposed to incorporate prior map knowledge (BOFUM) to better model the motion dynamics based on cell context, linear motion is too simplistic to model actual human motion in indoor environments.

Based on our observations of human trajectories, we found two aspects very important for explaining human motion: 1) Unlike in free space, people have to move under spatial constraints in indoor environments. For example, people tend to walk along the central area between walls in a corridor and change their walking directions when they have to make turns. That is to say, the spatial constraints limit human motion patterns and therefore human motion is very place dependent. 2) People move continuously in time and space, which means they does not disappear or appear out of thin air. If a person is currently at a cell of a grid map, he or she has to walk through its neighboring cells first. In other words, future movement events starting from a region of interests are highly influenced by state changes of its neighboring cells. These observations motivates us to model human motion in a way that captures both **place dependency** as well as **spatial correlation** between cells.

In computer vision community, researches in machine learning over last few years has shown a lot of successes. Image classification algorithms based on convolutional neural networks (CNNs) has won all ImageNet classification challenges since 2012 [39]. It turns out that CNNs can achieve good results in not only computer vision tasks but also many other domains. For example, recurrent neural networks (RNNs) can be applied in automatic language translation [8]. Deep reinforcement learning are good choice for teaching robots to perform human actions like grasping[27]. Essentially, the reason why CNN works in those domains is that it is able to capture complex structure in data. Once a CNN learns that structure, it can generalize to cases that never occurs during training.

In order to utilize the powerful generalization abilities of CNNs, we proposed to model motion patterns in a way that can be easily expressed as the output of a CNN. In this way, if we feed a gird map as input and human motion patterns as ground truth to a network, it should be able to learn the patterns after training with lots of data. We model human motion patterns as a set of conditional probabilities for every cell on the map. They represent how likely a person moves to one of the neighboring cells, conditioned on which neighboring cell he or she comes from. Since these probabilities incorporate information about the incoming cell, the motion model captures spatial correlations between cells. Besides, they are different for each cell based on cell's context on the map, which means the learned motion pattern is expressive enough to predict different motions at different locations. In other words, our motion model is place dependent. In fact, similar idea of making motion patterns place dependent has been proposed by [26], but in order to learn motion pattern on a map, they need sensor observations from that specific map. In other words, their method lacks of generalization ability. On the contrary, with a large training

set, we exploit the power of CNNs so that our method can generalize to maps that have never occurs.

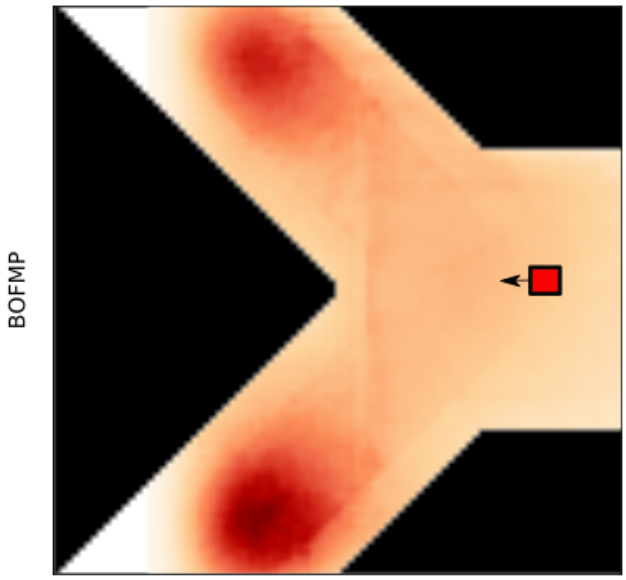

Figure 1.2: *An example of future occupancy predictions of our method. The map shows a T-section, where only white spaces are walkable. Initially, a person is shown as a red rectangle and with velocity towards left. Since no further sensor reading is given, BOFMP propagates occupancies over time. After some time interval, most occupancies appear in both upper and lower branches of the corridor, which proves that our motion model successfully learns that people tend to make turns at the intersection. Besides, our BOFMP predicts more occupancies in the middle of corridors, since it learns that people are less likely to walk near walls.*

The data we use for training network is generated from simulated human walking trajectories on SLAM-generated maps of real world offices. Once the network finishes learning, we can feed new maps to the network and obtain human motion patterns from network outputs. These motion patterns are then incorporated into BOF framework to perform human tracking in indoor environments. We call our tracking method as *Bayesian Occupancy Filter with Motion Patterns* (BOFMP). If only initial state of a person is given and no further observations are provided, our method is able to predict reachable areas after some time steps. Figure 5.8 shows an example of future occupancy predictions of our method without sensor observations. It shows that our way of modeling human motion patterns enables BOF to propagate occupancies to reasonable areas.

The main contributions of this thesis work are:

1. We present a way of modeling human motion patterns that captures both place dependency and spatial correlations between cells. Besides, thanks to powerful gen-

eralization abilities of CNNs, our method can generate motion patterns on maps that are never seen by our model.

2. We incorporate the learned motion patterns into BOF framework seamlessly for human tracking in indoor environment, and achieve better tracking performance than baseline method. The whole pipeline of modeling motion pattern and tracking presented in this thesis work can be applied in other scenarios, such as car tracking in ADAS systems.

The rest of this thesis is organized as follows: Chapter 2 summarizes the related works and highlights the similarities and differences between our method and others in literatures. Chapter 3 introduces how BOF is derived and mathematical formulation of CNNs. Chapter 4 explains the detailed implementations, such as generation of dataset and BOF. Chapter 5 summarizes the dataset and presents the tracking performance of our method and the baseline. Chapter 6 concludes the thesis work and presents possible improvements on our method.

Chapter 2

Literature Review

This chapter lists literatures that are related to methods or concepts used in this thesis work. Section 2.1 discusses two classical ways of tracking object and introduces BOF and its variants. Section 2.2 presents literatures that try to model dynamics in environment. Section 2.3 details recent researches on neural networks and correlations between RNNs and Bayesian filters.

2.1 Object Tracking

Perception of environment can be done with different sensors, which also differentiates object tracking applications. Visual tracking refers to object tracking with RGB cameras [37]. In this thesis work, however, we mainly concern object tracking with 2D laser scanners. One popular paradigm of performing tracking is tracking by detection: moving objects are firstly detected and then determine how to pair them with existing tracks. Generally, there are two approaches to deal with detection based object tracking: *model-free* and *model-based* [44].

The detection of model-free object tracking works based on motion cues. In this case, prior knowledge on neither object's semantic information (whether it is a car or a person) nor object's shape or geometric properties is needed. However, this approach only tracks objects that show instantaneous motion dynamics and potential moving objects might be neglected. BOF can be regraded as a model-free tracking algorithm since it predicts occupancies based on cells' velocities. Similar to BOF, Ross et al. [37] models the environment with occupancy gird map. Their work combines SLAM with dynamic object tracking in outdoor environment. Once the vehicle equipped with laser sensors is self-located and objects are detected by motion cues, they apply a method called Global Nearest Neighborhood (GNN) for tracking. On the contrary, for model-based tracking, semantic class of the objects being tracked is given, and detection is done with the help of a parametric

model of the object's shape. For human tracking, a person is represented as point blob or modeled based on detection of legs [3, 10].

The tracking by detection paradigm has to address *data association* explicitly, which is known to be as the main difficulty in tracking. Data associations refers to match detections of moving objects in new observations to a set of tracked trajectories from last time step. To address data association problems in tracking, many methods have been proposed in literature. Historically, nearest neighbors based approaches are used in the early stage [13]. Schulz et al. [40] detect people by their legs in 2D laser scans and propose the Joint Probabilistic Data Association Filter (JPDAF) for tracking. Arras et al. [3] is similar in identifying people with legs but their tracking is done with Multi-hypothesis tracking (MHT). Although the mentioned methods partially address the data association problem, their performance is not stable in densely cluttered environments, where occlusions happen quite often.

On the contrary to tracking by detection paradigm, tracking can also be done without detection, e.g., tracking by filtering. Coué et al. [9] propose Bayesian Occupancy Filter for object tracking in automotive applications. The environment is represented by a grid map, and the state of each cell is characterized by its occupancy and velocity. Inspired by Bayesian Filter, tracking works in a form of recursive *predictions* and *estimations* of cell's state. One of important advantages of BOF is that the concepts of *objects* and *tracks* are replaced by transitions of occupancies between cells over time. That is to say, data association is addressed implicitly by BOF. Later works extend BOF to better adapt to real world situations. BOF's linear motion model cannot adapt to traffic scenarios like curved roads. Therefore, Gindele et al. [15] propose to enrich motion model of BOF with prior map knowledge (BOFUM), which can be obtained from navigation systems. By this way, BOFUM predicts occupancies according to motion preferences at different locations, which results in reliable estimates even when occlusion happens.

Despite of various variants of BOF, all of them assume that objects perform linear motion. However, linear motion is obviously over-simplified for real world scenarios. Besides, since objects cannot appear or disappear suddenly, the state changes of a cell's neighboring cells contain valuable information for prediction of its future state. Therefore, we proposed to model human motion patterns in a way that captures both place dependency and spatial correlation of cells. This motion model is then incorporated into BOF framework to perform human tracking. A similar idea is used in the work of Luber et al. [31], which proposed a place dependent people tracking algorithm. Unlike BOF which tracks objects by filtering, they perform people tracking by detection and data association. They proposed to model human activity events as a three-layer spatial affordance map with each layer characterizing the probability distribution of new tracks, matched tracks and false alarms respectively. Each layer is represented by a *spatial* Poisson process which introduces spatial dependency and the parameter of each cell is learned from a sequence of observations. Figure 2.1 shows one example of two layers of the learned spatial affordance map. Their motion model is place dependent because it is dependent on the spatial affordance map. They use boosted

features for people detection [2] and they extend multiple hypothesis tracking (MHT) for data association. Their results show they achieve more accurate tracking behavior in terms of data association errors.

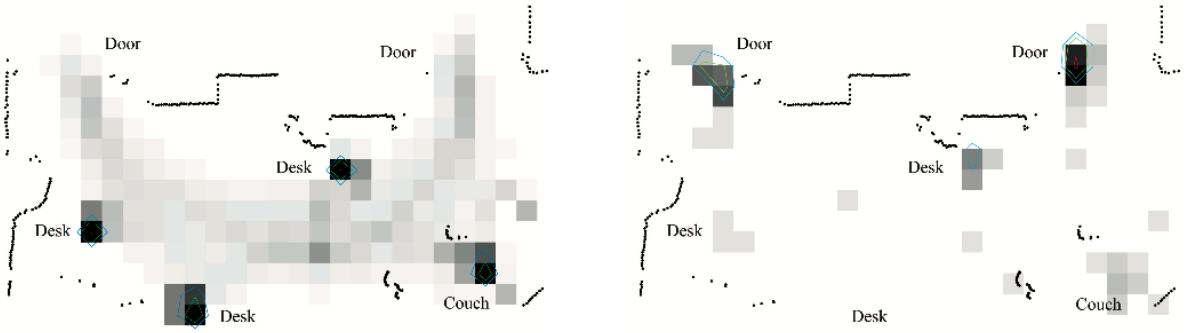

Figure 2.1: *Two layers of a learned spatial affordance map [31]. On the left shows probability distribution of matched track events, and on the right shows that of new track events. Note that the maxima show at different regions in two maps. While on the left the maxima appear at places frequently used by people (desks and coach), on the right the maxima of new track events indicate places where people normally appear from, i.e., doors, desks and coach.*

2.2 Dynamics Modeling

Nowadays, more and more robots are entering *dynamic* environments. In order to fulfill their task, it is imperative for robots to understand these dynamics. In the context of human tracking, these dynamics refer to human movements in the environment. In literature, some researchers try to model human dynamics explicitly by a motion model. Meanwhile, others model them implicitly by regarding human dynamics as part of the environment and therefore model the environment directly. We discuss dynamics modeling separately according to these two approaches. In both cases, we represent the environment as a gird map with each cell representing a possible location.

2.2.1 Human Motion Modeling

In the early stage of human tracking, researches adopts simple and conservative motion models. Montemerlo et al. [34] uses Brownian motion model for human tracking with Bayesian filters. The Brownian motion assumes people take random directions at each time step and there is no dependence between time steps. In other words, Brownian motion does not assign human dynamics with any pattern other than dispersion. As a consequence, when there is no observations, the predictions of people locations spread out

over a large area very quickly. This is a poor estimate as we know people normally does not move randomly. In reality, people normally go from the starting point and follow some motion patterns until they reach destination. A more realistic model is the first order motion model (a.k.a. linear motion model). For example, Meier and Ade [32] used this model together with Kalman filter for human tracking. First order motion model assumes people always move in the same direction as in the last time step. This assumption might be true if a person is walking along a straight corridor, but in many cases it is invalid because people often need to make turns around corners. Some researchers proposed better ways to model people motion patterns. For example, Bruce and Gordon [6] learns destinations by clustering real trajectories and uses a path planer to those destinations as a reference for human motion patterns. Liao et al. [28] assumes that human tend to move along Voronoi graph of the environment and therefore constraint motion patterns by Voronoi graph.

Our motion model is different to above methods in two aspects: fineness and cell dependency. The Voronoi graph is constructed based on a *global* representation of the environment. It is predefined and only well suited for applications where high-level motion clue is needed (e.g., which rooms a person has visited). This kind of motion clues are too less precise to be used for recovering a person's exact location. Therefore, in order to get a finer motion model, we decompose the *global* task of modeling human motion pattern into *local* tasks at *cell level*. On the other hand, although a path planer to the learned destination defines an effective path, it does not necessarily cover the motion dynamics at every possible locations and assumes no dependency between these locations. One intuition we captured based on observations of human motion is that, for a cell in the map, the state changes of its neighboring cells is a strong indicator of predictions of its future state. To capture this *spatial* correlation, we propose to model changes of movement directions. For each cell, we learn based on human trajectories how likely a person's moving direction changes for the next time step, conditioned on the direction with which he or she enters this cell. Moreover, this way of motion modeling also captures *temporal* correlations, since it builds connections for each cell with its neighboring cells from both last time step and next time step.

In fact, after we came up with above way of modeling human motion, we found in literature the same idea has been used by Kucner et al. [26] in an autonomous navigation scenario. Their model, which is named as Conditional Transition Map, is created by "learning the probability distribution of an object leaving to a certain neighboring cell, given the cell from which it entered into the current cell." They demonstrate their method with a roundabout, and the corresponding model is illustrated in Figure 2.2. One can see that at different locations in the roundabout, different motions are learned, which accurately models how cars drive through the roundabout. The difference between our method and theirs lies in how those probabilities are learned. The cross-correlation of temporal occupancy signals extracted from observations is used for learning in their method, while our model learns these probabilities by training a neural network. With lots of different maps as training data, our network is able to learn the motion patterns that are constrained by spatial structures of these maps. One of the most important advantages of our method is that

our network can generalize motion patterns for maps that never occur in training data. Therefore, our method can work in new environments without learning these probabilities from scratch.

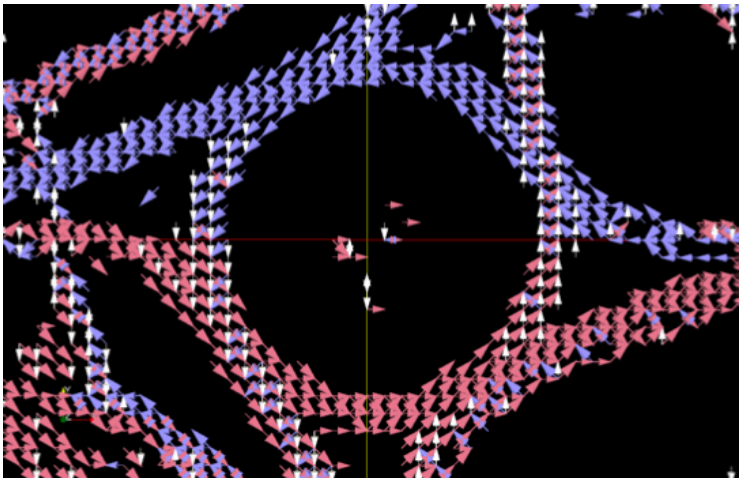

Figure 2.2: *Conditional transition map of a roundabout [26]. Arrows indicate the exit directions for that cell. For simplicity, the entering directions are not drawn. One can see that this model captures how cars drive through the roundabout.*

2.2.2 Dynamic Environment Modeling

For a mobile robot to operate, a proper way to model the environment is necessary. Elfes [12] introduces occupancy grid as a representation of environment, which divides the environment into a grid of cells with a predefined resolution. Each cell is a random variable with values of either *occupied* or *not occupied*. The occupancy grid assumes the environment being static, which does not hold in real world situations. For example, a robot might need to work in a traffic intense environment with driving cars and pedestrian.

Early attempts to model dynamics in environment extents occupancy gird with a timescale framework. Arbuckle et al. [1] propose to model dynamics with a stack multiple occupancy grids with each layer corresponding to a different timescale. They call this representation as Temporal Occupancy Grid (TOG). Objects' motion pattern can then be classified based on objects' occupancy traces across different layers of TOG. For example, if a cell is occupied across all layers of TOG (i.e., occupied at all timescales), it is likely to be part of background. Similarly, traffic patterns with various speeds can be identified based on its occupancy traces on a certain layer of TOG. Biber and Duckett [4] uses multiple map representations of the dynamic environment with different timescales for long-term SLAM. Updates of mapping is calculated based on a (dynamic) sample set of observations that is updated over time by random replacement. Different timescales correspond to different learning rate. Map with higher learning rate adapts to new observations faster than these with lower learning rates.

Although the above methods can be applied for modeling dynamics, essentially their static nature remains if we look at each layer that characterizes a specific timescale. In order to capture the dynamic nature of environment, Meyer-Delius et al. [33] propose to model state changes of cells by a two-state hidden Markov models (HMMs). Therefore, the dynamics in environments are explicitly characterized by the state transition probabilities of HMMs. The parameters of these HMMs are estimated with an Expectation Maximization (EM) algorithm.

However, although their method relaxes the static cell state assumption made by occupancy grid, they assume that state changes of cells are caused by a *stationary* process, i.e., cell's state transition probabilities are constants over time. Moreover, the assumption of independences between cells is not always valid (e.g., state changes of a cell's neighboring cells are strong indicators of that cell's future state because objects normally move in a continuous manner). Due to above reasons, cell-specific input-output HMMs (IOHMMs) are used by Wang et al. [45] to better model dynamic environments. An IOHMM imposes conditional dependences on latent variables and observed variables with an input sequence. An illustration of IOHMM used in their method is depicted in Figure 2.3. The input sequence is determined by past observations in neighboring cells. By this way, the transition model of their method is adaptive to input sequence that varies over time, which relaxes the assumption of stationary process made by Meyer-Delius et al. [33]. It also incorporates spatial correlations by making input sequence dependent on neighboring cells.

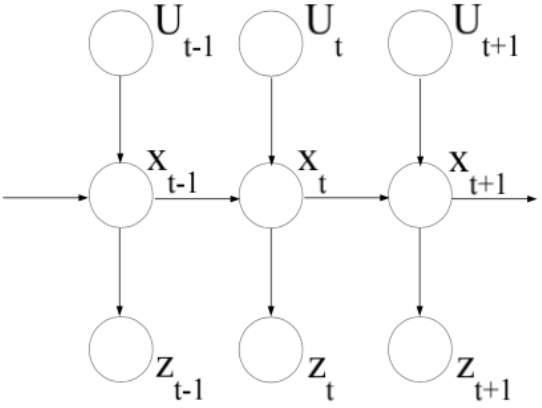

Figure 2.3: *An illustration of IOHMM used by Wang et al. [45]. Note in this IOHMM only latent variables are dependent on input sequence, which is determined by past events in neighboring cells.*

Our method is conceptually equivalent to theirs. In both methods, the influence of neighboring cells is captured by conditional dependence. Although our motion model is fixed for a specific map, the occupancies propagate to a certain cell only when its neighboring cells have velocities towards it. This mechanism acts as a "trigger" which only get triggered at certain time and essentially makes our model time dependent.

2.3 Neural Networks

Since Krizhevsky et al. [25] applied deep convolutional neural networks (CNNs) in large scale image classification from 2012, CNNs have gained a lot of successes in computer vision. Researches show that the depth of network plays a very important role in CNNs' performance [42]. However, deep networks are more difficult to train, possibly due to poor gradient flow during back-propagation. To address this problem, He et al. [18] propose to fit a *residual* mapping with stacked layers of CNNs instead of fitting the desired underlying mapping directly. This idea is implemented as "shortcut connections" between layers, which branches from one layer and connects to one of the followed layers. Moreover, Huang et al. [20] extends the idea of residual learning so that each layer has a shortcut connection to every other layer in the network. This design guarantees layers with easy access to its preceding layers and makes feature reuse very easy. Figure 2.4 shows a five layer densely connected CNN (DenseNet). Compared with other CNN structures like ResNets, DenseNets are even better at alleviating vanishing-gradient problem and therefore improve classification accuracy with even fewer parameters.

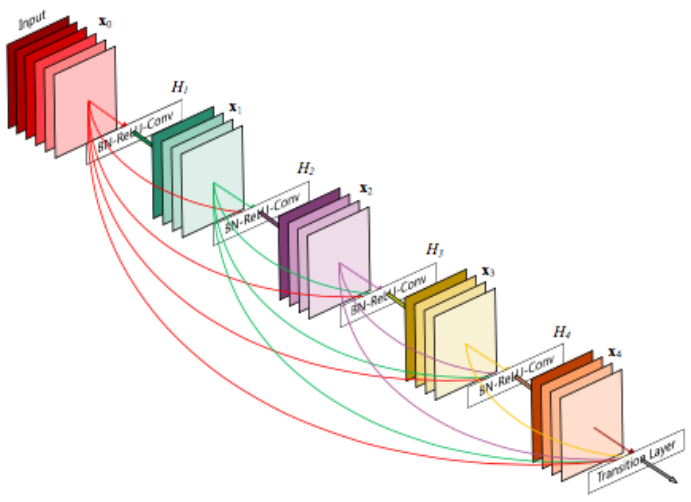

Figure 2.4: *A five layer dense block Huang et al. [20]. As can be seen, each layer has a shortcut connection to every other follower layer in the network.*

Besides image classification, CNNs have been extensively used in other computer vision tasks like semantic segmentation. Unlike in image classification where only one class has to be determined for the whole image, semantic segmentation requires classification at pixel level. *Fully* convolutional neural networks have been proposed by [30] to tackle this problem. By removing fully connected layers and adding a deconvolution layer that functions as upsampling, their network is able to take arbitrary size image as input and predicts pixel-wise classification. Their network also includes skip connections that combine deep

coarse features (e.g., semantic information) and shallow fine features (e.g., geometric structure and appearance), both of which are essential for semantic segmentation. Jégou et al. [23] extends fully convolutional networks with dense blocks, i.e., a block of convolutional layers that *densely* connected as in DenseNet. Their proposed structure is designed for semantic segmentation, with a downsampling path extracting high-level semantic feature and an upsampling path recovering outputs to full resolution as input image.

The network structure used in this thesis is similar to that of Jégou et al. [23]. Since our proposed motion model needs to be place dependent, our network is required to predict probabilities at cell level, similar to pixel level prediction in semantic segmentation. The differences lie in the meaning of these probability outputs. For semantic segmentation, these are probabilities of a pixel belonging to a specific semantic class, while in our case they are probabilities of object's exiting direction conditioned on which direction it takes to reach current cell.

As CNNs are good fit for image based tasks, recurrent neural networks (RNNs) are found to be very useful to deal with sequential data such as videos and texts. RNNs have been successfully applied in applications like object tracking [35], automatic language translation [8] and speech recognition[16]. Long short-term memory (LTSM) [19] addresses vanishing and exploding gradient problems and becomes one of most popular variants of RNNs.

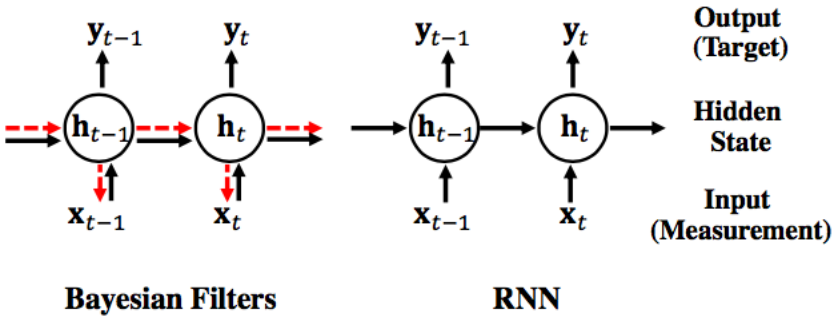

Figure 2.5: *Comparison between RNN and Bayesian filters [11]. For Bayesian filers, the dynamics are modeled by a Markov process (red dash lines). In both cases, measurements are fed into the systems as inputs.*

Ondruska and Posner [35] propose a framework named as *deep tracking* for object tracking with 2D laser data. Deep tracking uses RNNs as the underlying model, and it is designed to be trained in an end-to-end fashion from raw sensor data without hand-crafted feature engineering. Unlike in most supervised learning applications where human annotated ground truth is needed, they train the network with future sensor observations (i.e., future inputs to network) as ground truth, which essentially makes the training unsupervised. Further, Ondruska et al. [36] extends their deep tracking framework so that it can perform object tracking and semantic segmentation at the same time.

The fact that object tracking can be achieved by both Bayesian filter based approaches like BOF and RNN based approaches like deep tracking leads us to understand their

connections. In a paper that aims at dynamic facial analysis, De and Kautz [11] try to compare Bayesian filters with RNNs as depicted in Figure 2.5. Given noisy measurement as input, the goal of Bayesian filter is to estimate the hidden state and optionally the target output, while RNNs sequentially predict target output as a function of hidden state which is in turn dependent on new measurements. The performance of Bayesian filters is highly dependent on how much are the assumed state transition and measurement model close to real situations. While for RNNs, this step of handcrafted engineering is avoided since RNNs are able to model them implicitly by learning from data, which is similar to how CNNs extract features from images without artificial feature engineering. On the other hand, despite its successes in many applications, how the hidden states involve over time and how to interpret the hidden states remain a challenge in understanding RNNs. For Bayesian filter, interpretations of the transition processes are usually quite straightforward.

Chapter 3

Background Knowledge

This chapter introduces the theoretical background of this thesis, which mainly concerns two distinct parts: Bayesian Occupancy Filter (BOF) and Convolutional Neural Network. In Section 3.1.1, we start with introduction to the idea of Bayesian filtering. BOF is naturally introduced in Section 3.1.2, since it is an extension of Bayesian filtering on occupancy grid for object tracking. To better adapt BOF to human tracking context, Section 3.1.3 explains how our human motion model can be seamlessly incorporated into the BOF framework. In Section 3.2, we firstly introduce the general structure of CNNs, and the state-of-the-art densely connected CNNs (DenseNets) are explained in Section 3.2.1. To exploit the power of DenseNets for networks that produce *pixel-wise* predictions , the fully convolution DenseNets with upsampling path are introduced in Section 3.2.2. In Section 5.1, different metrics for evaluating the performance of tracking are discussed. The arguments for our choice of metrics used in this thesis are also provided.

3.1 Bayesian Occupancy Filter

In a tracking system, the locations of tracked objects are estimated by taking into accounts both incoming measurements from sensor and predictions from last time step. This process is very similar to Bayesian filtering, which is a probabilistic way to *recursively* estimate the state of a dynamic system.

3.1.1 Bayesian Filtering

Bayesian filters work in a dynamic system. At any time t, the true state x_t of the dynamic system is unobservable. Instead, measurements z_t can be obtained from sensor data and their relationship can be expresses as [22]:

$$z_t = h_t(x_t, \omega_t) \tag{3.1}$$

where h_t is normally given in this context and called as *observation model*, ω_t is known as *measurement noise*. The true state x_t evolves from last state x_{t-1} based on a *process model* f_t:

$$x_t = f_t(x_{t-1}, v_{t-1}) \tag{3.2}$$

where variable v_t is called as *process noise*. This equation describes a *Markov process* with order one, since current state x_t does not depend on past states of $x_{1:t-2}$ and only depends on last state x_{t-1}. For a dynamic system, the goal of Bayesian filtering is to recursively estimate the *posterior* probability distribution, $P(x_t|z_{1:t})$. This is done in two steps: *prediction* and *estimation*. In the prediction step, a *prior* distribution on the state of the system is calculated based on posterior distribution from last time step and the process model. In the estimation step, the prior distribution is used with new measurements to calculate posterior distribution on the current state.

3.1.2 BOFUM Formulation

BOF, similar to Bayesian filer, consists of two stages of *prediction* and *estimation*. Bayesian Occupancy Filter (BOF) can be applied on robots to reason about the dynamic environment, which in this context is represented by occupancy grid. Applications of BOF include collision avoidance [9] and object tracking [7].

Historically, based on different representations of the cell's state space, there are two formulations of BOF: 4D-BOF and 2D-BOF. In 4D-BOF, each cell has four dimensions, with two dimensions representing its position on the gird, and the other two representing the orthogonal velocity components [9]. While for 2D-BOF, cells are organized as in a 2-dimensional occupancy grid, and each cell is associated with probability distributions of its velocity. The advantage of 4D-BOF is that it is able to represent overlapping objects with different velocities. However, 4D-BOF is more computationally expensive and it is not able to estimate cell's velocity. On the contrary, 2D-BOF dose not allow overlapping objects but requires less computations and is able to infer velocities. In this thesis, we follow the 2D-BOF formulation as proposed by Gindele et al. [15], which extends original 2D-BOF using prior map knowledge (BOFUM) as an indicator of motion preference.

We start the formulation of BOFUM by introducing relevant random variables. The subscript n of each random variable represents the cell index on the grid map. We assume there are N cells in total. For time step t, we define:

O : Vector for occupancy state of all cells. Each element O_n takes two possible values, *occ* and *nocc*.

$$O = (O_1,\ \dots\ ,\ O_N)^T \in \{occ, nocc\}^n$$

V : Vector for velocities in x and y directions. Velocities are discretized into cells, which have unit of *cells/timestep*.

$$V = (V_1,\ \dots\ ,\ V_N)^T \in \{\mathbb{Z}, \mathbb{Z}\}^n$$

X : Combination of both occupancy and velocity.

$$X = (O,\ V)$$

X^- : Occupancy and velocity state at time $t-1$.

$$X^- = (O^-,\ V^-)$$

R : Matrix of random variables describing reachability from cell a to cell c. This information is hand-crafted based on cell's context on the grid map.

$$R \in \{reach, nreach\}^{n \times n}$$

T: Transition vector. For $T_i = j$, it represents that the occupancy in cell i will move to cell j in the next time step. The transition is an abstraction over velocity, reachability and any other information about the cell movement.

$$T = (T_1,\ \dots\ ,\ T_N)^T \in N^n$$

Z: Measurement vector that comes from sensor data.

$$Z = (Z_O,\ Z_V), Z_O \in \{occ, nocc\}^n, Z_V \in \{\mathbb{Z}, \mathbb{Z}\}^n$$

In our context, we use no sensors for measuring velocities. Therefore, Z_v will be omitted.

We assume that cells are independent to each other. This simplifies the formulation and also makes the implementation of BOF highly parallelisable. The goal of BOF is to estimate the posterior probability distribution of the state of a specific cell c given measurement Z:

$$P(X_c|Z_c) = \frac{P(X_c, Z_c)}{P(Z_c)} \tag{3.3}$$

$$= \frac{\sum_{X^-,R,T} P(X_c, X^-, R, T, Z_c)}{\sum_{X^-,R,T,X_c} P(X_c, X^-, R, T, Z_c)} \tag{3.4}$$

Since $P(Z_c)$ can be obtained by marginalization, i.e., $P(Z_c) = \sum_{X_c} P(X_c, Z_c)$, we only concern the numerator of Equation 3.3:

$$P(X_c|Z_c) \propto \sum_{X^-,R,T} P(X_c, X^-, R, T, Z_c) \tag{3.5}$$

In order to calculate joint probability $P(X_c, X^-, R, T, Z_c)$, we need to decompose it based on Baye's rules and some assumptions.

$$\begin{aligned} P(X_c, X^-, R, T, Z_c) &= P(X_c, X^-, R, T)P(Z_c|X_c, X^-, R, T) && (3.6)\\ &= P(X_c, X^-, R, T)P(Z_c|X_c) && (3.7)\\ &= \underbrace{P(X_c|X^-, R, T)P(X^-, R, T)}_{prediction}\underbrace{P(Z_c|X_c)}_{correction} && (3.8) \end{aligned}$$

So far, the two main stages of Bayesian filtering can been identified by Equation 3.8:

1. **prediction**: The prediction step calculates a prior distribution of the state of cell X_c, which is joint distributed with past state X^-, reachability R and transition T. It can be further decomposed as in Equation 3.8.

   $P(X_c|X^-, R, T)$ specifies the *process model*, which states how the current state can be predicted given a prior state and extra knowledge. Since occupancy and velocity are independent and transition subsumes velocity and reachability, we have

   $$P(X_c|X^-, R, T) = P(O_c|O^-, T)P(V_c|O^-, T) \tag{3.9}$$

   For the propagation of occupancy, we assume that it is sufficient for a cell c to be occupied, if at least one antecedent cell a moves to cell c. This assumption can be expressed as:

   $$P(O_c = occ|O^-, T) = \begin{cases} 1, & \vee_{a\in N}((O_a = occ) \wedge (T_a = c)) \\ 0, & else \end{cases} \tag{3.10}$$

   For simplicity of implementation, we calculate '*nocc*' instead of calculating '*occ*':

   $$\begin{aligned} P(O_c = occ|O^-, T) &= 1 - P(O_c = nocc|O^-, T) && (3.11)\\ &= \prod_{a\in N}(1 - P(O_c = occ|O_a^-, T_a)) && (3.12) \end{aligned}$$

   where

   $$P(O_c = occ|O_a^-, T_a) = \begin{cases} 1, & (O_a = occ) \wedge (T_a = c) \\ 0, & else \end{cases} \tag{3.13}$$

   For an antecedent cell a that moves to cell c, the propagation of velocity is expressed as:

   $$P(V_c = v(a, c)|O^-, T) = P(O_c = occ|O_a^-, T_a) \tag{3.14}$$

where v is helper function that expresses velocity from cell a to cell c.

$P(X^-, R, T)$ is the joint distribution for transition. By assuming occupancy and transition are independent, we have

$$P(X^-, R, T) = P(O^-, V^-, R, T) \tag{3.15}$$
$$= \prod_{i \in N} P(O_i^-) \prod_{j \in N} P(T_j, V^-, R) \tag{3.16}$$

Since we assume *Occupancy preservation*, occupancy of an antecedent cell a must propagate to follower cells m. This introduces a normalization coefficient μ:

$$P(T_a = c, V^-, R) = \frac{P(T_a = c, V_a^-, R_{a,c})}{\sum_{m \in N} P(T_a = m, V_a^-, R_{a,m})} \tag{3.17}$$
$$= \mu_a P(V_a^-) P(R_{a,c}) P(T_a = c | V_a^-, R_{a,c}) \tag{3.18}$$

where

$$P(T_a = c | V_a^-, R_{a,c}) = \begin{cases} 1, & V_a^- = v(a,c) \wedge (R_{a,c} = reach) \\ 0, & else \end{cases} \tag{3.19}$$

2. **estimation**: The estimation step multiplies a prior distribution calculated from *prediction* with a conditional probability distribution that represents *correction* based on an *observation model*. We do not use any sensor for measuring velocity, and we assume cell's occupancy state can be extracted from raw sensor data after preprocessing. The observation model captures the uncertainty involved in sensor measuring given cell's occupancy state:

$$P(Z_{O,c} = z_c | X_C) = P(Z_{O,c} = z_c | O_C) \tag{3.20}$$
$$= \begin{cases} 1 - \omega_Z, & z_c = o_c \\ \omega_Z, & else \end{cases} \tag{3.21}$$

where $\omega_Z \in [0, 1]$ and higher its value, more confident we are in measurements.

To summarize, the *prediction* and *estimation* steps of BOF work recursively as shown in Figure 3.1. To calculate posterior probability distribution $P(X_c | Z_c)$, we need firstly

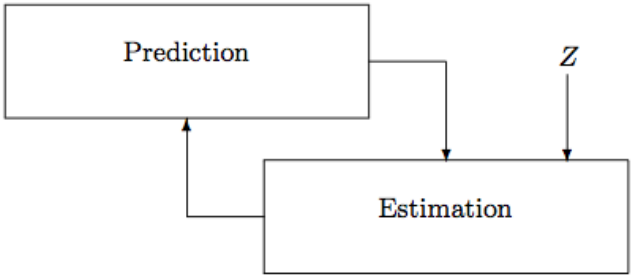

Figure 3.1: *The prediction and estimation loop of BOF [43]*

calculate the joint distribution in Equation 3.5 as:

$$\begin{aligned}\sum_{X^-,R,T} P(X_c, X^-, R, T, Z_c) &= P(Z_c|O_c) \sum_{O^-,V^-,R,T} P(V^-, R, T)P(O^-)P(O_c|O^-, T)P(V_c|O^-, T) \\ &= P(Z_c|O_c) \sum_{O^-,T} P(T)P(O^-)P(O_c|O^-, T)P(V_c|O^-, T) \\ &= \underbrace{P(Z_c|O_c)}_{correction} \\ &\quad \underbrace{\sum_{O^-,T} P(T)P(O^-)P(O_c|O^-, T)}_{occupancy\ prediction} \qquad (3.22) \\ &\quad \underbrace{\sum_{O^-,T} P(T)P(O^-)P(V_c|O^-, T)}_{velocity\ prediction} \qquad (3.23)\end{aligned}$$

As can be seen from above equations, $P(T)$ is needed for prediction. Based on Equation 3.18 and 3.19, we have:

$$\begin{aligned}P(T_a = c) &= \sum_{V^-,R} \mu_a P(V_a^-)P(R_{a,c})P(T_a = c|V_a^-, R_{a,c}) \qquad (3.24) \\ &= \mu_a P(V_a^- = v(a, c))P(R_{a,c} = reach) \qquad (3.25)\end{aligned}$$

Based on Equation 3.14 and 3.23, velocity prediction can be derived as:

$$\begin{aligned}P(\hat{V}_c = v(a, c)) &= \sum_{O^-,T} P(T)P(O^-)P(\hat{V}_c = v(a, c)|O^-, T) \qquad (3.26) \\ &= P(O_a^- = occ)P(T_a = c) \qquad (3.27)\end{aligned}$$

We then apply uncertainty in acceleration on the predicted velocity $\hat{V}$. The acceleration noise is assumed to be normal distribution with zero mean and covariance Σ. Therefore

the final velocity is predicted as:

$$P(V_c) = \sum_{\hat{V}_c} P(V_c|\hat{V}_c)P(\hat{V}_c) \tag{3.28}$$

where

$$v_c = \hat{v_c} + \omega_a \Delta t,\ \omega_a \sim \mathcal{N}(0, \Sigma) \tag{3.29}$$

For occupancy prediction, the probability of a cell being *not* occupied is firstly calculated. It is easier to calculate since only one case needs to be considered, namely when all possible antecedent cells do not move to that cell.

$$\begin{aligned} P(O_c = nocc) &= \sum_{O^-,T} P(O^-)P(T)P(O_c = nocc|O^-, T) && (3.30)\\ &= \sum_{O^-,T} \prod_{a\in N} P(O_a^-)P(T_a)P(O_c = nocc|O_a^-, T_a) && (3.31)\\ &= \prod_{a\in N} \sum_{O_a^-,T_a} P(O_a^-)P(T_a)P(O_c = nocc|O_a^-, T_a) && (3.32)\\ &= \prod_{a\in N} (1 - P(O_a^- = occ)P(T_a = c)) && (3.33) \end{aligned}$$

Therefore, the probability of a cell being occupied can be calculated as:

$$\begin{aligned} P(O_c = occ) &= 1 - P(O_c = nocc) && (3.34)\\ &= 1 - \prod_{a\in N} (1 - P(O_a^- = occ)P(T_a = c)) && (3.35) \end{aligned}$$

3.1.3 BOF with Motion Pattern (BOFMP)

In the above formulation of BOFUM, if occupancy propagates from an antecedent cell a to cell c with velocity $v(a, c)$, cell c will propagate the occupancy with the same velocity $v(a, c)$ along with zero-mean acceleration noise. In other words, BOFUM assumes a constant velocity motion model with uncertain zero-acceleration. In reality, object actually follows some motion patterns based on its location on the map. For example, a car driving on a lane are more likely to drive along the lane instead of driving to sidewalks. Therefore, Gindele et al. [15] incorporate this motion preference into the reachability matrix R.

For a cell c on the occupancy gird, they assign a terrain type for each cell with a helper function u. In their automotive application, they define three terrain types: $U = \{lane, sidewalks, unknown\}$. The reachability probability between cell a and cell c can be calculated as:

$$P(R_{a,c} = reach) = S_{u(a),u(c)}\omega(a, c) \tag{3.36}$$

where matrix $S \in [0,1]^{U \times U}$ defines the likelihood of change terrain types, and $\omega : N \times N \rightarrow [0,1]$ captures the likelihood of the movement when both cells are in the same terrain type or when the distance between these two cells has influence on the likelihood.

However, their way of modeling place dependent motion patterns has two disadvantages: 1) Manually classifying terrain types might be reasonable in their automotive applications, but it is not feasible in our human tracking scenarios. Although indoor environment, like offices and factories, can be classified as *walkable* or *not walkable*, this kind of classification does not provide too much information on motion preference. 2) As shown in Equation 3.36, the S matrix and weight function ω have to be manually defined. When an object is being tracked, the environment is updated rapidly, which requires to update the reachability matrix at the same time. Besides, when large maps are considered, it requires a lot of manual work to define function ω since any two cells on the grid map have to be considered. Therefore, defining reachability is very time consuming and inflexible.

To deal with the mentioned disadvantages, we propose to model motion patterns in a *learning-based* way. Although we apply this method only on modeling human motion patterns in indoor environment, it can be also used in other scenarios, such as modeling car motion pattern in outdoor environment. The idea is to train a neural network based on relevant data extracted from human trajectories. Once the network finished training, it learns the implicit motion patterns that are imposed on the human trajectories used for training. For a given gird map, the network is able to output data that can be interpreted as motion patterns without human intervention. This facilitates its application on real scenarios even in rapidly changing environments.

The relevant data we extracted from human trajectories can be used as an proxy of human motion patterns. They are represented as a set of conditional probabilities for every cell on the map. They represent how likely a person moves to one of the neighboring cells (exit direction), conditioned on which neighboring cell he or she comes from (entering direction). Mathematically, for any cell $c \in [1, N]$ on the gird map, they can be expressed as

$$P_c(V^{ex}|V^{en}) \tag{3.37}$$

where V^{ex}, V^{en} represent exit and entering direction respectively, and $V^{ex}, V^{en} \in \{U, L, R, D, DL, DR, UL, UR\}$ [1]. Since these probabilities incorporate information about the incoming cell, the motion model captures *spatial correlations* between cells. Besides, they are different for each cell based on cell's context on the map, which means the learned motion pattern is expressive enough to predict different motions at different locations. In other words, our motion model is also *place dependent*.

Figure 3.2 shows an example of how our model captures motion preference based on its location on the map. A person is currently at the location indicated by the green rectangle and arrives by taking entering direction of left. The most possible exit directions, as shown

[1] U: up, L: left, R:right, D:down

by red arrows in the left plot, show that this person is likely going either up-left or down-left in the next time step. This is expected since we can infer no bias towards either of the directions from the symmetric map. The motion model for the cell related to that location is represented by conditional probabilities shown in the right plot. It only shows $P_c(V^{ex}|V^{en} = L)$.

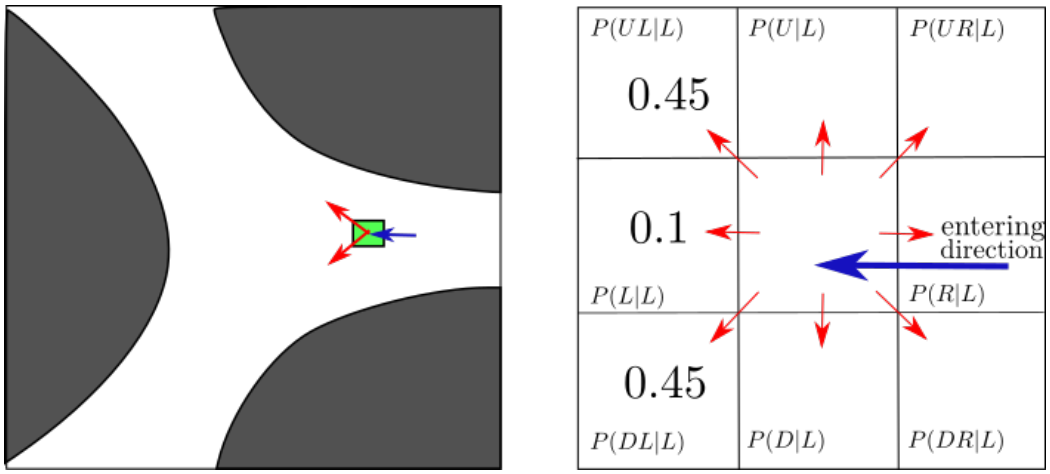

Figure 3.2: *An example of our proposed motion model.* ***LEFT****: A person is at location indicated by green rectangle. White area is walkable. The green arrow and red arrows show entering and exit directions.* ***RIGHT****: The motion pattern at that cell represented as $P_c(V^{ex}|V^{en} = L)$. It shows that, for the next time step, this person is more likely to turn either up-left or down-left than keep its entering direction of going left.*

The way how we incorporate our motion model into BOFUM similar to adding acceleration noise to predicted velocity in Equation 3.29. After the inferred velocity $\hat{V}$ is calculated based on process model as shown in Equation 3.27, the final velocity is calculated as:

$$P(V_c) = \sum_{\hat{V}_c} P_c(V^{ex} = V_c|V^{en} = \hat{V}_c)P(\hat{V}_c) \tag{3.38}$$

where $P_c(V^{ex}|V^{en})$ is *cell-specific* motion pattern. In this ways, the BOFMP knows how to propagate occupancies according to human motion pattern at certain locations. When compared with BOFUM, this method leads to better tracking performance especially when no observations are given or in occluded scenarios.

In this section, we assume the maximum velocity of the object is 1 *cell/timestep*. In reality, the objects may perform higher velocities. However, our neural network that extracts motion patterns models only velocity of 1 *cell/timestep*. The method of how we adapt our motion model to higher velocities will be explained in Section 4.4.

3.2 Convolutional Neural Network

Neural networks are mathematical abstraction about how our human nervous system process and learn from data. Over the last few years, neural networks have been extensively used in many applications, such as computer vision [39], natural language processing [8] and robotics [27]. Neural networks are organized as a layered structure, which replicates forward flow of information through human nervous system. Each layer consists of a certain number of *neurons*. In regular neural networks, those layers are *fully* connected. That is, neurons between two adjacent layers are pairwise connected, but neurons within a single layer have no connections with each other. For a single neuron, assuming its inputs are expressed by a vector $\mathbf{x}$, its parameters are weights $\mathbf{w}$ and bias b, and its output is a scalar value:

$$y = f(\mathbf{w}^T\mathbf{x} + b) \tag{3.39}$$

where f is a non-linear function, which is known as *activation function*. Popular choices for activation function include:

1. **Sigmoid function**: $f(x) = \frac{1}{1+e^{-x}} \in [0, 1]$. One undesired property of sigmoid function is that it gets saturated at both ends, which means the gradient for very small or very large inputs are almost zeros. In other words, sigmoid function "kills" gradient which may affect training of networks.

2. **Tanh function**: $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \in [-1, 1]$ Similar to sigmoid function, tanh function also saturates and "kills" gradients. However, its output is zero-centered.

3. **ReLU**: $f(x) = max(0, x) \in [0, +\infty)$. ReLU never saturates and this may accelerate convergence of network training as shown in [25]. Variants of ReLU include leaky-ReLU and PReLU [17].

To train a neural network, ground truth must be given since neural network is a supervised learning algorithm. Besides, a *loss function* has to be defined in order to measure how close are our network predictions to the ground truth. Commonly used loss functions are cross-entropy loss for classification and L2 squared norm for regression. Based on the loss calculated for training data, the network has to be optimized in terms of finding the best parameters of the network so that the loss is minimized. It turns out that following the (reversed) direction of gradients of the loss function is an effective way to update these parameters. Based on this observation, many optimization algorithms for neural networks have been proposed, such as Stochastic Gradient Descent (SGD), RMSProp and Adam (see more in [38]).

Convolutional Neural Networks (CNNs) are a special kind of neural network that works well with image data. In regular neural networks, the neurons for each layer is organized as a one-dimensional vector. While for CNNs, they are arranged in three dimensions: width, height and depth. For the input layer (i.e, the first layer), width and height are the spatial dimensions of an image and depth corresponds to the RGB channels of a color image. For

image classification, the output layer is $1 \times 1 \times N$ dimensional, with N corresponding to number of classes. CNNs are also organized as a stack of layers, and the most common layers are:

1. **Convolutional Layer** (CONV): The core layer of a CNN. CNNs learn the complex structure in image data by firstly extracting useful features, such as edges or a blotch of some color in shallow layers and semantic wheel-like or nose-like features in deeper layers. Those features are extracted by applying filters on 3D volume of data on each layer and this process is represented by a CONV layer. CONV layer utilizes *local connectivity*, which means they extract feature only on a local region of the input data. Thanks to its *parameter sharing* scheme, the number of parameters in CONV layers are much less than in layers of a regular neural network. The output volume of CONV layer depends on three hyperparameters: depth, stride and zero-padding.

2. **ReLU Layer** (RELU): This layer introduces non-linearities to the network and work as activation function as in regular neural networks. It applies ReLU function pointwise on the input volume.

3. **Pooling Layer** (POOL): This layer works as a downsampling operation that reduces the size of input data and ensures spatial invariance to some degrees. Typical pooling operation include max pooling and average pooling.

4. **Fully-connected Layer** (FC): As its name indicated, each of the neuron in FC layer is connected to all neurons in its preceding layer as in regular neural networks. For image classification tasks, the last layer is normally a FC layer.

5. **Batch-Normalization Layer**(BN): Batch normalization is a technique that has been proposed in [21] to address the so-called internal covariate shift phenomena in network training. Using BN layers allows higher learning rate and makes network less sensitive to initialization.

As an example, a simple CNN that used for image classification can be expressed as:

$$\text{INPUT} \rightarrow [\text{CONV} \rightarrow \text{BN} \rightarrow \text{RELU} \rightarrow \text{POOL}] * 2 \rightarrow \text{FC}$$

3.2.1 Densely Connected Convolutional Networks

In the development of CNNs, Simonyan and Zisserman [41] show that the depth of CNNs plays an very important role in its performance. However, the experiments from He et al. [18] show that with increased depth, the accuracy of network predictions firstly gets saturated and then degrades rapidly. In theory, the deeper network should perform at least as good as its shallower counterpart, because the redundant layers can at least learn an identity mapping, i.e., directly produce input as output. This unexpected performance degradation leads to their conclusion that CNNs are not good at learning identify mapping. To address this problem, they propose *residual* learning. Assume the underlying

desired mapping that represented by few stacked layers is $\mathcal{H}(\mathbf{x})$. Instead of learning it directly, its residual,

$$\mathcal{F}(\mathbf{x}) = \mathcal{H}(\mathbf{x}) - \mathbf{x}$$

is learned and a *skip connection* is connected from the input to the output of these stacked layers. Therefore, the final learned mapping is $\mathcal{F}(\mathbf{x}) + \mathbf{x} = \mathcal{H}(\mathbf{x})$. This idea is illustrated in Figure 3.3.

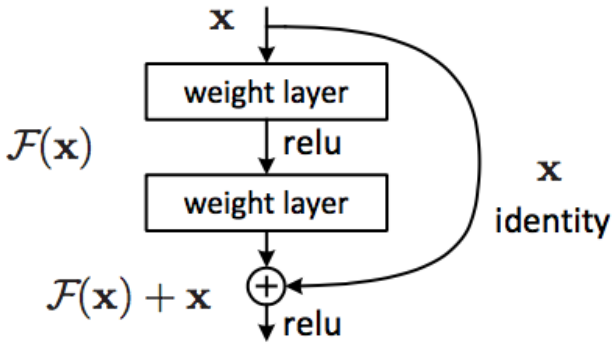

Figure 3.3: *An illustration of residual learning [18]. Instead of fitting the desired underlying mapping directly, a residual mapping $\mathcal{F}(\mathbf{x})$ is learned with few stacked layers.*

The advantages of residual learning is that it facilitates the gradient flow from deep layers to shallow layers through the skip connection. We define a composite function $H_l(\cdot)$ that can be represented by a few stacked layers such as BN, POOL, RELU and CONV. l is the layer index. Therefore, for the input $\mathbf{x}_{l-1}$, the output of residual learning from layer l is:

$$\mathbf{x}_l = H_l(\mathbf{x}_{l-1}) + \mathbf{x}_{l-1} \tag{3.40}$$

Huang et al. [20] propose a similar network structure that heavily uses skip connections. To encourage *feature reuse*, they concatenate feature maps from all preceding layers $0, \ldots, l-1$, which are then used as input of current layer l:

$$\mathbf{x}_l = H_l([\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{l-1}]) \tag{3.41}$$

A block of L layers that are densely connected in such a way is called as a *dense block*, and it has $\frac{L(L+1)}{2}$ connections. On example of a five layer dense block is shown in Figure 2.4. This special design of DenseNet leads to its state-of-the-art performance in image classification tasks.

3.2.2 Fully Convolutional DenseNet

The regular CNN are designed for image classification, where only one vector of class scores is needed from network output. In some other tasks such as semantic segmentation,

every pixed has to be classified into a certain class, i.e., the output must have the same spatial resolution as input image. This cannot be achieved by regular CNNs, since pooling operations are essential in networks but they also reduce spatial extent.

To achieve classification at pixel level, fully convolutional neural networks (FCNs) have been proposed by Jégou et al. [23]. By using only convolutional layers and *deconvolution* layers that functions as upsampling, their network is able to take arbitrary size image as input and predicts pixel-wise classification. A deconvolution operation is essentially *backward* convolution. Deconvolution, compared with interpolation, is a natural choice for upsampling in CNNs since it is learnable and thus facilitates end-to-end learning of the networks. How downsampling and upsampling is achieved in CNNs is illustrated in Figure 3.4

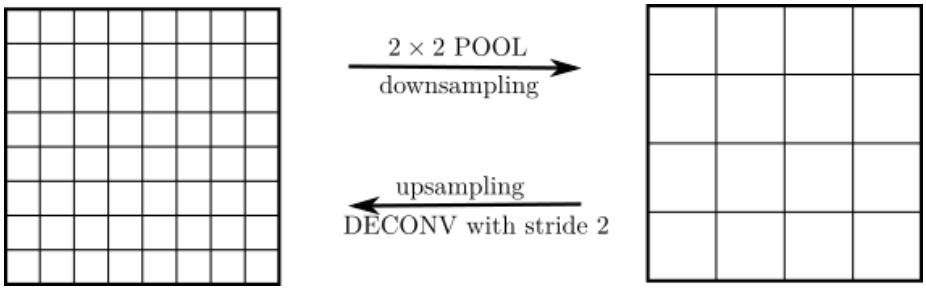

Figure 3.4: *Downsampling and upsampling in FCNs. Left represents a data volume with spatial size of* 8×8*. Downsampling with* 2×2 *pooling operation reduces its size to* 4×4*. Upsamping with deconvolution of stride 2 resumes its size.*

Jégou et al. [23] extends fully convolutional networks with dense blocks. Their proposed structure has a downsampling path extracting high-level semantic feature and an upsampling path recovering outputs to full resolution as input image. Their network structure, with some modifications, is used in thesis to extract motion patterns from static maps. As described in Section 3.1.3, we represent motion patterns as cell-specific conditional probabilities $P_c(V^{ex}|V^{en})$. Instead of predicting classification scores, the last layer of our network predicts $8 \times 8 = 64$ entries, which are interpreted as eight conditional probability distributions $P_c(V^{ex}|V^{en} = v)$. Each of these conditional probability distributions corresponds to one of eight entering directions.

Chapter 4

Implementation Details

In this chapter, the details on implementations will be described. In detail, how the ground truth data used for training network are generated from simulated human trajectories is described in Section 4.1. In Section 4.2, the details on training CNNs are provided. Section 4.4 introduces the code structure of implementing BOFMP. Finally, hyperparameters in the BOFMP tracking algorithm are introduced in Section **??**.

4.1 Human Trajectory Simulation

Neural networks are able to capture complex structures in data. For this reason, we train a neural network so that it can be used to extract human motion patterns from static maps. In order to make the network generalizes well, a large amount of human trajectory data in different indoor environments are needed. However, recording human trajectories with 2D laser scans in various indoor environments is expensive, since the hardwares have to be transported to different locations. Besides, since our model requires cell-specific motion patterns, the recorded human trajectories need to cover each discretized cell on the gird map. Even if it is feasible, this requires a lot of time for the laser scanner to work. To collect such a huge amount of real human trajectories is obviously out of the scope of this thesis. As a workaround, we simulated human trajectories on real-world SLAM-generated maps and those simulated trajectories are used for extracting human motion patterns. To validate our method, we evaluate our tracking algorithm on data recorded from real world.

The maps on which human trajectories are sampled from are floor plans of indoor environments such as laboratories and offices. We assume that human motions in these environment are only constrained by the spatial configurations. Other factors, such as time, social forces and different functional areas, are not modeled. In total, we simulate human trajectories on eight maps, with a total free space area of more than $6.6 \times 10^3\, m^2$. Two of those eight maps are shown in Figure 4.1. Each pixed of the map represents a cell

on its corresponding occupancy grid, and has resolution of 0.2 $m/pixel$.

Figure 4.1: *Two example maps on which human trajectories are sampled from. White areas are walkable, and black areas are desks or walls that human cannot walk through.*

We use A* algorithm to calculate human trajectory between two points on a map. The A* algorithm is a well-know algorithm for path planning. In each iteration of its main loop, A* algorithm decides which step to take based on the summation of two values: one value represents the cost from start to current node and the other represents a heuristic estimate of cost from current node to the goal. The path found by A* algorithm replicates human trajectory, since people normally have a clear goal location in their mind (thus a heuristic estimation) and take a path that requires least efforts to reach it. To account for the fact that human tend to walk some distances away from an obstacle (e.g., people walk in the middle areas of a corridor, which are distant to walls on both sides), we apply two Gaussian filters with different widths on the static maps to get cost maps.

The actual process from sampling trajectories to obtaining conditional probabilities as motion patterns are implemented in three steps:

1. Sampling trajectories on the full map. In order to get motion dynamics at all possible locations, we sample a predefined number of trajectories with every empty cell on the map as start location. The goal location for each trajectory is randomly sampled but the distance between start location and end location must be within a predefined

range. After the start and goal location are defined, a trajectory is calculated by A* algorithm.

2. Calculate probabilities based on trajectories. A *transition* is defined in this context as a person moves from cell a to c through b, i.e., $a \to b \to c$, where a and c are neighboring cells of b. If we consider eight neighboring cells, for each cell it has $8 \times 8 = 64$ different transitions. A helper function $v(c_1, c_2)$ is also defined to calculate the velocity from cell c_1 to c_2. Assume a cell c is identified by its coordinates on x, y axis, i.e, $pos(c) = (c_x, c_y)$, then

$$v(c_1, c_2) = \frac{pos(c_2) - pos(c_1)}{\Delta t}$$

Firstly a tensor C is initialized to store these transition counts. So far, Tensor C has 6 dimension of $32 \times 32 \times 3 \times 3 \times 3 \times 3$, with first two dimensions representing spatial size of a map window, next two dimensions representing V^{en} and last two dimensions representing V^{ex}. For example, for transition $a \to b \to c$, we increment the following entry of C:

$$C(pos(b), v(a, b), v(b, c))$$

For each sampled trajectory, we add every transition along this trajectory to its corresponding entry in C. The conditional probability $P_i(V^{ex}|V^{en})$ is then calculated from transition counts tensor C as:

$$P_i(V^{ex} = v_1 | V^{en} = v_0) = \frac{C(pos(i), v_0, v_1)}{\sum_v C(pos(i), v_0, v)} \tag{4.1}$$

3. Sampling map window with fixed size and data augmentation. The map window that we feed into our network is of size 32×32 cells (i.e., 6.4×6.4m). Therefore, we randomly crop a predefined number of map windows from the whole map. If a map window has free space less than 50%, it is not used. To increase the number of data samples, each map window is augmented eight times: for both the map window and its horizontal mirroring, they are rotated by 90°, 180° and 270°.

   So far, the ground truth of each map window has same dimensions as C, i.e, $32 \times 32 \times 3 \times 3 \times 3 \times 3 \times 3$. However, since the output of network has at most three dimensions, the ground truth has to be resized to $32 \times 32 \times 64$, without considering velocity $v = (0, 0)$.

The above algorithm is described in Algorithm 1.

4.2 Architecture of Neural Network

As described in Section 3.2, our network architecture is similar to that of Jégou et al. [23], with a minor difference in the last layer (i.e., the output layer). For their use in semantic

Algorithm 1 Algorithm for generating data for neural network.

```
1: procedure SamplingTrajectories(map, n)
2:     allTrajectories ← ∅
3:     for cell in emptyCells(map) do                ▷ loop over all empty cells on map
4:         trajectories ← ∅
5:         tries ← 0
6:         while tries < N do                        ▷ try to sample N trajectories for each cell
7:             start ← cell
8:             goal ← sampleGoalLocation(map, start)
9:             trajectory ← AStar(start, goal)
10:            if isValid(trajectory) then           ▷ check whether trajectory is empty
11:                add trajectory to trajectories
12:            tries ← tries + 1
13:        append trajectories to allTrajectories
14:    return allTrajectories
15:
16: procedure GetConditionalProbs(Trajectories)
17:    Initialize C with zeros
18:    for trajectory (c_1, ··· , c_n) in trajectories do   ▷ loop over all sampled trajectories
19:        for t = 2 : n − 1 do                      ▷ add transition c_{t−1} → c_t → c_{t+1}
20:            idx ← (pos(c_t), v(c_{t−1}, c_t), v(c_t, c_{t+1}))
21:            C(idx) ← C(idx) + 1
22:    probs ← calculateProbs(C)
23:    return probs
24:
25: procedure CropMapWindow(map, probs, S)
26:    data ← ∅
27:    count ← 0
28:    while count < S do                            ▷ try to get S samples
29:        loc ← sampleRandomLocation(map)
30:        window ← getWindow(map, loc)
31:        if freeSpace(window) < 0.5 then
32:            continue
33:        probWindow ← getProbWindow(probs, window)
34:        add (window, probWindow) to data
35:        add agument(window, probWindow) to data
36:        count ← count + 1
37:    return data
```

classification, the input $\mathbf{v}$ to the last year has dimensions of $W \times H \times N$. The first two dimensions are spatial size of the input image, and last dimension corresponds to number of semantic classes. The last layer, known as spatial softmax layer, applies the following function.

$$\varphi(\mathbf{v})_{x,y,j} = \frac{e^{\mathbf{v}_{x,y,j}}}{\sum_{n=1}^{N} e^{\mathbf{v}_{x,y,n}}}$$

In other words, the output of last layer are class scores that can be interpreted as probabilities. In our case, since the network has to output conditional probabilities $P_c(V^{ex}|V^{en})$ for all possible V^{en}, there are essentially eight probability distributions existing in $\mathbf{v}$. Therefore, our last layer has to firstly segment values to its corresponding V^{en}, then apply softmax accordingly.

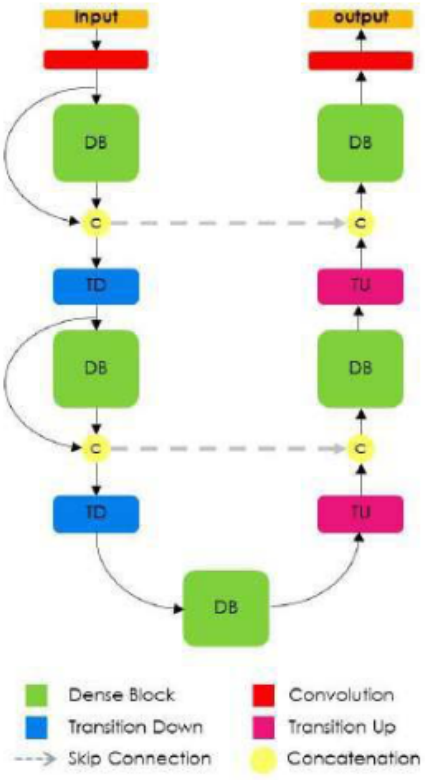

Figure 4.2: *Illustration of the fully convolution DenseNet in [23]. The network consists of a downsampling path extracting high-level semantic features and an upsampling path recovering outputs to full resolution as input image. The skip connections, as indicated by dashed lines, combines both deep coarse features and shallow fine features.*

The overall architecture of the network is depicted in Figure 4.2, and how each component is structured is summarized in Table 4.1. For the network that we use in this thesis, it has 31 convolutional layers, 13 in the downsampling path, 5 in the bottleneck dense block, and

13 in the upsampling path. Two max pooling layers are used to downsample the spatial size of inputs, and two deconvolution layers are used to recover outputs' resolution. Each dense block consists of 5 layers. One important parameter for dense block is the *growth rate*, which is the number of feature maps in each layer. We use a growth rate of 12. Thus, the output of each dense block has $5 \times 12 = 60$ feature maps. The number of feature maps at the end of blocks is listed in Table 4.2. The network has in total 448,648 parameters.

Component	Structure
Transition Down (TD)	BN $\rightarrow$ RELU $\rightarrow$ 1×1 CONV $\rightarrow$ 2×2 MAXPOOL
Transition Up (TU)	3×3 DECONV with stride 2
Layer	BN $\rightarrow$ RELU $\rightarrow$ 3×3 CONV
Dense Block (DB).	5 Layers that are densely connected

Table 4.1: *Structure of each component in Figure 4.2.*

Blocks	Number of feature maps
Input	1
CONV + DB + C	68
TD + DB + C	128
TD	128
DB + TU + C	188
DB + TU + C	128
DB + CONV	64
Softmax	64

Table 4.2: *Number of feature maps at the end of blocks.*

4.3 Preprocessing of Tracking Data

To evaluate our method, we apply the BOFMP filter on real data that are recorded by laser scanners. The real data records occupancy information on a static map which changes over time. Since occupancy indicates a person's location on the map, the time-varying occupancy is able to represent human trajectories. We define the real data that records occupancy information on a given map window for a certain amount of time as a **scene**. After preprocessing, each scene records human trajectories on that map window so that we can evaluate our method on. However, due to various reasons, such as localization error of robots and discretization errors, the real data could be very noisy. Therefore, a preprocessing pipeline is proposed. It consists of following steps:

1. **Downsample**. The real data are recorded by laser scanners with frequency of 12Hz, which means we get 12 frames of data per second. Human walking speed in office is roughly 1$\sim$1.2 m/s. Since the map resolution is 0.2 m, this means for moving one cell in map it takes roughly 2$\sim$2.4 frames. Therefore, We decide to downsample real data by merging the occupancy in 3 continuous frames to 1 frame.

2. **Remove static occupancy**. A scene is supposed to show the *dynamic* changes of occupancy which represent human trajectory. Therefore, if a cell is occupied over a quarter of the time in a scene, it is likely to be sensor failure and therefore should be removed.

3. **Remove occupancies near walls**. According to our observations on the recorded data, it is very likely that the occupied cells near walls are false positive due to discretization errors. Therefore, we remove occupancies that are one cell away from walls.

4. **Remove outliers.** A person in the real data are represented as a blob of occupied cells. Therefore, if there is a single occupied cell, it is thought to be sensor noise and thus should be removed.

Figure 4.3 shows a scene at certain time steps. The upper plot shows the original data before preprocessing and the lower shows after preprocessing. One can see that, after preprocessing, the noisy occupancies are removed without affecting the dynamics of human motion.

Even after preprocessing, we found there are still a lot of scenes which do not show proper occupancy information. For example, there might be no occupied cells for a long period of time. On the other hand, classification on trajectories with different motions (e.g., going straight or making a turn) might be interesting for future work. Therefore, we developed a small application to filter out useless scenes and classify scenes into a set of predefined motion classes based on the underlying trajectory. A screenshot of the application is shown in Figure 4.4.

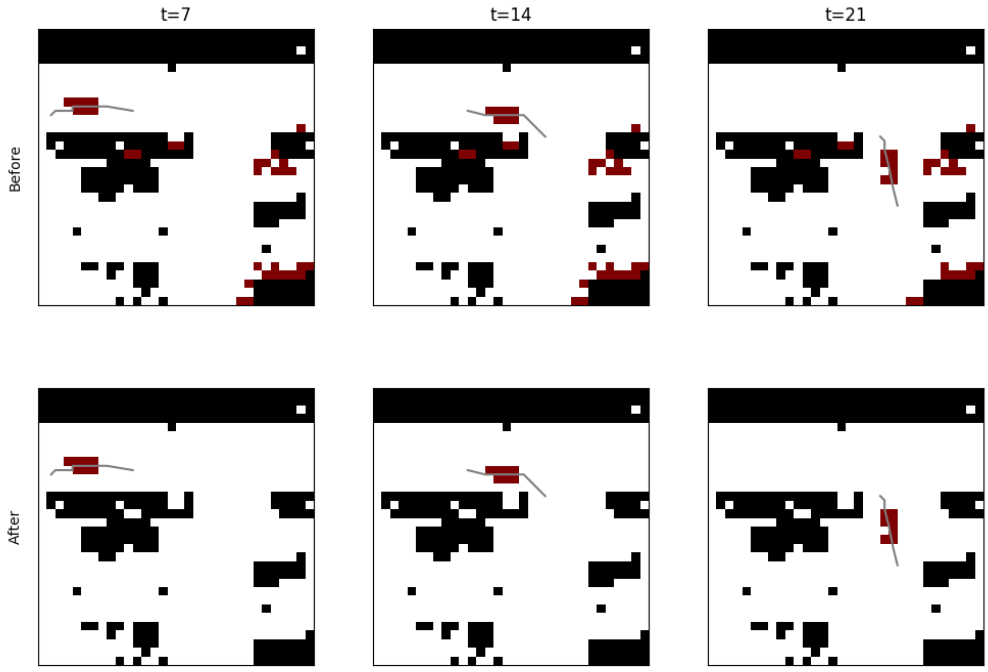

Figure 4.3: *One example of preprocessing of real data. Black cells are from the static map. Red and white cells represent occupied and not occupied respectively. A person is identified by a blob of occupied cells and the gray line indicates the trajectory.* ***Upper****: The real data recorded by laser scanners.* ***Lower****: The corresponding data after preprocessing. One can see that, the noisy occupancies are removed without affecting the dynamics of human motion.*

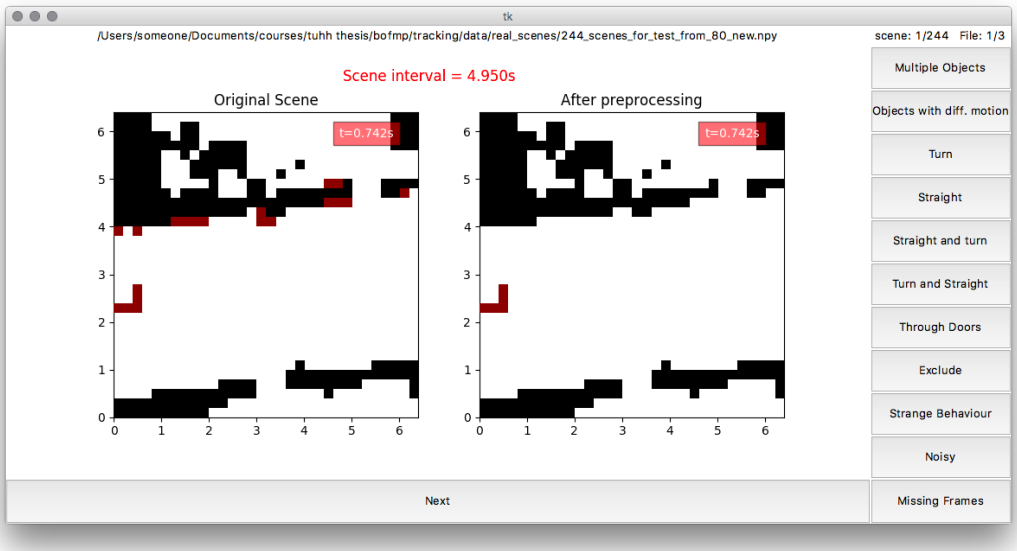

Figure 4.4: *Application used for labeling scenes. User can define their own classes, which are shown as buttons on the right. On the left, both the original scene and preprocessed scene are displayed as animations. User can continue labeling by clicking "Next" button.*

4.4 Implementation of BOFMP

4.4.1 Extension of Network Output

As described in Section 4.1, the motion pattern extracted from network output considers max speed of 1 *cell/timestep*. However, since human does not always walk in a constant speed, we need to adapt our motion model to higher speeds. This is done by applying Gaussian blur in acceleration space. For now, we omit the cell index c, and the motion pattern from network output is represented as:

$$P_{NN}(v^{ex} = (v_x^{ex}, v_y^{ex})|v^{en} = (v_x^{en}, v_y^{en})) \tag{4.2}$$

where $v_x^{ex}, v_y^{ex}, v_x^{en}, v_y^{en} \in [-1, 1] \cap \mathbb{Z}$. We define the *extent* as the number of values that velocity can take in either x or y axis. Due to its symmetry, the max speed for extent e is $\lfloor \frac{e}{2} \rfloor$ cell/step. Therefore, P_{NN} has an extent e_n of 3 and dimension of $3 \times 3 \times 3 \times 3$.

Let us assume that we want motion model for extent e_w. This is achieved by firstly padding P_{NN} to dimension $e_w \times e_w \times e_w \times e_w$ with zeros, s.t., $v_x^{ex}, v_y^{ex}, v_x^{en}, v_y^{en} \in [-\lfloor \frac{e_w}{2} \rfloor, \lfloor \frac{e_w}{2} \rfloor] \cap \mathbb{Z}$. Then we can calculate the probabilities for acceleration conditioned on entering velocity:

$$P_{NN}(a|v^{en}) = P_{NN}(v^{ex} = v^{en} + a|v^{en}) \tag{4.3}$$

With a 4-dimensional Gaussian kernel $K(a, b) \sim \mathcal{N}((a, b), \Sigma)$, we have:

$$P(a|v^{en}) = \sum_{\hat{a}, \hat{v}^{en}} P_{NN}(\hat{a}|\hat{v}^{en}) \times K(\hat{a}, \hat{v}^{en}) \tag{4.4}$$

Finally we get motion pattern $P(v^{ex}|v^{en})$ by restoring to velocity space and normalization:

$$\hat{P}(v^{ex}|v^{en}) = P(a = v^{ex} - v^{en}|v^{en}) \tag{4.5}$$

$$P(v^{ex}|v^{en}) = \frac{\hat{P}(v^{ex}|v^{en})}{\sum_{v^{ex}} \hat{P}(v^{ex}|v^{en})} \tag{4.6}$$

One example is shown in Figure 4.5. As can be seen from the figure, the original motion pattern $P_{NN}(v^{ex}|v^{en}$ has max speed of 1 *cell/step*, while for the extend one $P(v^{ex}|v^{en})$ the max speed is 2 *cell/step*. From the extended motion pattern $P(v^{ex}|v^{en})$, one can see that only the directions of velocity change but not the speed. This implies that even if a person changes his or her walking direction, the walking speed is normally not changed.

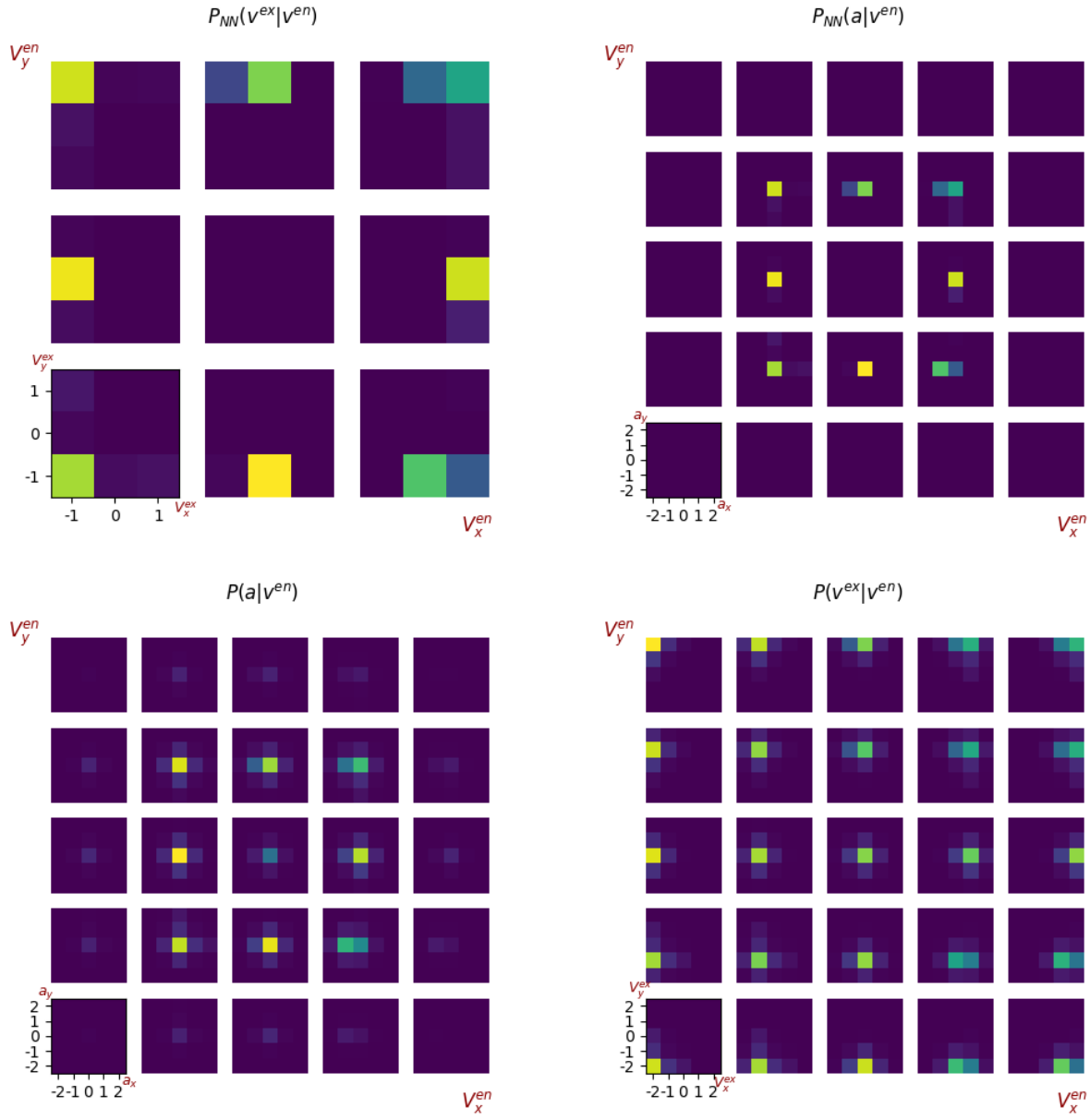

Figure 4.5: *One example of extending motion pattern from network output to higher extent. The definitions of these quantities shown in figure are explained in Equation 4.2 ∼ 4.6. The original motion pattern $P_{NN}(v^{ex}|v^{en})$ has max speed of 1 cell/step, while for the extend one $P(v^{ex}|v^{en})$ the max speed is 2 cell/step. This is achieved by applying Gaussian blurring in acceleration space. From the extended motion pattern $P(v^{ex}|v^{en})$, one can see that only the directions of velocity change but not the speed (in fact, with a small probability to accelerate or decelerate). This implies that even if a person changes his or her walking direction, the walking speed is normally not changed.*

4.4.2 Workflow of BOFMP

Before introducing the workflow of our method, some parameters that used to initialize BOFMP must be clarified:

1. **extent** e. As introduce in 4.4.1, extent is defined as the number of values that velocity can take in either x or y axis. For example, if extent is 7, the velocities on both x and y axis must be integers within range $[-3, 3]$. The extent determines the maximum speed speed our algorithm concerns.

2. **variance** δ^2. For BOFUM, this parameter refers to the variance of the Gaussian distributed acceleration noise in Equation 3.29. For BOFMP, it is the variance of the Gaussian kernel used for blurring in Equation 4.4. In both cases, this parameter reflects the algorithm's belief on how likely an object accelerates or decelerate.

3. **omega** Ω. In the correction step of BOF filters, measurements from sensors are taken into account for estimating the posterior probability. Since sensor could be noisy, this parameter determines how much does the filter trusts the measurements. The sensor used for in our tracking application is laser scanners, which is rather physically reliable and therefore has a low Ω value.

A trained CNN model that used for extracting motion pattern from static map must be available. With the above parameters being defined and the CNN model being ready, the BOFMP filter can be applied. Firstly, feed the static map to the network so that $P_{NN}(V^{ex}|V^{en})$ is retrieved. It is then extended to the desired extent e to get motion pattern $P(V^{ex}|V^{en})$. Before the filtering starts, the occupancy and velocity probabilities are initialized uniformly, i.e. for $c = 1 : N$,

$$P(O_c = occ) = P(O_c = nocc) = 0.5 \tag{4.7}$$

$$P(V_c = v) = \frac{1}{e^2} \tag{4.8}$$

After initialization, the actual tracking can start by recursively executing tracking step. A tracking step is composed of two steps: *prediction* and *estimation*. In prediction step, a prior distribution of the state of cells is derived. In estimation step, a posterior distribution is calculated by multiplying the prior distribution with a conditional probability distribution that represents correction based on incoming measurements. However, the measurement gives information about only whether a cell is occupied or not at each time step. In order to make proper predictions for the next time step, the filter has to infer velocities for each cell. To get some intuitions, Figure 4.7 shows an example of tracking step. The complete workflow is illustrated in Figure 4.6.

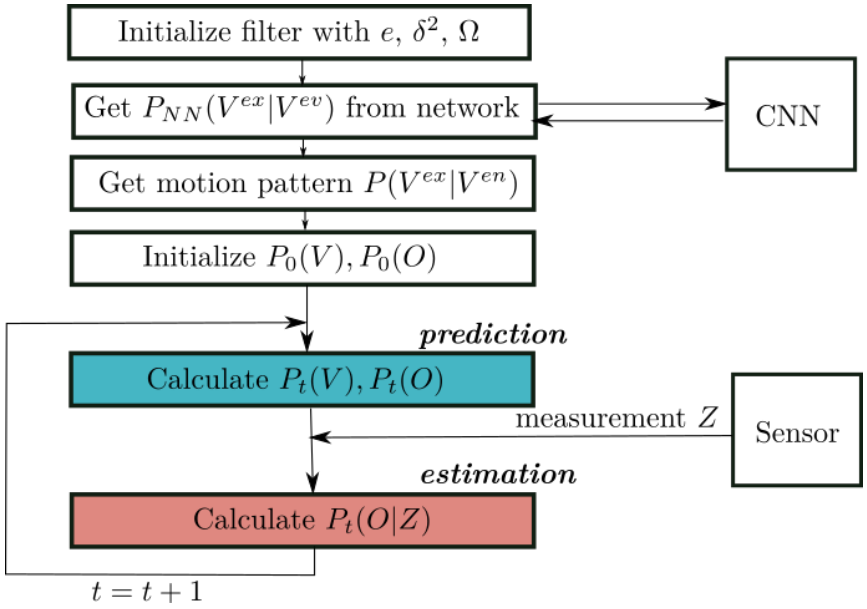

Figure 4.6: *Workflow of BOFMP filter. The parameters of the filter have to be defined firstly. The motion pattern probabilities are obtained from the outputs of a pre-trained CNN. The state of cells are initialized with uniform distributions and then recursively updated by prediction and estimation step. Measurements from sensor are used for estimation.*

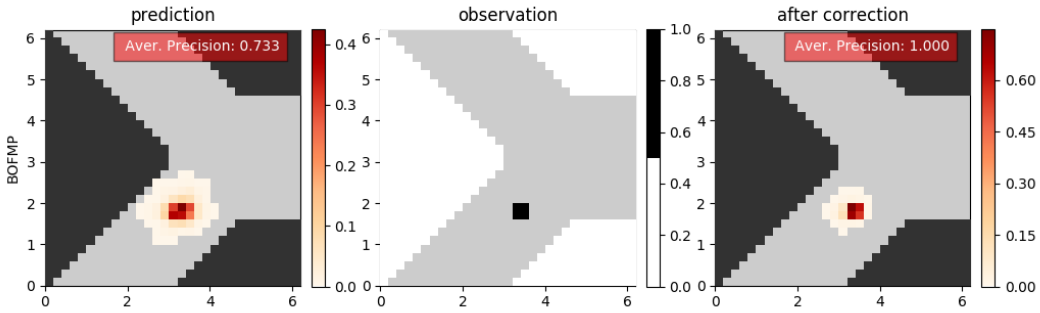

Figure 4.7: *One filtering step of BOFMP filter. At last time step, the person goes from up to down. Based on motion pattern, BOFMP predicts that there are possibilities that this person will turn to left-down and keep going downwards. After measurement shows that this person still goes downwards, BOFMP corrects its predictions and attenuates probabilities of turning left-down. Since the measurement adds additional information for BOFMP to make better predictions, the average precision increases after correction step.*

4.4.3 Code Structure

The functionalities of the code concern mainly three parts:

1. **Tracking filters**. Both BOFMP and the baseline BOFUM filters are implemented. Besides, based on the data used, i.e., either simulated data or real data, the implementation has to be different.

2. **Visualization of tracking**. To have a better idea about how tracking performs, visualization of tracking for each time step is necessary. We implemented two ways of visualization. One is to show tracking step by step as prediction and correction with static plots. The other shows the tracking process with animation.

3. **Evaluation on tracking performance**. In order to compare our method with its baseline, evaluation on tracking performance is necessary. The metric used for evaluation is described in Section 5.1.

We follow an object-oriented programming paradigm. By defining classes and subclasses, this facilitates code reuse and better structure of the code. Figure 4.8 shows the main classes defined in the code. They are grouped based on their functionalities.

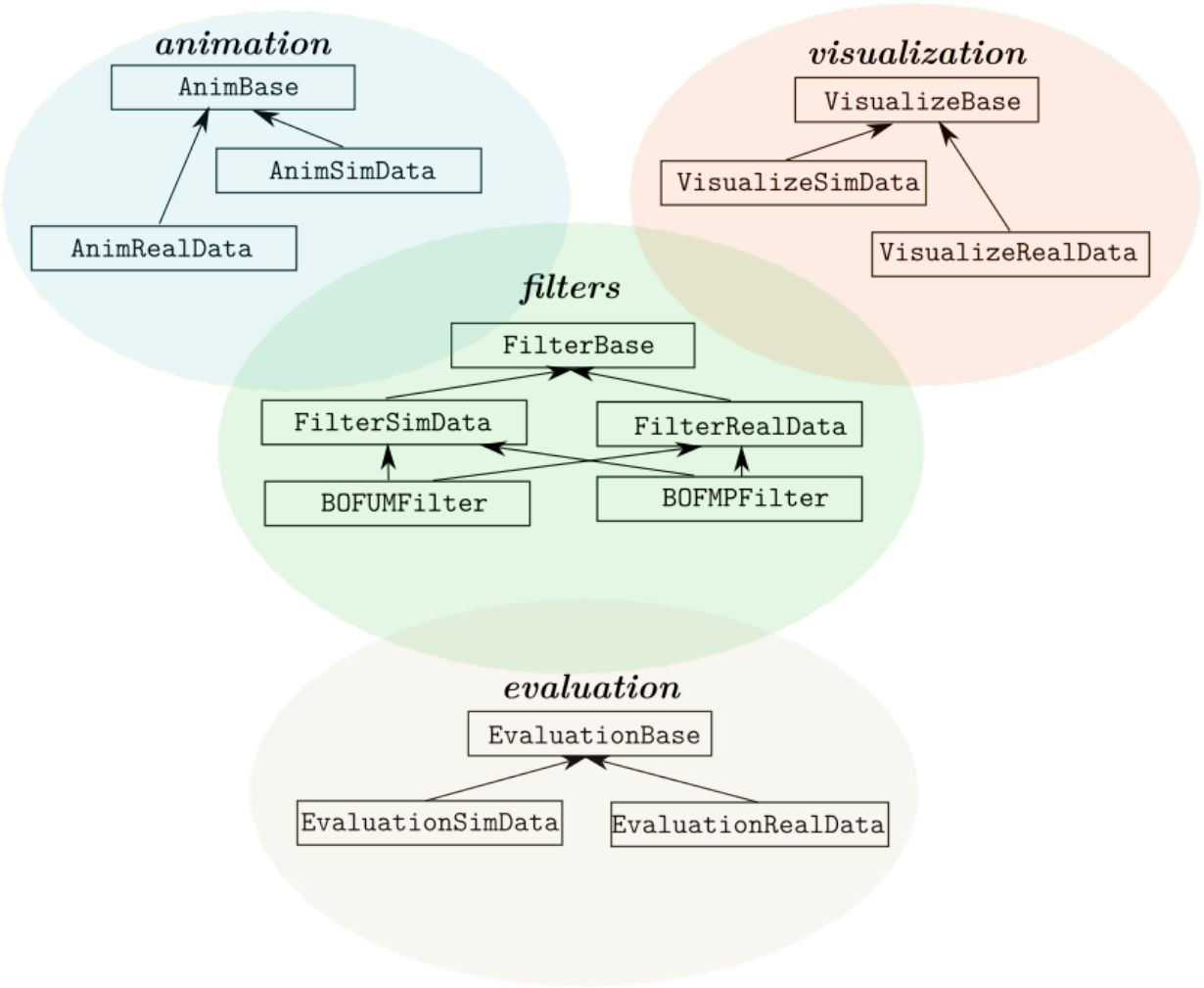

Figure 4.8: *Classes defined in the code. Classes are grouped according to their functionalities. The "filters" is shown in the center since it is used by other classes. The arrows show class inheritance. For example,* `ClassA` → `ClassB` *indicates that* `ClassA` *is inherited from* `ClassB`.

Chapter 5

Results and Discussions

5.1 Metrics

5.1.1 Metrics for Neural Network

5.1.2 Metrics for Tracking

The possible metrics that could be used for measuring the consistency between occupancy predicition and ground truth are: *cross entropy*, *f1 score* and *average precision*. We choose average precision as our metric and the reasons will be detailed in the thesis.

5.2 Training of Neural Network

After the data is collected as described in Section 4.1, we split them into training, validation and test set. The training and validation set are sampled from seven full maps, and test set is sampled from a separate map. The number of samples we generated are summarized in Table 5.1.

Figure 5.1 shows one example of map window as network input and its ground truth. On the left, a cell is labeled with red color, and the sampled trajectories that go through this cell are displayed. On the right, it shows a visualization of the ground truth $P_c(V^{ex}|V^{en})$ for that red cell. The axes show velocities on x, y directions. Since we assume a person has maximum speed of 1 *cell/timestep*, the velocities must be in range $[-1, 1]$.Outer axes represent entering direction V^{en}, and inner represents exit velocity V^{ex}. One can see that $P(V^{ex} = DL|V^{en} = DL) = 0.13$ and $P(V^{ex} = D|V^{en} = DL) = 0.87$ [1]. This indicates that

[1]DL represents going down-left, and its corresponding velocity is (-1, -1). D represents going down,

	training	validation	test
number of samples	27,119	4,785	3,760

Table 5.1: *Number of samples in training, validation and test set.*

if a person reaches the red cell by taking direction down-left (i.e., coming from upper-right neighboring cell), it is very likely that he or she will change to going down for the next time step. This proves that our way of modeling motion pattern captures human motion dynamics.

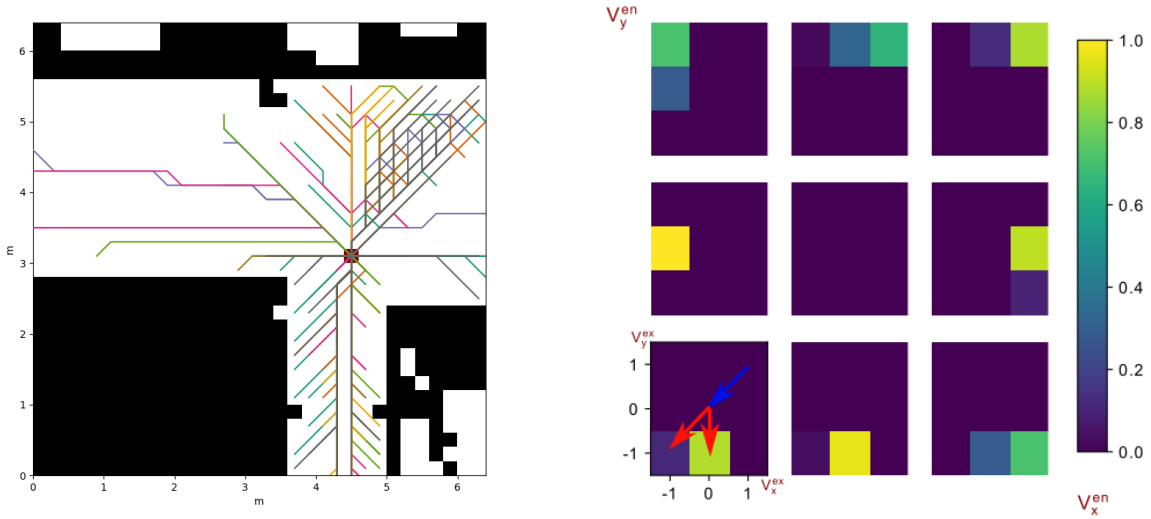

Figure 5.1: *one example of map window as network input and its ground truth* ***Left****: The map window has size of* 32×32 *cells, with resolution of* $0.2m/cell$*. It also shows trajectories that goes through the red cell.* ***Right****: Visualization of conditional probability* $P_c(V^{ex}|V^{en})$ *for the red cell on left map. It can be seen that if a person reaches that cell by taking down-left direction, it is very like that he or she will change direction to going down.*

.

Practically, the hyperparameters of a neural network, such as learning rate and weight decay, are optimized by random search. However, due to time limitation, this step is skipped in this thesis work. Instead, we use the hyperparamters from a similar network (the only difference is the output layer) that is optimized for predicting average human occupancy on static map. The network is trained with mini-batches of size of 128, and is optimized with Adam optimizer [24]. The training runs for 100 epochs, with early stopping

and its velocity is (0, -1).

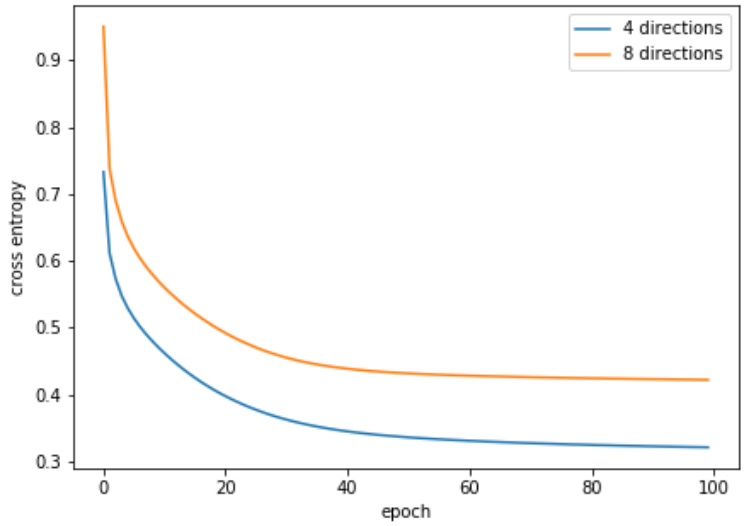

Figure 5.2: *Cross entropy loss during training. The networks are trained to learn conditional probabilities $P_c(V^{ex}|V^{en})$. The blue line shows training dynamics using data generated with 4 directions "left, right, up and down". The yellow line also takes diagonal directions into account, which has 8 directions in total.*

patience of 15 epochs. The best model is selected based on the KL-divergence on validation data.

Figure 5.2 shows the cross entropy loss during training. We trained networks for two cases: 4 directions and 8 directions. The 4-direction case considers only "left, right, up and down". Therefore, the dimension of its output is $32 \times 32 \times 16$. The 8-direction case considers also diagonal directions, therefore the output dimension is $32 \times 32 \times 64$. The loss curves show that the hyperparameters are set correctly since the loss decreases exponentially in the early stage and keeps decreasing afterwards. Naturally, more directions implies higher complexity, thus the overall loss for yellow line is higher than blue line. The cross entropy on the test set is listed in Table 5.2.

networks	cross entropy
4-direction	0.334
8-direction	0.383

Table 5.2: *Cross entropy loss on test data.*

5.3 Evaluation of Tracking Performance

Our method BOFMP is proposed for human tracking in indoor environment. It is similar to BOFUM, and we claim it improves BOFUM since it is able to predict occupancy according to human motion pattern. To verify our claim, we compare the performance of BOFUM and BOFMP in two scenarios:

1. **tracking stage**. Before a certain time point t_{lost}, measurements are given at each time step, i.e., $t = 1 : t_{lost} - 1$. We evaluate consistency between occupancy prediction and the ground truth at every time step.

2. **future prediction stage**. From time point t_{lost} on, the measurement is no longer given. Then we evaluate occupancy predictions with ground truth for the next n time steps.

These two stages can be illustrated as:

$$\text{start} \rightarrow \text{tracking stage} \xrightarrow{\text{measurement lost}} \text{future prediction stage} \rightarrow \text{end}$$

In this thesis, we set $t_{lost} = 9$ and $n = 8$, i.e., both tracking and future prediction stage lasts for 8 time steps. The parameters of filters are tuned based on their performance in *future predictions* stage.

5.3.1 Overview of Datasets

Simulated Dataset

We generated 500 scenes as training set for tuning filter parameters and 500 scenes for test set. Training and test data are generated from different maps. In these scenes, there are either one or two walking humans.

Real Dataset

Although our neural network are trained on simulated human trajectories, we expect that our method outperforms BOFUM on real tracking scenes. We record human trajectories on two different maps by a laser scanner mounted on a robot. After processing raw laser data, we get 500 tracking scenes for training from one of the maps and 244 scenes for test from the other.

5.3.2 Tracking on Simulated Data

For both filters, we randomly sample 100 sets of parameters from value ranges listed in Table 5.3, apply them on scenes in the training set and calculate average precision for

every time step. The best set of parameters for each filter are selected based on the mean of average precision in the future prediction stage.

	value range for simulated data
e	$\{3, 5, 7\}$
δ^2	$[0.1, 0.6]$
Ω	$[0.01, 0.2]$

Table 5.3: *Value ranges of filter parameters for simulated data.*

We firstly tuned the parameters on training set for BOFUM and BOFMP individually. Then we apply both filters with their best parameters on test set. The best set of parameters and its corresponding average precision from $t = 9$ to $t = 16$ on test data are listed in Table 5.4.

	e	δ^2	Ω	average precision for $t = 8 : 16$	mean
BOFUM	7	0.556	0.0164	0.767 0.599 0.500 0.416 0.349 0.279 0.232 0.192	0.417
BOFMP	7	0.373	0.0353	0.785 0.637 0.552 0.479 0.416 0.358 0.311 0.252	0.474

Table 5.4: *Best parameters for BOFUM and BOFMP on simulated data.*

Figure 5.3 shows mean of average precision on 500 scenes in test set for every time step. The average precision for both filters start with values close to zero. Since we highly trust our measurements (low Ω), both filters are able to successfully track the objects within 3 time steps. The fact that average precision keeps a high value from $t = 3$ to $t = 8$ indicates that both filters can predict very well for the *next immediate* time step. Starting from $t = 9$ (see the red dash line), measurements are no longer given. The prediction is still accurate for the next time step ($t = 9$), but decreases progressively over time. This is expected, since without measurements, the state of world becomes more and more uncertain. Even though, we can see that our BOFMP have a higher average precision value than BOFUM at almost every time step, which indicates the improvements of our method in both tracking and future prediction stages.

Figure 5.4 shows how BOFUM and BOFMP perform tracking on one scene from test data. In this example, a person is walking from the lower door towards upper door. The gray curve shows the trajectory of the person. At $t = 8$, the person walks upwards with velocity of 1 *cell/timestep*. Both algorithms track the person very well with average precision of 1.0. At $t = 9$, the measurement is lost and the future prediction stage of tracking starts. At $t = 10$, BOFUM predicts that the person will continue to go upwards, with a low possibility going other directions. However, since BOFMP knows there is a door in the top right corner, it predicts that the person is also very likely going to that door, and thus turns to upper right. At later time steps, BOFMP continues to predict occupancy towards upper door as well as other possible directions (i.e., door on the left and empty

space on the right). On the contrary, BOFUM still propagates most occupancies upwards. At $t = 16$, since the object accelerates to higher velocity, most of occupancy predictions of BOFMP are left behind the tracking object.

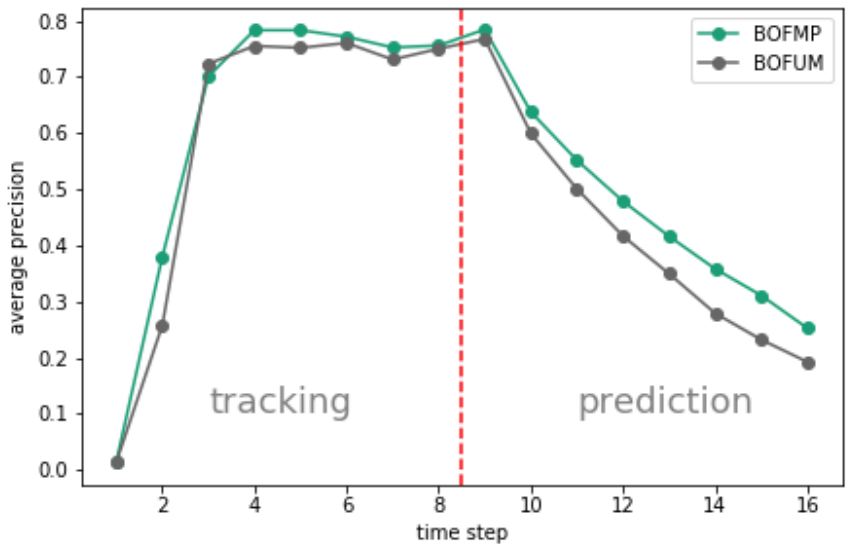

Figure 5.3: *Evaluation results on simulated test data. The tracking stage lasts from $t = 1 : 8$, and future prediction starts from $t = 9$. One can see that in both stages, our proposed BOFMP outperforms BOFUM for almost every time step.*

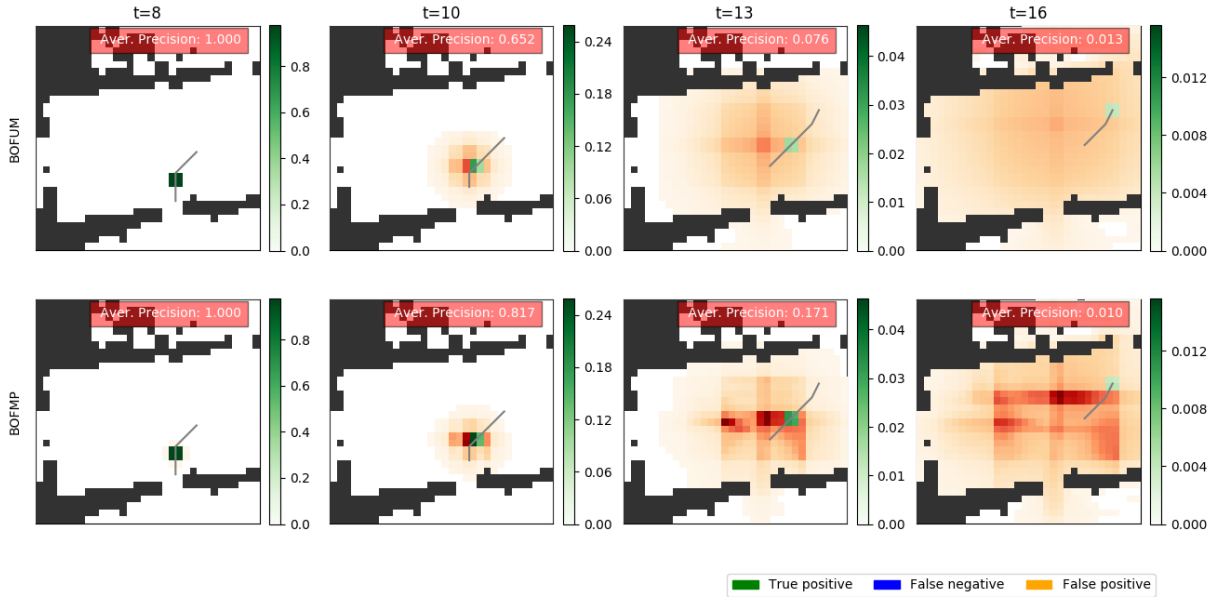

Figure 5.4: *One example of tracking case from test data. A person is walking from the lower door to the door in the upper right corner. The measurement is lost at $t = 9$. For BOFUM, it predicts most occupancies going upwards. For our BOFMP, since it has knowledge on the human motion model, it predicts occupancies to all possible directions, such as going to the door in the upper right corner, the door on the left and empty space on the right.*

5.3.3 On real data

Table 5.5 shows the value ranges from which the filter parameters are sampled. Note that value range of noise variance δ^2 for real data is higher than that for simulated data. This is because in real data, there are higher uncertainties with human motion and sensor failures.

	value range for real data
e	$\{3, 5, 7\}$
δ^2	$[0.2, 0.7]$
Ω	$[0.01, 0.2]$

Table 5.5: *Value ranges for filter parameters on real data.*

Spatial blurring of motion probabilities

The simulated trajectories do not always reflect real human motion patterns. This is because, as shown in Figure 5.5, people are more flexible to decide when and where to make turns.

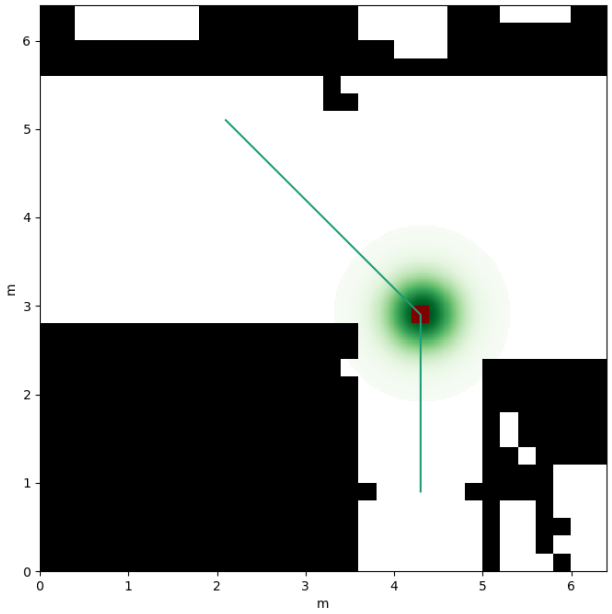

Figure 5.5: *A turn on simulated human trajectory. In order to reach the goal location in upper left corner, the simulated trajectory shows that a person will make a turn from going up to going up-left at location indicated by the red square. However, in real scenarios, a person is more flexible in deciding where to make that turn and he might turn at any location in the green area.*

Therefore, to better adapt to real data, we introduce a technique that blurs the motion pattern probabilities $P_c(V^{ex}|V^{en} = v\}$ of a cell c *spatially* into its neighbors if a *turn* is detected. A *turn* on cell c for velocity v^{en} is defined as:

$$turn(c, v^{en}) = \begin{cases} True, & \arg\max_v(P_c(V^{ex} = v|V^{en} = v^{en}) \neq v^{en} \\ False, & \text{otherwise} \end{cases}$$

For each detected turn at cell c, we add its motion probability $P_c(V^{ex}|V^{en} = v^{en})$ to that of its neighboring cells in a Gaussian blurring way. Intuitively, this means if a person makes a turn at a certain cell, it is also likely that people make that turn somewhere near that cell. As a consequence, another two parameters, blur extent $blurExt$ and blur variance $blurVar$, are introduced and their value ranges are listed in Table 5.6. If a turn is detected on cell c for velocity v^{en}, for its neighboring cell n that is less than $blurExt$ cells away from c,

$$P_n(\cdot|v^{en}) = P_n(\cdot|v^{en}) + P_c(\cdot|v^{en}) \times G(pos(n) - pos(c)) \tag{5.1}$$

where $G \sim \mathcal{N}(\mathbf{0}, \Sigma)$ is a 2-dimensional discrete Gaussian kernel.

	value range	*note*
$blurExt$	$\{3, 5, 7, 9\}$	determines how far the Gaussian blurring can reach
$blurVar$	$[0.5, 2]$	variance of the Gaussian kernel used for blurring

Table 5.6: *Parameters introduced by spatial blurring and their value ranges.*

Motion keeping for future prediction

Our proposed BOFMP, just like BOFUM, is *memory-less*. That is to say, the last time step of tracking stage has absolute influence on future predictions, and the time steps before the last step has no influence at all. In the real data we recorded, a time step equals to 0.25 second in real time. However, this short period of time is not able to summarize a person's motion in the past time steps. Therefore, we propose to add moving average velocity of last few times steps to the predicted velocity $P(V_{pred})$ (which is $P(\hat{V})$ in Equation 3.29), and then calculate next velocity $P(V)$ from it. This process of incorporating moving average velocity is called as *motion keeping*. It starts from the beginning of future prediction stage ($t = 9$ in our setting) and the influence of moving average velocity is exponentially decreased for the later time steps.

Motion keeping also introduces new parameters. Assuming the measurement is lost since time step t_{lost}, the new parameters are:

1. **window size** w. It determines how many last time steps are considered when calculate moving average velocity .
2. **initial motion factor** $initMF$. This coefficient determines how much of moving average velocity $P(V_{ma})$ is incorporated into predicted velocity $P(V_{pred})$ for the first time step of future prediction stage (i.e., $t = t_{lost}$),.
3. **keep motion factor** $keepMF$. This constant is the base of the exponential function used to decrease moving average velocity $P(V_{ma})$ for the later time steps. Therefore, for $t \geq t_{lost}$,

$$factor = initMF \times keepMF^{t-t_{lost}} \tag{5.2}$$

$$P(V_{merge}) = factor \times P(V_{ma}) + (1 - factor) \times P(V_{pred}) \tag{5.3}$$

The value ranges of these parameters are listed in Table 5.7. One example of tracking on real scene with motion keeping is shown in Figure 5.6.

	value range
w	$\{2, 4, 6\}$
$initMF$	$[0.3, 0.8]$
$keepMF$	$[0.3, 0.8]$

Table 5.7: *Parameters introduced by motion keeping and their value ranges.*

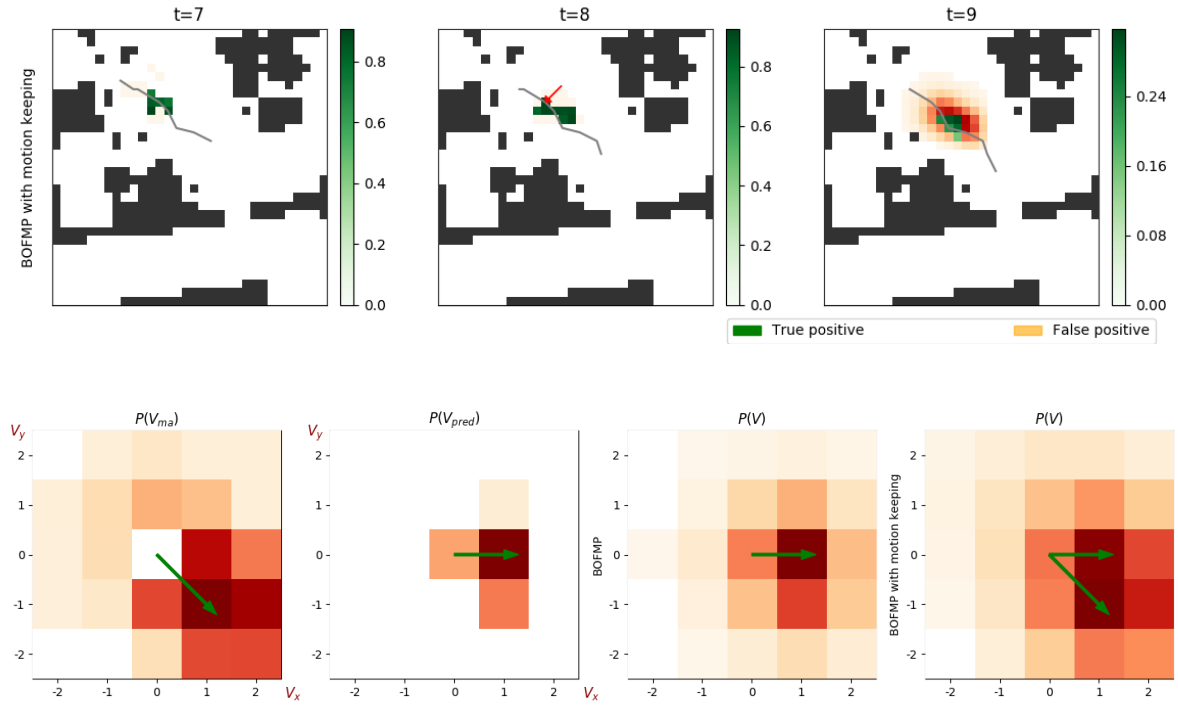

Figure 5.6: ***Up****: Three tracking steps of BOFMP with motion keeping. A person is walking from upper left towards lower right. The measurement is lost at $t = 9$.* ***Down****: Velocities of the cell pointed by the red arrow at $t = 8$. On each plot, the most possible direction is shown by green arrows. Based on measurements from $t = 7$ and $t = 8$, the predicted velocity $P(V_{pred})$ shows it is more likely the occupancy will propagates towards right for the next time step. However, the motion trend, which is represented by $P(V_{ma})$, shows it is more likely to move to lower right. As a consequence, BOFMP with motion keeping shows there are possibilities going both right and lower right, which is more realistic in this tracking example.*

We randomly sample 100 sets of parameters for each scenario, evaluate them on training set, and select the best set of parameters based on the mean of average precisions for the future prediction stage ($t = 9 : 16$). The best parameters are shown in Table 5.8.

	e	δ^2	Ω	$blurExt$	$blurVar$	w	$initMF$	$keepMF$	mean of a.p.
BOFUM	5	0.677	0.152	-	-	-	-	-	0.302
BOFMP	5	0.649	0.191	-	-	-	-	-	0.321
BOFMP spatial blurring	5	0.636	0.100	5	1.093	-	-	-	0.327
BOFMP motion keeping	7	0.744	0.026	-	-	4	0.563	0.707	0.381

Table 5.8: *Best parameters for BOFUM and BOFMP on real data.*

Then we apply each filter with its best parameters on test data of 244 scenes. Figure 5.7

shows the average precision for each time step, and Table 5.9 lists the average precision in future prediction stage and their mean. Compared with BOFUM, our methods have performance gain of 29%, 31% and 43% respectively.

	average precision for $t = 9 : 16$	mean
BOFUM	0.670 0.512 0.384 0.314 0.252 0.205 0.167 0.148	0.331
BOFMP	0.705 0.570 0.468 0.424 0.351 0.336 0.305 0.255	0.427
BOFMP spatial blurring	0.694 0.564 0.480 0.439 0.371 0.349 0.311 0.264	0.434
BOFMP motion keeping	0.762 0.675 0.561 0.496 0.405 0.353 0.305 0.238	0.474

Table 5.9: *Future predictions for BOFUM and BOFMP on real data.*

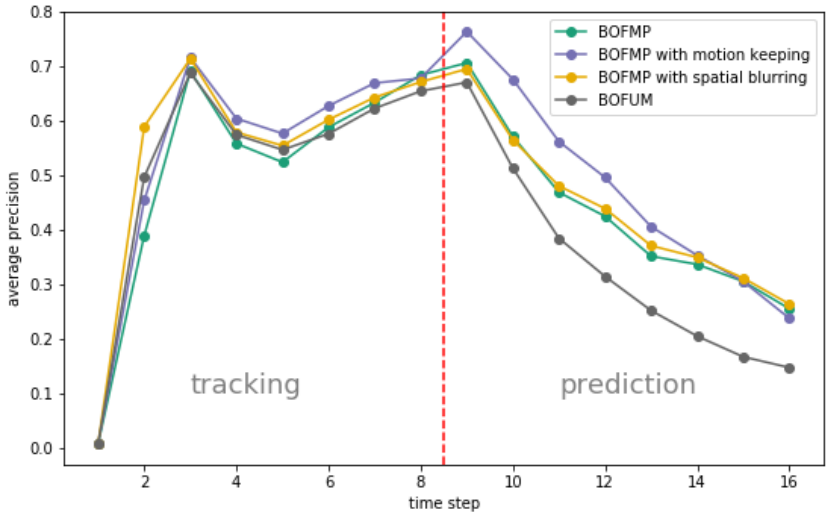

Figure 5.7: *Evaluation results on real test data. Like for simulated data, average precision starts with values close to zero, and increase rapidly over the next two time steps. From $t = 3$ to $t = 8$, average precisions keep rather stable at high values, which proves that filters predict very well for the next immediate step. In future prediction stage ($t = 9 : 16$), average precisions decrease severely as time horizon increases, since the state of the world becomes more uncertain. However, our BOFMP and its variations are still better than BOFUM for every time step in this stage.*

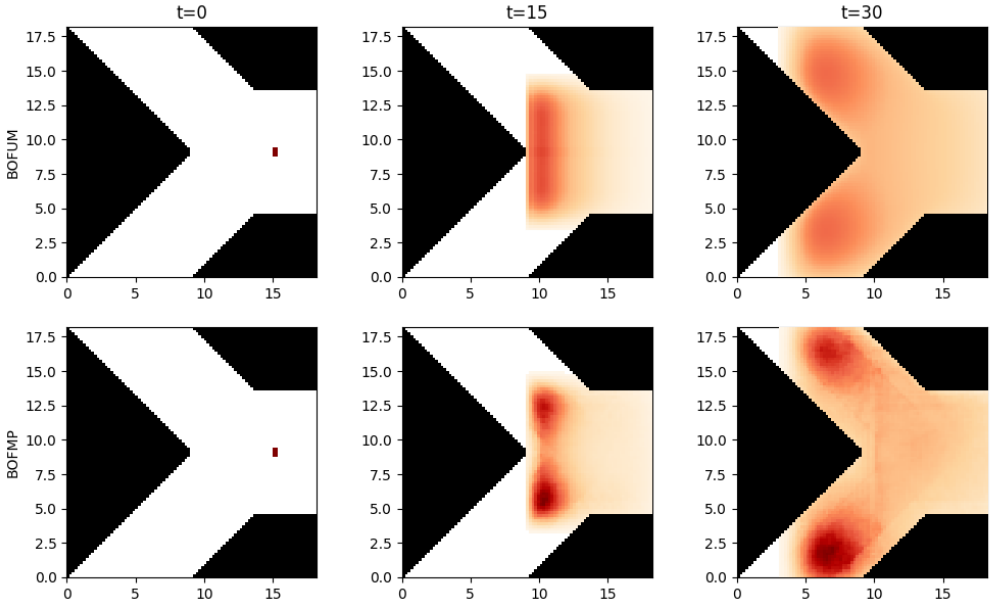

Figure 5.8: *Occupancy predictions for BOFUM and our proposed BOFMP after several time steps. The map shows a T-section. At $t = 0$, a person is shown as a red rectangle and with initial velocity towards left. At $t = 15$, the person encounters intersection. BOFUM has no information about human motion pattern, and continues to propagate occupancy towards left. Our BOFMP knows that humans are likely to turn to either upper or lower corridors. At $t = 30$, since occupancies going left vanish due to the wall, BOFUM predicts occupancies in corridors, but they are biased towards walls on the left. Our BOFMP predicts more occupancies in the middle of corridors, since it knows humans are more likely to walk in the middle.*

5.4 Applications

5.4.1 Future Occupancy Prediction

5.4.2 Dynamic Analysis

5.4.3 Get occupancy map

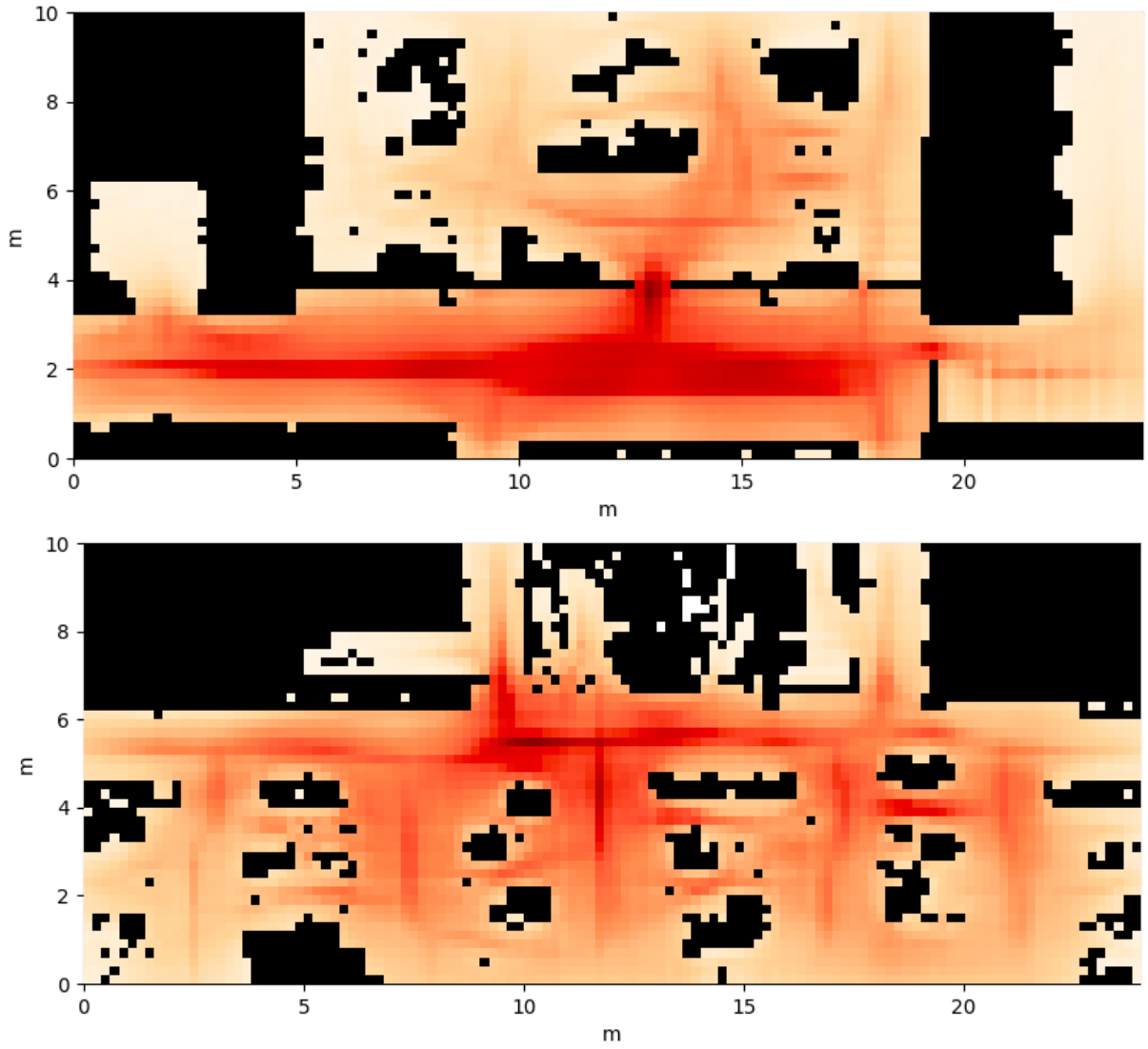

Figure 5.9: *Average occupancy prediction on static map.*

Chapter 6

Conclusions and Outlooks

1. train on simulation, but works on real data.

contribution: implement BOFUM with python

6.1 End to End Training

6.2 Future Work

Bibliography

[1] Daniel Arbuckle, Andrew Howard, and Maja Mataric. Temporal occupancy grids: a method for classifying the spatio-temporal properties of the environment. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 1, pages 409–414. IEEE, 2002.

[2] Kai O Arras, Oscar Martinez Mozos, and Wolfram Burgard. 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, 2007.

[3] Kai O Arras, Slawomir Grzonka, Matthias Luber, and Wolfram Burgard. Efficient people tracking in laser range data using a multi-hypothesis leg-tracker with adaptive occlusion probabilities. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 1710–1715. IEEE, 2008.

[4] Peter Biber and Tom Duckett. Experimental analysis of sample-based maps for long-term slam. *The International Journal of Robotics Research*, 28(1):20–33, 2009.

[5] Sebastian Brechtel, Tobias Gindele, and Rüdiger Dillmann. Recursive importance sampling for efficient grid-based occupancy filtering in dynamic environments. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 3932–3938. IEEE, 2010.

[6] Allison Bruce and Geoffrey Gordon. Better motion prediction for people-tracking. In *Proc. of the Int. Conf. on Robotics & Automation (ICRA), Barcelona, Spain*, 2004.

[7] Cheng Chen, Christopher Tay, Christian Laugier, and Kamel Mekhnacha. Dynamic environment modeling with gridmap: a multiple-object tracking application. In *Control, Automation, Robotics and Vision, 2006. ICARCV'06. 9th International Conference on*, pages 1–6. IEEE, 2006.

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

[9] Christophe Coué, Cédric Pradalier, Christian Laugier, Thierry Fraichard, and Pierre

Bessière. Bayesian occupancy filtering for multitarget tracking: an automotive application. *The International Journal of Robotics Research*, 25(1):19–30, 2006.

[10] Jinshi Cui, Hongbin Zha, Huijing Zhao, and Ryosuke Shibasaki. Laser-based interacting people tracking using multi-level observations. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 1799–1804. IEEE, 2006.

[11] Jinwei Gu Xiaodong Yang Shalini De and Mello Jan Kautz. Dynamic facial analysis: From bayesian filtering to recurrent neural network. 2017.

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

[13] Ajo Fod, Andrew Howard, and MAJ Mataric. A laser-based people tracker. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 3, pages 3024–3029. IEEE, 2002.

[14] Hervé Gauvrit, Jean-Pierre Le Cadre, and Claude Jauffret. A formulation of multitarget tracking as an incomplete data problem. *IEEE Transactions on Aerospace and Electronic Systems*, 33(4):1242–1257, 1997.

[15] Tobias Gindele, Sebastian Brechtel, Joachim Schroder, and Rudiger Dillmann. Bayesian occupancy grid filter for dynamic environments using prior map knowledge. In *Intelligent Vehicles Symposium, 2009 IEEE*, pages 669–676. IEEE, 2009.

[16] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

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

[20] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

[21] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[22] Andrew H Jazwinski. *Stochastic processes and filtering theory*. Courier Corporation, 2007.

[23] Simon Jégou, Michal Drozdzal, David Vazquez, Adriana Romero, and Yoshua Bengio. The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*, pages 1175–1183. IEEE, 2017.

[24] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *Computer Science*, 2014.

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

[26] Tomasz Kucner, Jari Saarinen, Martin Magnusson, and Achim J Lilienthal. Conditional transition maps: Learning motion patterns in dynamic environments. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1196–1201. IEEE, 2013.

[27] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, page 0278364917710318, 2016.

[28] Lin Liao, Dieter Fox, Jeffrey Hightower, Henry Kautz, and Dirk Schulz. Voronoi tracking: Location estimation using sparse and noisy sensor data. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 1, pages 723–728. IEEE, 2003.

[29] Angel Llamazares, Vladimir Ivan, Eduardo Molinos, Manuel Ocana, and Sethu Vijayakumar. Dynamic obstacle avoidance using bayesian occupancy filter and approximate inference. *Sensors*, 13(3):2929–2944, 2013.

[30] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[31] Matthias Luber, Gian Diego Tipaldi, and Kai O Arras. Place-dependent people tracking. *The International Journal of Robotics Research*, 30(3):280–293, 2011.

[32] Esther B Meier and Frank Ade. Using the condensation algorithm to implement tracking for mobile robots. In *Advanced Mobile Robots, 1999.(Eurobot'99) 1999 Third European Workshop on*, pages 73–80. IEEE, 1999.

[33] Daniel Meyer-Delius, Maximilian Beinhofer, and Wolfram Burgard. Occupancy grid models for robot mapping in changing environments. In *AAAI*, 2012.

[34] Michael Montemerlo, Sebastian Thrun, and William Whittaker. Conditional particle filters for simultaneous mobile robot localization and people-tracking. In *Robotics*

and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on, volume 1, pages 695–701. IEEE, 2002.

[35] Peter Ondruska and Ingmar Posner. Deep tracking: Seeing beyond seeing using recurrent neural networks. *arXiv preprint arXiv:1602.00991*, 2016.

[36] Peter Ondruska, Julie Dequaire, Dominic Zeng Wang, and Ingmar Posner. End-to-end tracking and semantic segmentation using recurrent neural networks. *arXiv preprint arXiv:1604.05091*, 2016.

[37] David A Ross, Jongwoo Lim, Ruei-Sung Lin, and Ming-Hsuan Yang. Incremental learning for robust visual tracking. *International Journal of Computer Vision*, 77(1): 125–141, 2008.

[38] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[39] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[40] Dirk Schulz, Wolfram Burgard, Dieter Fox, and Armin B Cremers. Tracking multiple moving targets with a mobile robot using particle filters and statistical data association. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 2, pages 1665–1670. IEEE, 2001.

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

[42] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[43] MK Tay, Kamel Mekhnacha, Manuel Yguel, Christophe Coue, Cédric Pradalier, Christian Laugier, Th Fraichard, and Pierre Bessiere. The bayesian occupation filter. In *Probabilistic Reasoning and Decision Making in Sensory-Motor Systems*, pages 77–98. Springer, 2008.

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

[45] Zhan Wang, Rares Ambrus, Patric Jensfelt, and John Folkesson. Modeling motion patterns of dynamic objects by iohmm. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 1832–1838. IEEE, 2014.