
Institute of Cognitive Science

*Master's thesis*

# Trajectory Prediction of Individual Pedestrians in Open Spaces

Annalena Bebenroth

October 2024

First supervisor: Prof. Dr. Joachim Hertzberg
Second supervisor: Georg John

# Abstract

Pedestrian trajectory prediction is a key challenge in various scenarios such as traffic or crowd management. This thesis aims to compare deep learning and classical methods to predict pedestrian trajectories in open spaces. In order to capture the behaviour of pedestrians in non crowded open spaces a state-of-the-art method named Human Scene Transformer is implemented and adapted towards individual pedestrians as well as compared to the non-deep learning methods Kalman Filter, Interacting Multiple Models and Particle Filter. The methods are evaluated on different real world and a custom synthetic dataset based on the average displacement error, final displacement error and the runtime. Results show that the Kalman Filter with a constant velocity model achieves the best accuracies and fastest runtime for the open space scenario across all datasets. Although the adapted version of the Human Scene Transformer achieves comparable accuracies for short-time prediction, the Kalman Filter outperforms on longer trajectories and is the best solution in terms of real-time application.

# Contents

# Chapter 1

# Introduction

Human behaviour is complex and driven individually by many different factors. Humans mostly have a goal they are heading towards or an internal motivation to move and they are following social norms and rules, e.g. following given paths like sideways or stopping at a red traffic light [1]. A simple model cannot accurately describe many real-world situations. When thinking of human interactions there is a wide range of possible behaviour, from groups walking together in the same direction to humans avoiding each other when their paths may cross or when external factors like loud sirens in a street are influencing the behaviour. Understanding and modelling systems of human behavior has always been a particular interest of many researchers due to their direct application in situations like crowd management. Until now, the complexity of such systems has inhibited progress in this field. That is the reason why there are different objectives that set a focus and limitation. From action or intention detection of humans [2] to trajectory prediction [3], each task includes still diverse, complex human behaviour and there are many different methods trying to capture the task-specific behaviour [4][5].
Aside from the objective, the use case has a large influence on the relevant behaviour and what needs to be considered in order to model the human behaviour. One common use case in research is autonomous driving or traffic scenarios [6][7] where models try to capture and predict the pedestrians behaviour especially at crossings or traffic lights. This is done in order to improve the security of pedestrians by predicting human behaviour and to be able to stop autonomous systems with the goal of collision avoidance. Another huge research area is interaction modelling [8][1][9], which focuses on the interaction between a number of pedestrians. This can be captured from surveillance cameras in malls or from movable robots in indoor or urban environments.
In order to model human behaviour one therefore needs an objective to focus on and a given environment and setting, a so called use case. This thesis therefore focuses on the task of pedestrian trajectory prediction in open space scenarios.

## 1.1 Motivation

The overall goal of this thesis is to predict pedestrian trajectories. As described before, this is a large and complex task due to the complexity in human behaviour and therefore some assumptions were made in order to specify the use case as described in section 1.2. This goal

was not only set due to interest in this research area but also because of its intended application in a real world scenario: A mobile robot could incorporate the pedestrian prediction into its path planning algorithm in order to react to occurring pedestrians in its proximity and avoid collisions with them. This could enable the robot to work in public spaces without the need for a safety barrier, making it much more tolerable and practical for the general public.
Now with a task, use case and goal in mind, one has to decide how to model the given behaviour. Choosing the right model is difficult and for traffic and interaction scenarios, state-of-the-art neural networks like Generative Adversarial Networks or Transformers are shown to achieve the best predictions so far [5]. Though deep learning can handle such prediction problems by learning the underlying probability distributions, non-deep learning methods that are based on simple rule sets or other algorithms can be applied too and are common in pedestrian trajectory research[5]. That wide range of models does not make it easier to choose one and therefore the idea of this thesis is to not decide on one model for the given problem but to compare different approaches and especially compare deep learning with non-deep learning models. The intention for choosing this comparison is that recent trends seem to always consider deep learning, and this thesis evaluates if simpler methods are sufficient for tasks like the relatively simple one, described in the following section. The implementation can be found in [10].

## 1.2 Problem Statement

The intended use case for pedestrian trajectory prediction is based on the ChargePal project [11] by Deutsches Forschungszentrum für Künstliche Intelligenz GmbH among other partners. The project aims to develop a mobile robot that can charge electric cars automatically with a mounted robotic arm and is designed to recharge multiple cars by moving from vehicle to vehicle. Pedestrian Trajectory Prediction should help the robot during path planning in order to be able to dynamically adapt the path if pedestrians are probably crossing the robot's intended future path. From the project's intention, the use case can be inferred as an area with parking lots and some parking cars. Since the main operators and customers of the project are likely to be larger operators who want to charge a large number of cars in a commercial context, it can be assumed that the parking lots are rather deserted with only few pedestrians possibly crossing the robot's path.
From that the scenario for this thesis can be described as follows: A parking area with parking lots, parking cars, a moving robot and few pedestrians walking around. The data is collected from the mobile robot, since this is the agent in the use case, has the sensors to detect the pedestrians and needs the information for its own path planning. A more general use case can therefore be described as an area with no obstacles except single moving pedestrians and the data is collected by a mobile robot. This use case is in the following thesis called open space scenario and data fits into it if it fulfills the following criteria:

- egocentric view - data is collected from the view of a mobile robot
- detected pedestrians - in general it can be any 2D or 3D relative to the robot detection or at least a sensor that is usable for detection algorithms

- single trajectories - there will probably be no complex interactions between many humans, but single ones following one intention like searching an intended car to drive it
- no crowds - complex behaviour is rather rare and not the average behaviour

## 1.3 Objectives of the Thesis

Given the open space scenario as a use case with assumed simplistic human behaviour and the fact that related research largely uses deep learning approaches, the research question arises as follows:

> **Is a state-of-the-art neural network for such a simplification still a large benefit over classical non-deep-learning methods?**

Since choosing the best model for a problem is difficult, the thesis will handle one state-of-the-art deep learning method that is shown to achieve good prediction accuracies on a similar use case and multiple simple non-deep learning methods to compare their accuracies and efficiency. From an intuitive view, one would assume that a deep learning approach can handle human behaviour better because it can in general capture complex data distributions and can be designed to capture rare cases like missing or sparse data. Moreover, both methods use prior knowledge that is especially influencing the predictions when few history is available. The difference is that deep learning approaches train on many samples and build up a non-linear, complex prior knowledge while the Bayesian Filters are receiving their prior knowledge from the assumed motion models that are in this case simple (like the assumption of constant velocity). The hypothesis to be validated in this thesis is therefore:

> **The deep learning approach is better in short time prediction and models human behaviour more realistic.**

## 1.4 Structure of the Thesis

In chapter 2, a literature review presents an overview of the fundamentals that are used in this thesis as well as state-of-the-art methods and datasets. Then the methodology (chapter 3) describes what is done in this thesis in order to answer the given research question and how the comparison between the approaches is achieved by pre-processing the datasets and implementing the different approaches. The experimental setup including the used hardware and used metrics is described in chapter 4. The analysis of all methods (chapter 5), individually and compared to each other, is done in terms of prediction accuracy and computational efficiency. The results of this analysis are then discussed (chapter 6) with the research question and hypothesis in mind. Especially the implications for the use case, advantages and limitations as well as possible insights for future research are examined. Finally, the work of this thesis is summarized and the main conclusion is given (chapter 7).

# Chapter 2

# Literature Review

The following section will discuss and explain the most relevant concepts for this thesis. First of all, the fundamental methods used in the later implementations are discussed, namely the transformer algorithm as well as the Kalman Filter, the Particle Filter and the Interacting Multiple Models approach. Afterwards, the state-of-the-art deep learning model, the Human Scene Transformer, is described as well as the use case relevant datasets that are later used for analysing the different models.

## 2.1 Fundamentals

For giving a short overview over the most relevant fundamental methods for the further understanding of this thesis, the general transformer architecture and its advantages are discussed because of its close relation to the the Human Scene Transformer. Further, the basic concepts of the non-deep learning methods that are chosen as simple algorithms to contrast the Human Scene Transformer are explained.

### 2.1.1 Transformer

The Transformer [12] is a state-of-the-art deep learning method for processing sequential or ordered data. Its advantages over methods like recurrent neural networks or convolutional networks are the use of attention mechanisms and processing the whole sequence as once. The main usage of transformers is for sequence-to-sequence tasks, where recurrent networks are mainly processing single tokens (words or other data) from the sequence to output a sequence but therefore struggle to capture long-time dependencies (see Figure 2.1). By passing through all tokens of a sequence step-by-step, tokens that are processed at the beginning are getting lost along the way of processing the whole sequence and if there is for example a strong dependency between the first and the hundredth word, an RNN may not capture that. Though attention mechanisms were invented and used before, the transformer uses an adapted version, namely the multi-head attention, with a scaled dot-product attention, that is mainly responsible for its fast computations, in a simple encoder-decoder structure.
The overall encoder-decoder structure (see Figure 2.2) means that the input sequence is encoded to a hidden vector (output of the left side encoder in Figure 2.2) which is then decoded to a

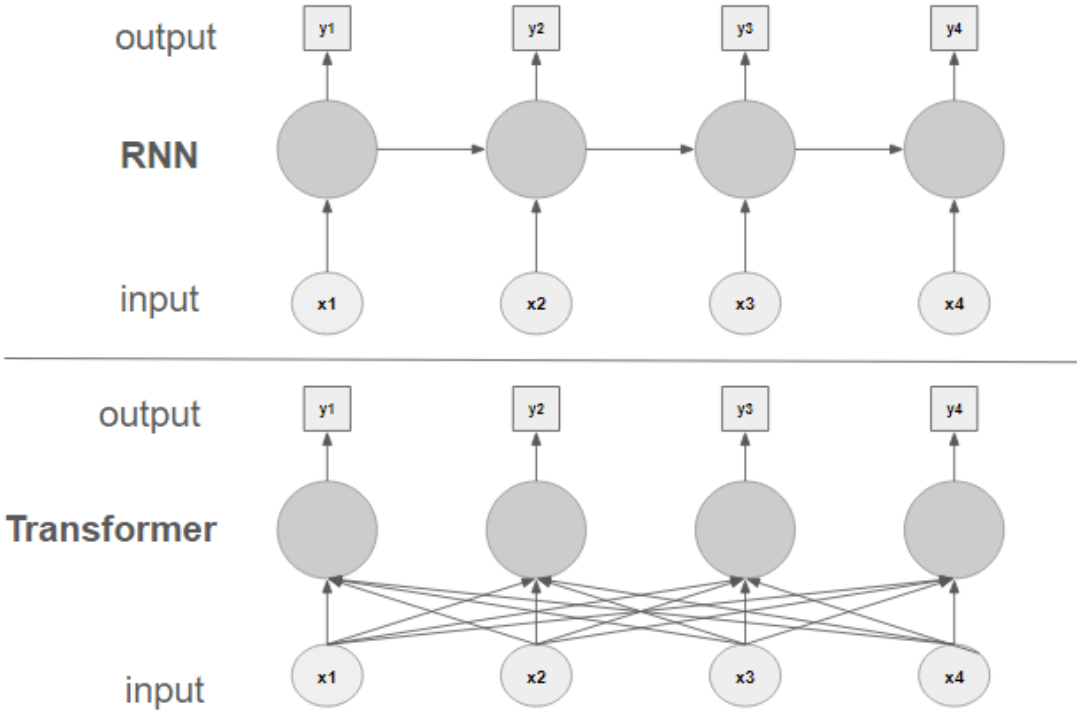


**Figure 2.1:** A comparison between an RNN (top) and Transformer (bottom) and how they process a sequence of tokens x1-x4 to output a sequence y1-y4. The RNN processes the input x to a hidden state (middle, grey dot) that uses previous hidden states and the new input to produce the output y. The transformer uses self-attention over all sequence tokens (x1-x4) to produce a single output of the sequence.

sequence again. Additionally, the transformer embeds the input once by its meaning and once by its positions. Embedding an input is a normal procedure in deep learning, since it allows representing the meaning of the input in a manner that is easier to process by the neural network. Embedding the position is new and necessary because with the simultaneous processing of the whole sequence in the attention mechanisms, information about the position is getting lost. Positional encoding works by using sine and cosine functions to assign a unique set of values to each position in the input sequence. These functions represent different frequencies, which help the model understand where each word or token is located [12]. The input to the transformer is then the learned embedding of the input plus the positional embedding.
Despite the positional embedding, attention mechanisms are also an important point of transformer networks. The attention mechanism [12] in general enables the network to focus on all parts of the sequence simultaneously. It takes three inputs with the provided information as values $V$ and keys $K$. The keys are used to identify the tokens in the value sequence. The third input is the query $Q$ that represents the token of interest where the model tries to find the tokens that are the most relevant and thus influential. The mechanism itself is based on a scaled dot-product and calculates similarities between the query and the keys to find the most relevant

indices. The higher the similarity, the higher the weight (attention score) for that token, which is then used for a weighted sum over the values for an output sequence.

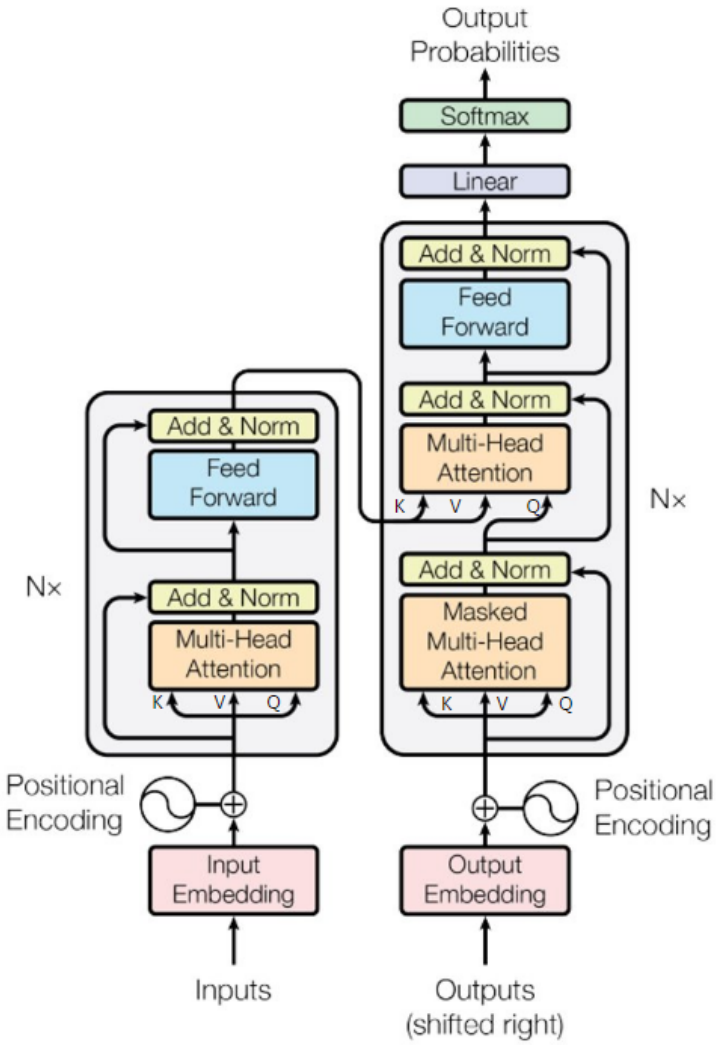


**Figure 2.2:** The transformer model architecture from [12]. The architecture shows the processing pipeline from the input including the positional and informational embedding as well as the overall encoder (left side) and decoder (right side) structure. The outputs are evolving with every new predicted output. The Multi-Head Attention Mechanisms weighs the input by its importance for the query and the three input arrows indicate the queries, keys and values.

Now, multi-head attention is used by dividing the input sequence into smaller parts and calculating attention for each part in parallel. Instead of processing the entire sequence at once, attention is applied to the smaller parts simultaneously. This does not limit the models ability to capture dependencies across the entire sequence because by using linear projections to split

the sequence, the model is still able to learn relationships between all tokens. It is an advantage in terms of computational efficiency and also allows the model to focus on different parts of the sequence simultaneously, otherwise the attention may average over all aspects of the sequence. All in all, the transformer is a state-of-the-art model in sequence-to-sequence tasks by using multi-head attention in an encoder-decoder style. The transformer uses positional encoding and the whole sequence as input and therefore has to learn the sequential nature of the data. Due to its parallel computations it is moreover fast and computationally efficient.

### 2.1.2 Kalman Filter (KF)

Though more known for tracking problems, Bayesian filters like the Kalman filter are a common tool for trajectory prediction too. Adapted versions of the Kalman filter are commonly used in trajectory prediction. First of all, the Bayes Filter is a general concept that helps estimating a state of a system over time. While the state can be any variable that describes a system, it should correspond to a possible measurement, e.g. coordinates or velocities [13]. It is called filter, because the noise contained in the measurements is filtered out in order to get closer to the underlying real state.
The filter process can be described as building a belief from measurements and an assumed model that describes the process over time. Therefore the filter should remove the noise as best as possible by weighing a known model and the given data to get as close to the real state as possible. The main process stays the same for all Bayes filters like the Kalman filter:

**Algorithm 1:** Generic Kalman Filter Algorithm [14]

**Initialization**
1. Initialize the state of the filter.
2. Initialize the system's belief in the state.

**Predict**
1. Use system behavior to predict the state at the next time step.
2. Adjust belief to account for the prediction uncertainty.

**Update**
1. Get a measurement and associated belief about its accuracy.
2. Compute residual between estimated state and measurement.
3. Compute scaling factor based on whether the measurement or prediction is more accurate.
4. Set state between the prediction and measurement based on scaling factor.
5. Update belief in the state based on how certain we are in the measurement.

The Bayesian Filter is a probabilistic approach that relies on the same-named Bayes' theorem in order to estimate a posterior distribution or belief:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{2.1}$$

The theorem 2.1 can be interpreted for the Bayes Filter as using the previously predicted belief about the system's state ($P(A)$), the likelihood of the new measurement given the state ($P(B|A)$) and a normalization constant ($P(B)$) to estimate the new belief about the system's state. Moreover, the likelihood of the measurement is determined by the given measurement model, similar to the prior belief that is determined by the system model and estimated in the previous prediction step.
The Kalman filter is one form of an implemented Bayes filter and is designed to filter and predict motion in linear systems. That means that the system is assumed to be linear, like the noise in the process and the measurement is assumed to be Gaussian. With that, the models needed to estimate each new state and measurement need to be linear too. The outputs of these models are therefore Gaussian distributions described by their mean $\theta$ and variance $\sigma$. A discrete-time Kalman filter model that describes state transitions looks like:

$$x(t) = Ax(t-1) + Bu(t-1) + w(t-1) \tag{2.2}$$

Equation 2.2 [15] is a simple linear transition model with $x(t)$ as the new state or belief, given a transition matrix $A$ that models the state transition from the previous state $x(t-1)$ to the current one. Additionally, the given measurement $u(t-1)$ transitioned by the matrix $B$ and noise $w(t-1)$ is simply added. Models describing different behaviours are defined then by different transition matrices for the system states $A$ and the covariances of the noise processes.
One prominent dynamical model is the constant velocity (white noise acceleration) model (CV) [16] that works on systems whose states can be represented by their position and respective velocities. The CV model assumes movement with no acceleration and therefore constant velocity, like the name suggests. To account for small, unpredictable factors and uncertainty, the model includes small white noise as acceleration (see Figure 2.3).

Similar to the CV model, there exist a constant acceleration model (CA) [16] that assumes

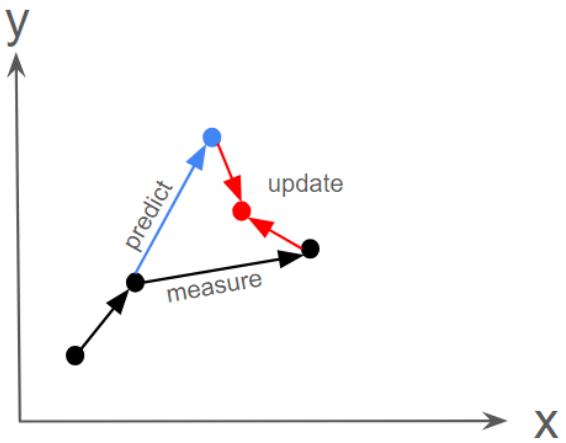


**Figure 2.3:** An example for the Kalman Filter process that operates in 2D space, takes the black dots as position measurements, predicts the new belief from a constant velocity model (blue point), gets a new measurement and updates the internal belief (red) according to that new measurement.

the acceleration of the system to be constant, e.g. like a car speeding up. Moreover the system state is described by position, velocity and acceleration. The derivative of the acceleration is assumed to be independent white noise, to account for unmodeled dynamics like unexpected little accelerations in the CV model.
A more complex model is the constant turn model (CT) [16] that models motions that have constant velocity and constant turn rate, like objects that follow a curved path. This motion description needs functions like sine and cosine as well as a variable for the turn rate and is therefore non-linear. To nevertheless use a normal, linear Kalman Filter, the turn rate needs to be known or estimated beforehand and stay fixed during the application of the model. This can be achieved by approximating the turn rate from the first steps or by combining multiple models with different turn rates. This is discussed in the next section 2.1.3. One can also use an Extended Kalman filter [17] [13] (EKF) which is a KF adapted to work with non-linearity.
The EKF is designed to handle non-linear systems by linearizing non-linear functions. Instead of simple transition matrices, the models use the non-linear function **g** to describe the system's dynamics and measurement processes:

$$x(t) = g(u(t), x(t-1)) + w(t) \tag{2.3}$$

Now, the belief in the system's state should still be described by a Gaussian with a mean and variance, which is not possible if put through a non-linear function [14]. Therefore, a linearization happens by calculating the Jacobian for each current state estimation. The Jacobian matrices are matrices of partial derivatives, which makes it possible to apply the linear KFs prediction and update step. Therefore, the EKF allows for the use of non-linear models while keeping the computational efficiency and simplicity of the simple KF.
This is the most simplified form to model a coordinated turn motion and there are different variants around, like the augmented coordinated turn model (ACT) with cartesian or polar coordinates as well as the CT model with kinematic constraints [17]. Overall, the accuracy of tracking or prediction is highly problem dependent and therefore model and state dependent.
While these approaches are mainly used for tracking, the Kalman filter can be simply adapted for use in prediction, especially of pedestrian trajectories. This can be achieved by only including new measurements for history time steps and then estimating the new state without the measurement part [15].

### 2.1.3 Interacting Multiple Models (IMM)

Though the Kalman filter handles single dynamical models, many systems can have changing behaviour and thus cannot be handled by a single motion model. The Interacting Multiple Models approach (IMM) [15] is designed to overcome this problem by enabling the model to change motion models between steps. One could therefore theoretically handle straight-line movements and turning movements in one method by combining the constant velocity model with the constant turn model and therefore capturing both behaviours, not simultaneously but by switching behaviour back and forth. The IMM can handle different amounts of motion models and handles switches between them by a transition probability matrix that defines each entry as the probability that the system transitions from one motion model to another [18]. This matrix

is designed beforehand by prior knowledge and may express that systems are likely to stay in their behaviour or change often. If there is knowledge that behaviour changes when a certain event happens or after a given time, this could be modeled in the transition probability matrix. The overall idea is that all models share the same state space as well as measurement model, but differ by the motion model and therefore predict a different state at each time step. The IMM then creates a combined state estimation, or belief, by combining the different predicted states with the respective model probability and the knowledge of the transition probability matrix (see Figure 2.4). After creating a belief, the single models are then updated by their respective algorithm, e.g. the simple Kalman Filter.

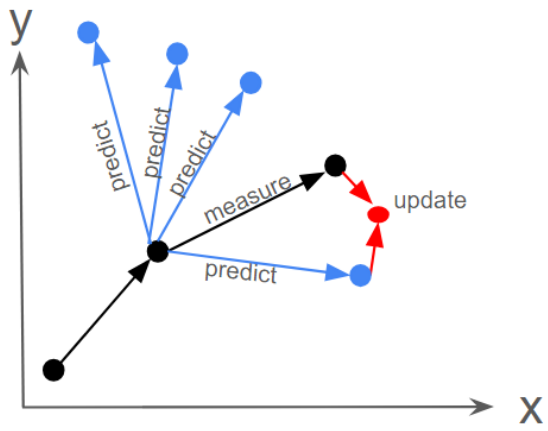


**Figure 2.4:** An example for the IMM process that operates in 2D space, takes the black dots as position measurements, predicts new beliefs from one constant velocity model and three constant turn models (blue points), gets a new measurement and updates the internal belief (red) according to that new measurement and the model probabilities for the different models.

Moreover the model likelihoods, so the weights for the models, for the next prediction step are then updated by the likelihood of the new state given the different models. So the IMM approach looks how well the single models predict the next state and those closer to the actual measurement get a higher weight to be the favoured model. The algorithm can be simplified to:

**Algorithm 2:** Interacting Multiple Models Algorithm [18]

**Initialization**

1. Initialize the system state.
2. Initialize a set of models (e.g. KF) with corresponding state transition models and measurement models.
3. Initialize the model probabilities and transition probability matrix.

**Predict**

1. Use system behavior to predict state at the next time step for each model independently.
2. Create belief by combining each predicted state with its model probability and the transition probability matrix.

**Update**

1. Get a measurement and associated belief about its accuracy.
2. Compute residual between estimated state and measurement.
3. Update model probabilities based on residual and likelihood of each model producing the measurement given the corresponding prediction.

To sum up, the IMM approach allows transitions between different motion models and continuously updates the likelihood of these models to best capture the current behaviour.

### 2.1.4 Particle Filter (PF)

The Particle filter (PF) [14] is similar to the Kalman filter in that it tackles the problem of estimating states of a system over time. Moreover, the PF is a general concept and includes several different approaches. The main intention is to tackle non-linear systems that are described by multiple independent random variables. The advantages over the Kalman filter are that it can handle occlusions in the measurements, non-Gaussian noise and unknown process models. The main idea behind the PF is that the probability distribution of the state is represented by a set of sampled particles instead of one mean and covariance (like in the KF). These particles each describe one possible state and the filter updates the particles probability based on the given measurements (see Figure 3.3). Through this set of particles, the process is more robust to complex and unpredictable noise since it is more likely to cover such a case by one of the particles than having only one state.

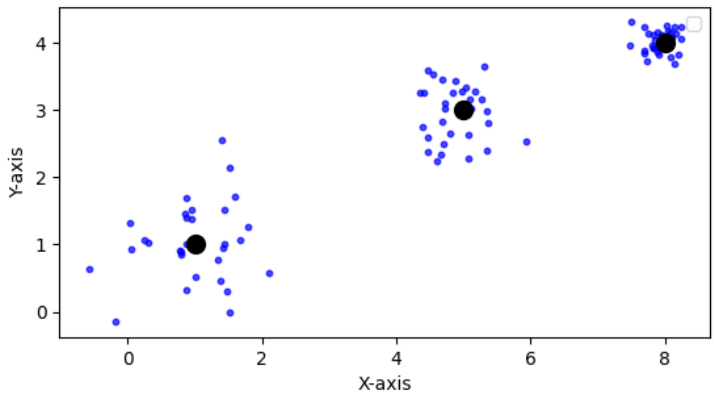


**Figure 2.5:** An example for the PF process that operates in 2D space, takes the black dots as position measurements, initializes the particles (blue dots) around the first measurement and updates the particles based on the incoming measurement of a new position. Therefore, the particles get closer to the measurement.

The overall Particle Filter algorithm can be described as:

**Algorithm 3:** Generic Particle Filter Algorithm [14]

**Initialization**

1. Initialize a set of particles and corresponding weights (should sum up to 1 to produce probability distribution.)

**Predict**

1. Use system behavior to move the particles.

**Update**

1. Get a measurement.
2. Update the weighting of the particles based on proximity to actual state.

**Resample**

1. Update set of particles by replacing those far away from the real state (low weight/probability) with ones that are more likely (high weight/probability).

**State Estimate**

1. Get a state estimate for the current time step by taking the weighted mean and covariance of the set of particles.

In order to use a PF for trajectory prediction, one problem needs to be considered. Even though the PF is similarly to the KF used for state estimation, it is based on tracking measurements and updating the movements of the particles accordingly (like landmark observations in order to infer the the observers position). Therefore, no explicit process model is needed for tracking but if the PF should be used for prediction, one needs a model that estimates the system's behaviour in order to move the particles accordingly. That is the reason why the PF needs to be extended by a system behaviour model with the highest possible accuracy for tra-

jectory prediction. Conde et al. [19] use a model that describes the motion of unmanned aerial vehicles where some parameters are set based on the knowledge of the vehicle while others are estimated based on real data. Moreover, they showed that the Particle Filter with a good model can estimate uncertainty sufficiently.
Other approaches are using combinations of the Particle Filter with other approaches, like the IMM in order to primarily handle non-linearity and non-Gaussian states [20]. For tracking maneuvering targets, Foo and Ng [20] show for example that combining an IMM with a CV, CA and CT model and using a Particle Filter instead of the Kalman filter reaches sufficient accuracies.

## 2.2 State-of-the-Art Methods for Pedestrian Trajectory Prediction

The following section describes one state-of-the-art deep learning approach, the Human Scene Transformer, that predicts pedestrian trajectories. Further, other deep learning approaches that are based on differing model architectures are described.

### 2.2.1 Human Scene Transformer (HST)

The Human Scene Transformer (HST) [21] is designed to model interactions of humans in crowded human-centric environments sensed by a mobile robot. The goal is to predict trajectories while taking diverse human behaviour as well as ego-motion of the robot into account (see overall architecture in Figure 2.6). Salzmann et al. [21] introduce visual features like skeletal keypoints, combined with positional information, to improve prediction accuracy. They showed that compared to other approaches, the HST can predict accurate trajectories, in indoor and outdoor scenes especially with limited knowledge of an agent's history.
The idea of taking visual aspects into account is influenced by the fact that in the setting of a moving robot, one has to deal more with smaller spatial environments than for example in street scenes and therefore may gain more detailed visual features. Moreover, with persons moving closely around a robot and possibly more obstacles such as indoors, bounding boxes in an image space may not capture sufficient information but features like detecting skeletal keypoints is possible. Another assumption about the described use case is that human behaviour is more diverse than in street scenes where more rules are given (explicit rules as well as implicit, i.e. socially given ones). This should be captured by predicting multiple possible trajectories as well as using visual features to indirectly model human intentions, like humans do when predicting other people's future movement.
The choice of a transformer-based architecture is mainly influenced by other research that showed the capability of transformers to handle varying amounts of agents simultaneously as well as being able to model interactions between a varying amount of agents in a scene. Additionally to previous work, the HST builds upon features in the world space compared to other approaches only predicting bounding boxes in image space, as well as explicitly using plausible visual features combined with positional information (see Figure 2.6).
Some implementational aspects are Masked Sequence-To-Sequence Prediction, Feature Combination via Attention, Agent Self-Alignment and Multi-Modality Induction. Masked sequence-

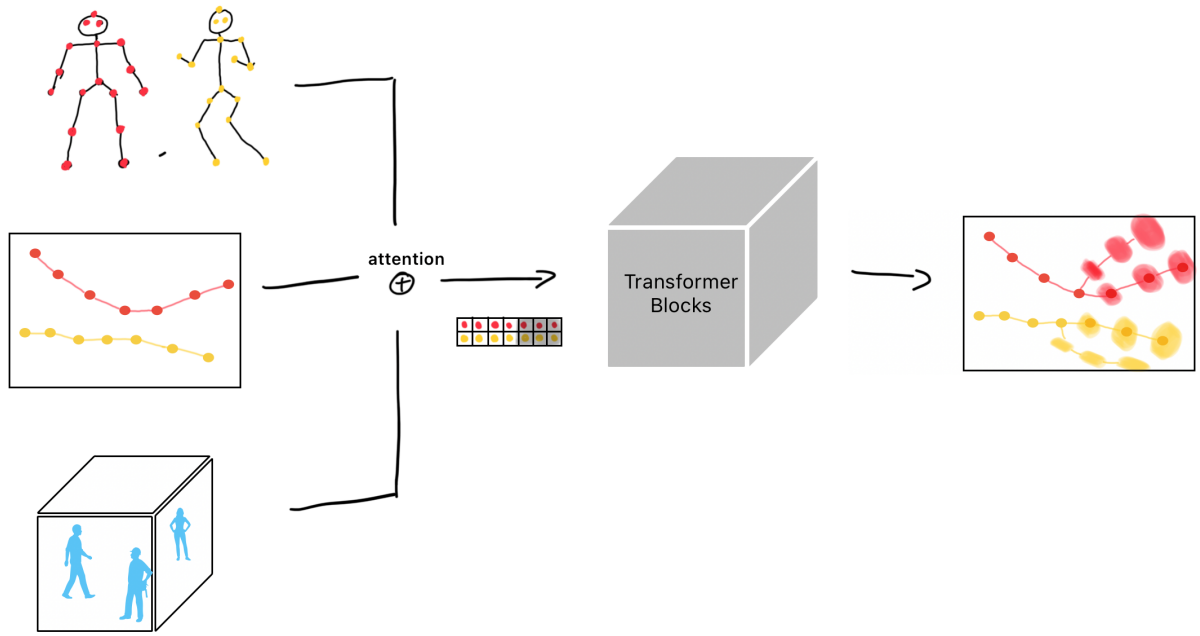


**Figure 2.6:** A simplified overview of the HST, which takes skeletal keypoints (left side, humans with red and yellow points at relevant joints), the position in 2D (left side, red and yellow trajectories) and the scene context (left side, 3D box with humans) as input. The features are combined via attention to agent-time step-tokens that combine all neighbouring agents (box with red and yellow dots for the neighbouring agents) and mask out the steps to be predicted (grey parts in the box). The tokens are processed in transformer blocks and output multiple possible outcomes by a probability distribution per step and agent (lighter coloured ovals in the right box).

to-sequence means that the whole sequence is inputted to the model, but the future steps that should be predicted are masked out by setting them to zero. Moreover, if some features like keypoints cannot be obtained or are NaN values, those can be masked out and the model learns to handle these cases and gets more robust to occlusions. That forces the model to learn from the given context and generalise as well as learn dependencies and relationships within the sequence. One important difference to other transformer architectures is that there is no explicit position encoding but the model has to understand the sequential nature from the masked sequences. Moreover, since the input of one sequence consists of different features, the combination of them should be optimized by using attention over the embeddings in order to let the model learn which features are important and let it efficiently manage this information.
Another aspect is that the model handles multiple agents at once by predicting the future of one agent while taking neighbouring agents into account. With that, the HST includes an agent self-alignment layer to show the model which agent is focused at that moment, otherwise the inputs would look the same for neighbouring agents. Taking neighbours into account should model interactions and is explicitly handling the complete sequences of the neighbours as well, because it is assumed that the history of others influences current behaviour too. In order to

model different behaviours, a multi-modality induction introduces different modes with probabilities that output different trajectories and shows which one is the most probable. The single predicted steps are modelled as Gaussian Mixture Models, meaning not as simple coordinate but as a probability distribution, to capture uncertainty better.
More details about the implementation are described in section 3.4.

### 2.2.2 Additional Deep Learning Approaches

There exists a range of different architectures and methodologies in order to predict pedestrian trajectories. One is the Social LSTM [9], introduced by Alahi et al. in 2016, with the idea of modelling interactions in a scene by giving each pedestrian an individual model that can take neighbouring pedestrians into account. Therefore a joint prediction over all pedestrians in a scene is made by sharing information and modelling the human sequential behaviour with a Long-Short-Term-Memory (LSTM).
This idea was adapted and further developed for higher prediction accuracies, for example by the Trajectron++ [22]. The improvement comes from a more complex modelling of the influence on the pedestrians to each other by a implementing a graph neural network. Similarly to the Social LSTM, each pedestrian is given its own LSTM to model its individual behaviour. The focus of the Trajectron++ is to model the overall scene as best as possible and to take other influences like cars or the general context better into account. Moreover, the Trajectron++ achieves state-of-the-art prediction accuracies on various datasets and similarly to the HST outputs multiple possible outputs with a Variational Autoencoder.
Despite other deep-learning methods like Generative Adversarial Networks [23] or Transformers [24], hybrid approaches that combine rule-based modelling with deep learning showed promise in the past years. One example that achieves state-of-the-art performance in various pedestrian trajectory datasets is the neural social physics approach (NSP) [8]. The NSP combines deep learning with non-deep learning approaches by limiting the learning of a neural network by assuming an explicit physics model that has specified parameters to learn. The physics model is inspired by the social force model [1] and therefore focuses on capturing crowded scenarios and interactions.
The decision for the Human Scene Transformer instead of one of the described ones above, is based on the use of skeletal keypoint information as well as the transformer architecture. Though GANs or Autoencoders are recently developed too, the transformer has its advantages in capturing long-term dependencies and highlights the intended comparison between a complex state-of-the-art method and simpler algorithms. To sum up, recently there has been no obvious method to choose for pedestrian trajectory prediction. Moreover, the state-of-the-art methods mostly focus on capturing complex interactions in order to model the human behaviour.

## 2.3 Relevant Datasets

The following datasets are chosen for the intended use case described in section 1.2. A dataset should therefore collect data from an egocentric view, offer pedestrian trajectories and the dataset should collect single trajectories without crowds. Since there is no dataset fitting all the described criteria, the single points are prioritised and a dataset should at least fulfill the

data collection from a mobile robot and offer pedestrian detections in 2D and 3D. In the best case, but not necessarily, the data captures scenes with a realistic number of obstacles and little context that may influence the pedestrians. Moreover, crowded scenarios are not permitted because the datasets can be filtered to include only a minimum of interactions.
Though there are datasets focusing on single trajectories, the intention behind the dataset collection is mostly too differing from the intended use case here. For example, the OxfordHIM dataset [6] is limited to only one room, has a single trajectory per scene and focuses on goal-oriented behaviour. It is excluded from this thesis because it is too scripted behaviour and does not model natural human behaviour in open spaces.
Moreover, one important criterion comes from the chosen state-of-the-art model additionally to those inferred from the use case. Since the Human Scene Transformer uses skeletal pose information as well as positional information, the datasets should include corresponding data. In the best case, detections of pedestrians in 3D space relative to the robot and detected skeletal keypoints are already given. Since skeletal keypoints are an uncommon feature and easily estimated by a pre-trained neural network for pose estimation, an image and 2D bounding box for the specific pedestrian would be sufficient in cases where skeletal keypoints are not given in a dataset.
The following datasets JRDB and Crowdbot are chosen based on the prioritised criteria and despite the possibly large amounts of pedestrians.

### 2.3.1 JackRabbot Dataset and Benchmark (JRDB)

The JackRabbot Dataset and Benchmark (JRDB) is a dataset and benchmark with the focus on collecting multi-modal data from an egocentric view of a movable robot. The data is collected at a campus with the JackRabbot mobile robot either stationary or moving in indoor and outdoor scenarios. The multi-modal data is collected by rotating laser scans and different RGB cameras. Moreover, the dataset offers a wide range of annotations including 2D bounding boxes for different cameras as well as 3D bounding boxes for LiDAR point clouds (see Figure 2.7).

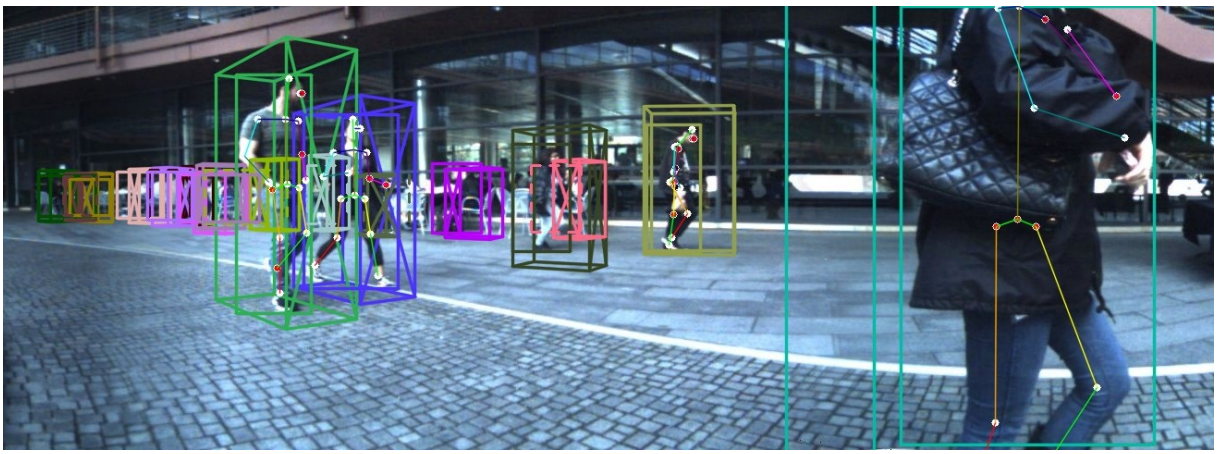

**Figure 2.7:** A sample frame from the JRDB dataset with visualised 2D and 3D bounding boxes for each pedestrian in the scene as well as the detected skeletal keypoints.

The whole dataset contains 64 minutes of collected data in 30 different locations resulting in 54 different recorded sequences. From the 54 sequences, 33 are indoors and 21 outdoors where 33 are recorded from a moving robot and 21 from a stationary one.
For the detection benchmark, JRDB offers additionally unique IDs for each pedestrian that are consistent across 2D and 3D annotations. While the 2D detection is given as a bounding box in the camera frame, the 3D detection is given as a bounding box in the local LiDAR frame. The 3D bounding box is defined by its cuboid center as x, y and z position as well as by its orientation as roll, pitch and yaw Euler angles [25]. The respective coordinate axes can be interpreted as that the x-axis is pointing forward, so where the robot is heading to, the y-axis is pointing to the left of the robot and the z-axis pointing upwards. This is the local frame of the specific LiDAR sensors, but for simplifying sensor fusion, the coordinates are translated to be centered on the camera. This makes later calculations between 2D and 3D less computationally expensive.
For the pose estimation benchmark, JRDB-Pose, a JRDB annotation extension, offers skeletal keypoint annotations in the COCO style. This means that the human pose is described by 17 skeletal keypoints where each has a specific meaning, like head, left shoulder or right foot. Moreover, each keypoint has a visibility score and each detected person a head bounding box. Martin-Martin et al. [26] demonstrate that the dataset captures a range of natural human postures and behaviors. Even with only 54 recorded sequences, the dataset can still be split into training and test sets that reflect a similar variety and frequency of scenes.
Whilst "JRDB is the only multi-modal dataset that includes indoor and outdoor data, with the largest number of annotated 2D bounding boxes around pedestrians and second largest number of annotated 3D bounding boxes"[26], it still has some artifacts that need to be mentioned. One artifact in the data may come from the difference between outdoor and indoor scenes, meaning that indoors people are mostly in close proximity to the robot as well as trajectories in outdoor settings are more linear and regular due to less obstacles and more space. Another bias is that trajectories in closer proximity tend to be at the front or back of the robot on a specific axis relative to the robot due to the fact that the it moves in corridors or on pathways. Moreover, the dataset offers challenging settings for e.g. 3D tracking due to cluttered scenes, large variations of distances or crowded scenes as well as occlusions.
To sum up, the JRDB dataset offers consistently annotated pedestrians in 2D and 3D in different settings from an ego-vehicle view and differs from other datasets by the close proximity of pedestrians and large variations of behaviour and scenes.

### 2.3.2 Crowdbot

Crowdbot [27] is a multi-modal dataset that captures 2D and 3D data from a moving robot in a city environment. Moreover, it offers detections of pedestrians from the egocentric view with different amounts of pedestrians per scene. This dataset was chosen because it is a large dataset with more than 250.000 recorded frames in over 200 minutes and fulfilling of the use case relevant criteria. Though the dataset was collected in order to detect and track pedestrians in crowded scenarios, it fits best to the intended use case. The dataset can be described as an open space scenario, since it is recorded in the pedestrian area of a city where there are few obstacles except other humans.
The dataset is offered as Rosbag files (recorded sensor data from the robot) to each recorded

sequence and contains the problem relevant RGB data, Point Clouds and depth images (see Figure 2.8).

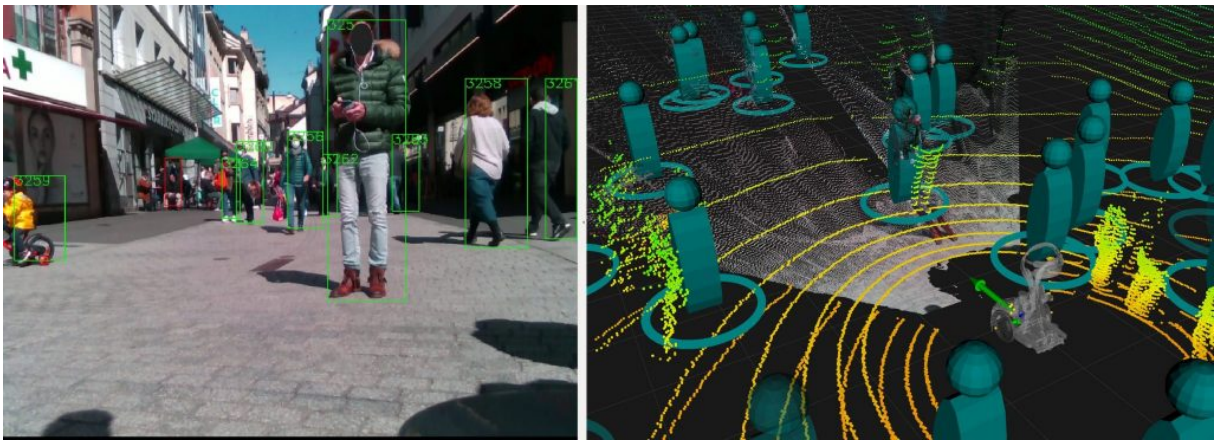

**Figure 2.8:** A sample frame from the Crowdbot dataset with visualised 2D bounding boxes for each pedestrian detected in the scene (left side) and detected 3D positions relative to the movable robot from the pointcloud of this scene (right side).

Though the dataset is also offered as easily accessible .npy format, it misses image as well as keypoint data and therefore I decided to take the Rosbags and do the pre-processing to tracks with pose and position data by myself (see section 3.2. The data is collected by sensors on the robot Qolo which is not an autonomous mobile robot but an assistance exoskeleton on wheels [28]. The relevant 2D and 3D data are collected by a frontal facing RGBD camera mounted on the Qolo and by a frontal and rear 3D LIDAR. Moreover the Qolo is equipped with real-time pedestrian detectors for 2D and 3D data.
The DrSPAAM detector [29] is a self-supervised deep learning approach that detects people from 2D range data. The detector uses the output of an image-based detector to improve its detection in the range data. Moreover, the detected pedestrians are tracked in real time with AB3DMOT [30], a 3D multi-object tracking approach. AB3DMOT is especially designed to work in real-time and shows comparable detection accuracies to more complex approaches in 2D and 3D and can therefore be also applied to image data.
To sum up, Crowdbot offers pedestrian detections from 2D range data and tracks these detections as well as 2D bounding boxes from RGB images.

# Chapter 3

# Methodology

The methodology chapter discusses most of the work done for the data pre-processing, in order to acquire the relevant features. The implementation of the HST and non-deep learning methods are explained in detail. Further, the generation of an own synthetic dataset that is later used in the analysis is described.

## 3.1 JRDB Pre-processing

The original JRDB dataset, as well as the JRDB-Pose dataset (that offers additional skeletal keypoints), is primarily used as a detection benchmark. The latest adaption JRDB-Traj, that is especially designed for trajectory prediction, is not published so far. Therefore, it was necessary to manually merge the necessary features to single pedestrian tracks (trajectories) together. Inspired by the original HST implementation [31] that uses a similar pre-processing step but with other features, I firstly split the JRDB dataset by scene names to achieve approximately a split of 80 percent training data and 20 percent test data.
For each scene I extracted the wanted features, grouped them by the unique pedestrian IDs that are offered by the JRDB dataset, and created tracks from that. This is done with the 3D labels and pedestrian 3D bounding boxes that are described by their centroid position in 3D, rotation in 3D and length, width and height of the bounding box for each time step. Secondly, from the 2D labels for each of the five cameras, the 17 2D COCO skeletal keypoints are extracted with their corresponding pedestrian ID and the time step. Lastly, the odometry data is obtained for each time step and the rotation and translation in 3D space is collected from that.
Each of the three collected datasets groups the features by the unique pedestrian id and therefore generates tracks of the respective feature. The features are then combined in one dataset based on the pedestrian id and are filtered when timestamps do not correspond between features. The resulting dataset contains per pedestrian tracks of all features.
One important aspect in the pre-processing of the JRDB dataset is that the detected persons in 3D are in a local robot frame and not in a global world frame (see section 2.3.1). This means that the located pedestrians are described by their relative position to the robot and if the robot moves, the robot frame moves too. For trajectory prediction, this may cause problems since there would be no difference in trajectories between a moving robot and a standing pedestrian with a

standing robot and a moving pedestrian (assuming the same movement). While the JRDB-Traj dataset [32] is especially designed as benchmark for trajectory prediction, its objective is to predict agents relative to the robot. In contrast, the HST [21] transforms the sensor data (2D position and keypoints) in order to compensate movement of the robot by using the odometry data.
The odometry gives the robot's position and orientation in the world frame, with the orientation as a quaternion to encode the 3D rotation. The original HST implementation [31] uses the 6DOF (degrees of freedom) pose of the robot, i.e. three-dimensional position and orientation, from each trajectories first time step and the main idea is then to transfer the agents position from their coordinate system to the odometry coordinate system. The important thing is that by taking the first odometry data point as world pose of the robot and then taking each incoming movement as transformation onto the agents position, the agents movement is kept in a static environment and compensates the robots movement. These steps can be summarized as follows and are the last step of the pre-processing that is done on the merged tracks of features:

**Algorithm 4:** Transformation of positions from robot to world frame

**1.** Get robot's 6DOF pose in world frame (from first odometry data in the trajectory), with the rotation $R_{robot} = [x, y, z, w]$ , described as a quaternion with $x, y, z$ being the coefficients for the imaginary components and $w$ being the real component, and the translation $T_{robot} = [x, y, z]$:

$$P_{robot} = (R_{robot}, T_{robot})$$

**2.** Get the inverse of the inital robot's pose to be able to map towards world space:

$$P^{-1}_{robot} = (R^{-1}_{robot}, -R^{-1}_{robot} T_{robot})$$

**3.** Get agent's 6DOF pose at current time step in local frame and therefore relative to the robots pose:

$$P_{agent,t} = (R_{agent,t}, T_{agent,t})$$

**4.** Get robot's 6DOF pose at current time step:

$$P_{robot,t} = (R_{agent,t}, T_{agent,t})$$

**5.** Compute agent's position in world frame by applying robot's inverse starting pose and current pose and with that compensating robot's overall movement:

$$P_{agent,t} = P_{agent,t} * P_{robot,t} * P^{-1}_{robot}$$

## 3.2 Crowdbot Pre-processing

Since the processed Crowdbot data is missing skeletal keypoints that are necessary for the Human Scene Transformer, I have done the pre-processing from raw Rosbags (library for recorded sensor data from the robot) to pedestrian trajectories with skeletal keypoints and position data by myself and therefore created a pipeline that can be used for other Rosbags from different mobile robots that have similar sensors. The general pipeline for processing the data looks like the following:

**Algorithm 5:** Pipeline from Rosbags to pedestrian trajectories

**1.** Read relevant messages by topics from the Rosbag: front raw images, detected persons and odometry.
**2.** Get detection data from corresponding messages and extract 3D position, 2D bounding box and confidence score in bounding box together with timestamp.
**3.** Get RGB images with timestamp from corresponding messages.
**4.** Get odometry data with timestamp form corresponding messages.
**5.** Create one DataFrame (2D tabular data) with all features, grouped by each timestamp.
**6.** Create tracks of synchronized features based on the detected bounding boxes - with motpy Python library.
**7.** Extend the track features by estimating skeletal keypoints with MoveNet based on the image and bounding box.
**8.** Filter tracks by amount of detected keypoints with enough confidence (50 percent of all keypoints should be detected with at least a confidence of 50 percent) and filter by sequence length to have at least 15 steps per track.
**9.** Transform positions from local to world frame (similarly to pre-processing of JRDB dataset).

While most of the data, such as pedestrian detections as well as odometry data, is already given, the dataset states that it provides tracks created with AB3DMOT [30] but due to missing documentation it was not obvious how and if the tracks are provided in the raw Rosbags. Moreover, the dataset does not offer skeletal keypoints and therefore the tracking and pose estimation had to be done manually. This involves research over possible methods and explanation for the decisions made.

### 3.2.1 Tracking

Since the detected bounding boxes as well as positions in 3D are already given by the dataset, one should use this information to create tracks and re-identify pedestrians because pedestrian ids and tracks are not given. Therefore one needs a tracker, that can take the given detections in each frame and combine them to a trajectory over time for each pedestrian. There are many already implemented trackers available one could use, but especially those not depending on a deep learning method are updated based on creating new bounding boxes and matching with the ones from the frame before. This means that pedestrians are detected and then tracked based on the image frame. For creating tracks this is a valid method, but sub-optimal for the Crowdbot data because the 2D and 3D detections (2D bounding box and 3D position) are correlated and this information would get lost if the first detection is taken and then further tracked only in image space. One tracker that takes pre-detected bounding boxes as input is the multi-object tracker motpy [33]. This tracker takes the detected bounding boxes with a corresponding confidence score per frame as input and then uses Intersection Over Union (IOU) and a feature similarity matching strategy to track the bounding boxes from one frame to another.
This tracking seems sufficient for the given dataset and due to its simplicity it generates fast

tracks. Nevertheless, it does not output consistent tracks, meaning especially in crowded scenarios where there are many people detected nearby, the tracker can fail by not being able to handle occluded pedestrians. This is no disadvantage for the intended use case, since crowds are not assumed to appear. When implementing the pipeline for other scenarios or use cases there are more accurate trackers or re-identification methods like DeepSort [34], BotSort [35] or STAM-MOT[36]. While DeepSort and BotSort are handling re-identification along tracking as well as taking visual features into account, STAM-Mot may be beneficial for crowds since it is designed to handle partial occlusions.

### 3.2.2 Pose Estimation

Human pose estimation is a research field of its own with mainly deep learning approaches as state-of-the-art. In order to determine skeletal keypoints from given images and detected pedestrians by bounding boxes, a method that is either directly applicable or has a pre-trained model available is necessary. Training such a method would extend the scope of this thesis, therefore a lot of methods had to be filtered out.
The Human Scene Transformer [21] uses BlazePose [37] to estimate skeletal keypoints in real-time from RGB images. The keypoints are given as 3D coordinates and each one has a specific meaning, such as left ear or right foot. For similarity reasons those are in the COCO style which means that there are in total 17 keypoints in a defined order (see Figure 3.1).

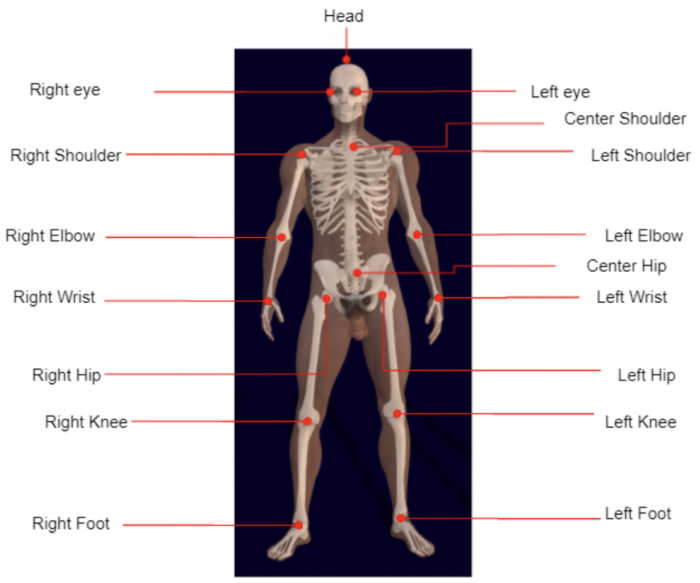


**Figure 3.1:** The 17 keypoints that are defined by the COCO dataset. Picture taken from JRDB-Pose official website [38] to explain the annotated joint locations of JRDB-Pose.

Due to varying distances and occlusions in the JRDB dataset, BlazePose could only estimate around 50 percent of the keypoints [21] and trying it on the Crowdbot dataset, it seems even

less from a first view (see Figure 3.2).

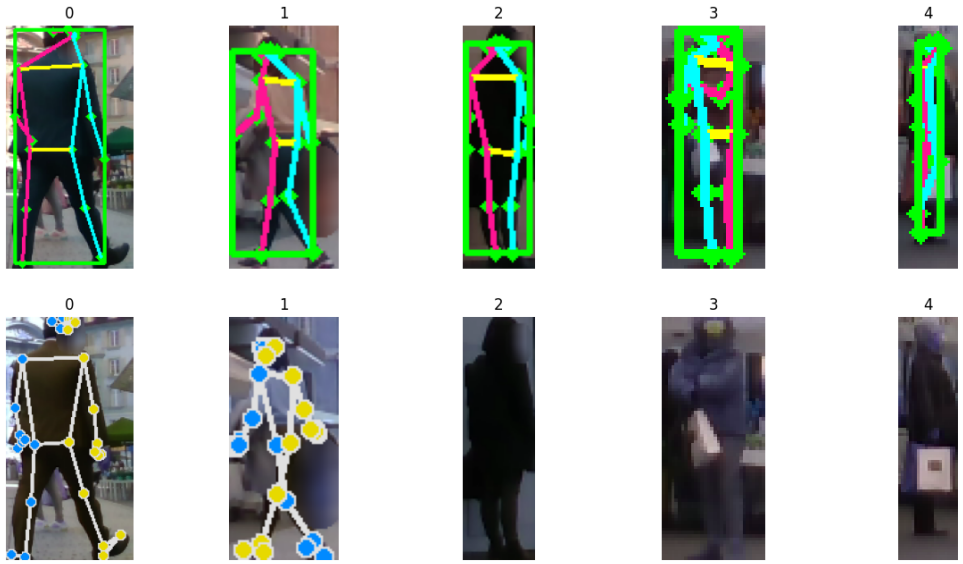


**Figure 3.2:** A comparison of skeletal keypoint estimation by MoveNet (first row) and BlazePose (second row) . Both methods should predict the same skeletal keypoints (Coco style) on five samples of the Crowdbot dataset. The detected left and right side of the body are shown in different colors. Each pedestrian is a cutout of the whole image based on the detected bounding box. The MoveNet predicts a bounding box as well (green box).

That is the reason why I chose another real-time applicable pose estimation net, namely MoveNet [39] which is easily applicable due to an implementation with a pre-trained model by TensorFlow [40]. Despite its easy application, the method is designed to be sufficiently accurate on real-time detections, Moreover, it can detect keypoints on partially visible bodies which is an advantage for occlusions and since it is trained on moving bodies, also fits to walking pedestrians.

## 3.3 Synthetic Dataset Generation

One large known problem in training neural networks is the lack of datasets. Though there are large benchmarks for pedestrian trajectory prediction, like JRDB [32] or for autonomous driving TrajNet [3], each is designed for a specific task. The JRDB dataset is especially designed for many interacting people and is therefore not perfectly fitting the use case. Since this lack of problem-fitting data is a general problem, using self-generated synthetic datasets for training a neural network has become a common solution.
On the one hand, one can use synthetic data to train a neural network in general for the intended problem and then use real data to fine-tune. This is shown to result in better performance than training only on real data [41] and in capturing variability in real-world data. Moreover, Fabbri et al. [42] show that with a diverse and large synthetic datasets like their MOTSynth, real data for training is not necessary and the model trained on synthetic data achieves comparable results

on real data. On the other hand, synthetic data can also be used to highlight rare cases in data by extending these scenarios to capture them better [43].
Inspired by this, and with the fact that none of the real-world datasets does exactly match the intended use case, I decided to create a simple, synthetic dataset that should represent the major trajectories one would expect in an open space scenario. The assumptions therefore are that there is a quadratic environment with a height and width of 10 meters, there is only one pedestrian at a time and no obstacles or other context. Inspired by the social force model [1] and intuitive thoughts of how people may behave in such scenarios, the pedestrian in the synthetic dataset has an internal goal inside the environment that they are heading towards. This goal-directed behaviour is one force of the social force model and sufficient for the intended use case, since the other forces are designed to handle interactions and context.
In order to match real world data, especially the JRDB dataset, the steps are made in a 0.4 second time step and the pedestrian has differing velocities in different trajectories. Moreover, some noise in direction and velocity is added to simulate the non-linear movement of humans as well as noisy sensors better.

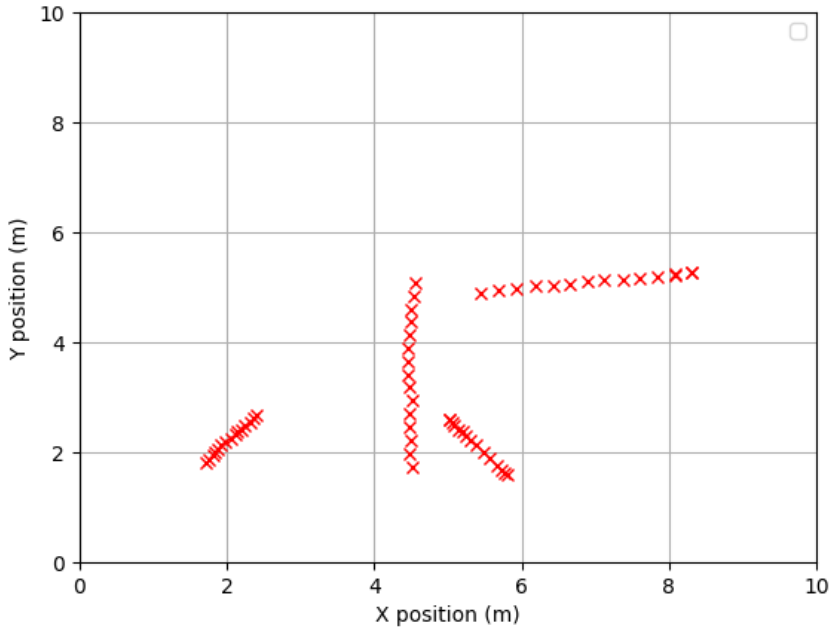


**Figure 3.3:** Five sample trajectories (generated individually but plotted in one figure) of the synthetic dataset where pedestrians have different speeds visualised by the length and proximity of the trajectory steps.

With that, I created a dataset of 256.000 trajectories where the positions are given as 2D coordinates that can be interpreted as positions on a ground plane in meters. The size of the dataset is limited based on given hard- and software limitations.

## 3.4 HST Adaptation and Implementation

The Human Scene Transformer is designed to predict a joint movement of multiple pedestrians in a scene by predicting the center of their bounding boxes on the image of the scene. For the intended use case, a single trajectory needs to be predicted and since there are no neighbours, the interaction does not need to be modelled and the HST can be decreased in complexity. Moreover, predicting the next position on the image does not seem to be the best method when intending to use the predicted trajectory for path planning of the mobile robot. It therefore seems more beneficial to directly use the position in the world space, i.e. where the pedestrians are moving to in the same coordinate system as the robot operates. Predictions in image space may be used for that too but then a post-processing is necessary to map from camera to world space.
Therefore, I adapted the official implementation of the Human Scene Transformer [31] by implementing it for single trajectories and made it applicable to the chosen datasets [10]. An overall adaption from the original implementation was the simplified handling of the features. The idea of processing all wanted features and intermediate results in a dictionary is kept but simplified and unnecessary transformations due to naming conventions are left out. Moreover, the agent dimension is removed in the inserted features as well as in the dimension handling attention mechanisms and other processing steps operating. To adapt the HST to work on 3D positions instead of 2D, adaptions for creating the loss and metrics from the Gaussian Mixture Models have to be implemented.

### 3.4.1 Input

Assuming a processed dataset with detected persons in 2D, 3D and matching keypoints to individual pedestrians, the data still needs some processing before it can be used for training. One important step is to set a fixed length of time steps that are given as input. The HST training assumes for the JRDB dataset six history steps and predicts the next 13 from that. Due to the limited data I chose a history length of five time steps to predict the next ten steps and therefore need only a sequence of length 15. To get the most training samples from the limited data, I windowed the trajectories to each a sequence length of 15. That means for a longer trajectory the first 15 steps are taken for one trajectory, three steps are skipped and the next 15 steps form another trajectory and therefore one gets multiple training samples from one trajectory. The HST also sub-samples the data to reduce its size and computational complexity by taking e.g. for the JRDB dataset only every fifth frame. Due to the limitation of training samples (see Figure 3.4) I did not implement the sub-sampling to keep as much data as possible.

### 3.4.2 Agent Self-Alignment

The motivation for an agent self-alignment layer stems from the fact that the HST handles whole scenes with multiple agents at once and inputs the target agent with its neighbours. Due to the fact that nearby agents lead to identical neighborhoods of agents even though the target agent is a different one, an identifier is needed to let the model know which agent is in focus. The agent self-alignment layer makes sure that the agents each know their own history, since

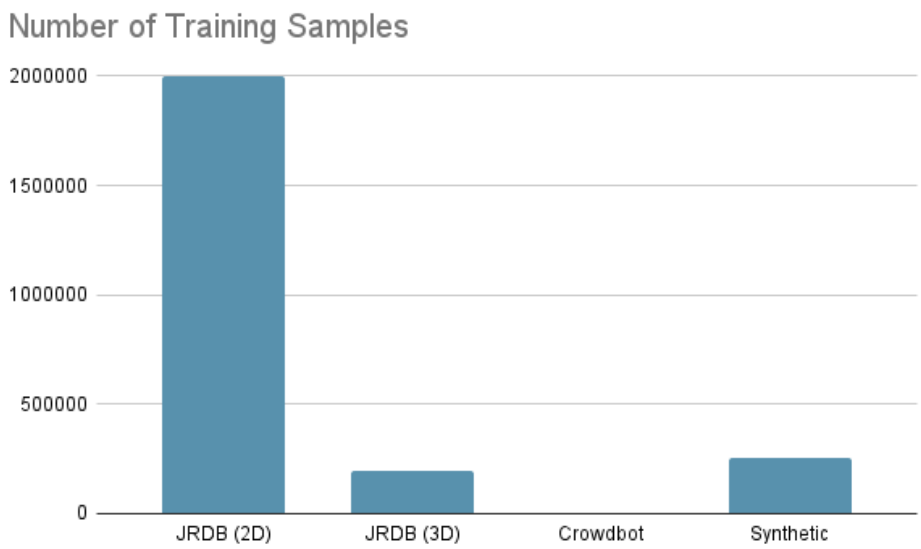


**Figure 3.4:** The number of training samples after extracting the relevant features for each dataset and augmenting it, therefore doubling the amount of samples. JRDB (2D) has 2.000.000 sample trajectories with 2D position coordinates and skeletal keypoints, JRDB (3D) has 192.000 sample trajectories with 3D position coordinates and skeletal keypoints, Crowdbot has 2.200 sample trajectories with 3D position coordinates and skeletal keypoints and the synthetic dataset has 256.000 sample trajectories with 2D coordinates.

temporal information typically gets lost in transformers.
That is realised by multi-head attention over the temporal dimension that allows to align each agent's current state with its past states and therefore capturing temporal dependencies. Such a method is useful in settings where agents need to be aware of their own history and therefore still useful in a single agent setting since the core functionality of giving awareness of one's own history is independent from the number of agents. That is the reason why this part that is intentionally designed for the special case of interactions and inputting neighbouring agents is still a benefit and kept in the implementation.

### 3.4.3 Multi-Modal Trajectory Distribution

One main contribution of the Human Scene Transformer to a better pedestrian trajectory prediction is the introduction of different modes. The idea of modes is that they capture different possible behaviours. Especially in interactions people behave differently from person to person, so the different modes should capture different behaviour patterns and the network should learn which one is most likely for each scenario.
This is implemented after some processing of the input via transformer layer, by simply repeating the current tokens (or hidden vectors) by the number of modes wanted. While the JRDB dataset was trained on four or five modes, the ETH/UCY datset was trained on 20 modes. Therefore the number of modes is a hyperparameter that has to be adapted to the dataset at

hand and is set to five for the adapted version.

### 3.4.4 Training Pipeline

The learning process of the Human Scene Transformer based on the official HST implementation [31] that uses the TensorFlow pipeline for training neural networks, to keep it as close to the theory and as comparable as possible. Therefore, the loss and learning rate are implemented manually instead of using a given implementation from TensorFlow. Additionally, the TensorBoard is used for visualising intermediate steps and the learning process. The training can be optimized and adapted towards the specific datasets in terms of the hyperparameters that are the batch size, the number of batches per training step, the number of batches per evaluation step, the number of steps in between evaluation steps, the learning rate parameters and the total amount of training samples.

#### Learning Rate Schedule

The official HST implementation [31] uses a learning rate schedule instead of a fixed learning rate. With that, the learning rate increases for a given amount of warm-up steps and therefore starts the learning process slowly. This may be an advantage in the early stages of training when large gradients can cause unstable updates.
The JRDB dataset is trained with a peak learning rate of $10^{-4}$ and $5 \cdot 10^4$ warm-up steps with a total of $5 \cdot 10^6$ training samples, whereas the ETH/UCY dataset is trained with a peak learning rate of $5 \cdot 10^{-4}$ and $5 \cdot 10^4$ warm-up steps with a total of $10^6$ training samples.

#### Loss

The loss function used for training the HST in the implementation is called "MinNLLPosition-MixtureCategoricalCrossentropyLoss". The overall idea is to get the mode that best fits the ground truth trajectory, so which outputted probability distribution for each future step is closest, and compute the negative log-likelihood between the prediction and ground truth.
The best fitting mode $m^*$, so the behaviour pattern that matches best the pedestrian (or scene in the original case) for the the single agent use case is defined by:

$$m^* = argmin_m(\sum_t -log(\mathcal{N}(x_t^*, \mu_{m,t}, \sigma_{m,t})) \tag{3.1}$$

The best mode $m^*$ is found by minimizing the cumulative negative log-likelihood across every time step $t$. The output of the HST is not a discrete position but a mean and a variance that represent the probability of the position by a Gaussian Mixture Model (GMM). A GMM is a probabilistic model that describes a complex distribution by a combination of gaussian distributions [44]. This combination is a weighted sum of the single probability distributions where each distribution has its own mean and variance. The weights for each probability distribution are given by the mode probabilities generated in the multi-modality induction and therefore the single models are representing each one mode. The probability distribution of one mode can therefore be defined as $\mathcal{N}(x_t^*, \mu_{m,t}, \sigma_{m,t})$ for a time step $t$, the ground truth position $x_t^*$, the predicted mean $\mu_{m,t}$ and variance $\sigma_{m,t}$ at that time step and for the specific mode $m$.

The negative log-likelihood is a commonly used metric for probabilistic models like the GMM and simply measures how good the predicted distribution fits to the actual data because the better the probability distribution can explain the real data, the lower the negative log-likelihood. Therefore the best mode $m^*$ is the one that on average has for every time step the best probability distribution explaining the actual data for each time step. Now the actual loss function is only calculated on the previously defined best mode and due to the probabilistic nature, the loss is simply the negative log-likelihood of the the best fitting mode $m^*$ and due to its definition as the minimum, the loss is called minimum negative log likelihood. In computational terms, one simply reuses the previously calculated negative log-likelihood for the mode $m^*$ but in general, the loss is defined as:

$$L_{minNLL} = \sum_t -log(\mathcal{N}(x_t^*,\, \mu_{m^*,t},\, \sigma_{m^*,t})) \tag{3.2}$$

Though not mentioned in the paper but used in the implementation [31], the loss is defined as the combination of the previously described minimum negative log-likelihood and a categorical cross-entropy loss for mixture weight. I can only assume that this addition is made to force the model to build different mode behaviours and comparing the accuracy of the model trained with each loss on the synthetic dataset showed improvements with the additional categorical cross-entropy. The additional loss combines again the negative log-likelihood and the categorical cross-entropy loss. The categorical cross-entropy loss computes the loss between a categorical distribution like the mixture weights and the ground truth labels. In the given context this could mean that the probability that each mode is responsible for a given data point is predicted. Like the first described loss, the best mode $m^*$ is found by the formula 3.1.
A mixture loss is then computed using the categorical cross-entropy between the one-hot encoded best mode $m^*$ and the predicted mixture probabilities (so called logits from the multi-modality induction module). The one-hot encoded best mode represents the index of the best mode, while the mixture logits represent the model's confidence in its prediction and therefore the categorical cross-entropy between these two computes the loss for the model assigning the wrong probabilities to the mixture components. With that, the mixture loss probably leads the model to assign higher probabilities to the mode that fits best the ground truth trajectory.

## 3.5 Kalman Filter

By facing a scenario like the proposed open space scenario, one could assume that especially for single pedestrians without interactions, the trajectories may not be that complex. Therefore, taking simpler Bayes models like the Kalman Filter with a constant velocity (CV) model into account is reasonable. From an intuitive view, pedestrians in an open space scenario are following a path that is either straight or have a smooth turn but are rarely making unexpected sharp turns. From that assumption ongoing, I implemented the CV model to predict the next ten time steps given the first five, similarly to the HST input for comparison reasons.
Since the position of the pedestrians is given in world space as 3D coordinate from the different datasets (JRDB, Crowdbot) and fits to the idea of predicting the trajectories in world space (on a ground plane relative to the robot's position), the state space consists of positions and velocities in 3D: $state = [x, y, z, v_x, v_y, v_z]$. For the constant velocity model, the assumption

is that the acceleration is almost zero and the velocity therefore constant. The corresponding state transition matrix is built so that the new position is the old one plus the velocity times the time step $T$, while the velocity itself stays the same. The system, so the pedestrian trajectory, therefore evolves according to the following matrix with $T$ being the time between the steps:

$$A = \begin{bmatrix} 1 & 0 & 0 & T & 0 & 0 \\ 0 & 1 & 0 & 0 & T & 0 \\ 0 & 0 & 1 & 0 & 0 & T \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

With having only the position as new measurement each time step, the measurement matrix $B$ maps the state space to the measurements only from these variables:

$$B = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{3.4}$$

The process noise matrix $Q$ should represent the uncertainty in the CV model and should therefore capture small noisy changes that are not directly captured by the constant velocity assumption. Since the KF assumes the noise to be gaussian, the uncertainty can be represented as variance. Based on the implementation of Labbe [14], I decided to also assume independent noise in the position (upper left variance block in the equation 3.5) and velocity (lower right variance block in the equation 3.5), so that noise can affect position and velocity but there is no correlation between them. The variance *var* in the process noise matrix $Q$, is a changeable hyperparameter and controls the amount of noise added to each time step. A higher value for the variance would represent that one expects more variation in the actual state and that one relies more on the measurements than the predicted state. This results in the following process noise matrix $Q$:

$$Q = \begin{bmatrix} 0 & var & var & 0 & 0 & 0 \\ var & var & var & 0 & 0 & 0 \\ var & var & var & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & var & var \\ 0 & 0 & 0 & var & var & var \\ 0 & 0 & 0 & var & var & var \end{bmatrix} \tag{3.5}$$

For the overall implementation of the KF, I used the filterpy [45] package that offers the Kalman Filter logic and one can design the models (like here the CV model) on their own and adapt it to the intended use case.

## 3.6 Interacting Multiple Models (IMM)

Since there are various approaches using the IMM approach with a constant velocity model and a coordinated turn rate model for similar problems like pedestrian path prediction [15], trajectory prediction for anticipative human following [18] and quick maneuver detection [46], I decided to implement these two models with the IMM. Moreover, the combination of straight-line and curved movements suits the idea that pedestrians are mostly following a goal and making turns if necessary, e.g. when there are obstacles.
Due to the movements of pedestrians being mostly two-dimensional, the state space can be described in terms of 2D motion. This is especially an advantage for the CT model, since turning is happening in x-y coordinates for pedestrian trajectories. The system's state should cover all variables necessary for the different filters and therefore contains the position, velocity and the turn rate (for the 2D-CT model): $x = [x, y, z, v_x, v_y, w]$.
The first model implemented for the IMM is the Kalman Filter with a constant velocity model. Therefore, the key points are like described in section 2.1.2 except that the matrices are adapted to fit to the five-dimensional state vector by adding a dimension for the turn rate. Nevertheless, since the constant velocity model does not take the turn into account, the variable does not have an impact and is handled as a static, non-changing variable.
The second model is a Kalman Filter with a constant turn rate model. For simplicity reasons and due to the limit of the master thesis, the constant turn rate (CT) model is simplified to work with a normal Kalman Filter by using the Jacobian matrix, that linearizes the non-linear system. This is applicable in this case because the pedestrians are moving in small time steps where non-linear behaviours can be approximated by linear behaviour due to their small change. The turn rate in the CT model describes the constant angular velocity and therefore influences the velocity in x and y direction but only the direction and not the magnitude.
In general, rotation in 2D space by an angle $\alpha$ is described as:

$$\begin{bmatrix} cos(\alpha) & -sin(\alpha) \\ sin(\alpha) & cos(\alpha) \end{bmatrix} \tag{3.6}$$

The rotation angle $\alpha$ can also be described as $\alpha = w\Delta t$, so the angular velocity during the time $t$ and the equation 3.6 then describes the rotation of the velocity in 2D. Now, the derivative with respect to the state vector, also called Jacobian, describes how the velocity changes over time and can therefore be applied to the state transition matrix:

$$A = \begin{bmatrix} 1 & 0 & \frac{sin(w\Delta t)}{w} & -\frac{1-cos(w\Delta t)}{w} & 0 \\ 0 & 1 & \frac{1-cos(w\Delta t)}{w} & \frac{sin(w\Delta t)}{w} & 0 \\ 0 & 0 & cos(w\Delta t) & -sin(w\Delta t) & 0 \\ 0 & 0 & sin(w\Delta t) & cos(w\Delta t) & 0 \\ 0 & 0 & 0 & var & 1 \end{bmatrix} \tag{3.7}$$

From the state transition matrix $A$ (from equation 3.7) one can therefore see that the velocity in $x$ and $y$ direction is updated based on the rotation described in equation 3.6 and the position

based on the derivative of that. Since the turn rate is fixed and not known beforehand, the IMM is implemented with multiple CT models with different turn rates to overcome that limitation.

Again, the implementation is built upon the filterpy package [45] and uses an IMMEstimator to work with the build CT and CV models. The IMMEstimator is initialized with the knowledge about the first position as well as the velocity between the first two steps in order to get the best state assumptions from the beginning. The states are estimated and updated for the first five time steps and then only predicted for ten additional steps. Since this is still a simple model and the data is not perfectly fitting the assumed scenario, I added, inspired by the Probabilistic Kalman Filter [47], a smoothing of the predicted states. This basically adjusts the estimated beliefs by the previous ones and a chosen smoothing factor $\alpha$:

$$smoothedPredictions[t] = \alpha \cdot predictions[t] + (1 - \alpha) \cdot smoothedPredictions[t - 1] \quad (3.8)$$

With that the original prediction is blended with the previous smoothed prediction, while the first smoothed prediction stays the original one. This smoothing yields better results, especially for shorter history steps in order to take the history into account and removes noise.
In order to receive the best results, hyperparameters like the process noise $q$, measurement noise $r$, initial covariance $P$, a transition matrix $M$, time step $\Delta t$, and the omega variance, related to the turn rate noise, can be tuned to fit best to the used dataset. Therefore, I implemented a hyperparameter tuning where one can define different possible options for the single parameters and make use of the python library sklearn that offers a parameter grid where all possible combinations of parameters are tested. Based on the average displacement error (ADE), the set with the lowest ADE is chosen (see section 3.7.1).

## 3.7 Particle Filter (PF)

Though the Particle Filter is a flexible method for estimating system states, it is usually not used for trajectory prediction as it is. Like discussed before in section 2.1.4 there are several approaches to tackle the prediction, where the main focus lies on choosing the most accurate process model, that describes the system's dynamics. Though there are models especially designed for pedestrian behaviour, like a social behaviour model [48], there are few usable open source implementations to use them. Therefore and in order to keep to the scope of this thesis, I decided to implement a Particle Filter for trajectory prediction with a simple constant velocity model. Although this model does not predict human-like motion well, it fits to the thesis' idea of comparing simple versus complex algorithms. The overall prediction algorithm can therefore be extended by other models but can in this thesis be compared to the simple KF with a CV model as a first impression.
Since the complexity and computational resources increase with the amount of state dimensions, I chose to only implement the PF for the 2D case, where the position is derived from the 3D position and represents the position on an x-y ground plane. The state of each particle can therefore be defined by the x and y position as well as the respective velocities which fits to the use case where pedestrians are moving on the ground plane. Moreover, with the knowledge of the first few steps, the particles are not randomly initialized but, inspired by [19], uniformly distributed around the initial position. The velocity is estimated from the first two observations,

since the model assumes constant velocity.
The implemented algorithm follows the generic algorithm presented in section 2.1.4 and adapts the PF implementation for target tracking presented by Labbe with the filterpy library in [14]. While Labbe used the measurements to update the particles, I implemented a simple constant velocity model to update the movements. The algorithm is then implemented as described: The weights are updated based on their proximity to the measured position, the particles are resampled according to their weights and the state for the specific time step is computed by as a weighted average of all particles. Moreover, the implemented PF algorithm updates and resamples only for the given amount of time steps and then only predicts the next particles based on the CV model.
All in all, even before testing one cannot assume the best accuracies in prediction since the underlying CV model is too simple, but it can serve for a comparison to the simple CV model as well as a first implementation that can later be extended.

### 3.7.1 Hyperparameter Tuning

The classical methods can be adapted to the specific behaviour of a dataset by choosing the best fitting hyperparameters. In order to get the best out of the implemented methods I therefore implemented a hyperparameter tuning that applies different sets of parameters and chooses the set that achieves the best prediction accuracies on the respective test dataset. This is done with the help of the sklearn package that offers a parameter grid where one only has to define all possibilities for each parameter and the grid creates all possible sets out of that.
For example, the Kalman Filter can be adapted by the process noise covariance $q$, the measurement noise covariance $r$, the initial estimation covariance $P$ and the time step between predictions. Having prior knowledge about the datasets and therefore the data distribution may help defining the proper parameters, else one has to test additional values. The process noise $q$ controls the uncertainty in the process model and is therefore responsible for the adaption of this model in-between the single steps, so with a higher value the process model adapts more to variations. The measurement noise $r$ controls the reliance on the measurement data and is therefore responsible for the update step in the algorithm and how much the belief relies on the measurement and how much on the model. The initial covariance $P$ controls the belief in the initial state so if it comes from real measurements it is more likely to fit to the actual state than random variables. The time step $\Delta t$ controls the prediction step, so the rate of prediction updates.
These variables are also found in the Interacting Multiple Model approach as hyperparameters, where additionally the omega variance can be adapted. The omega variance describes the uncertainty in the turn rate and therefore controls how much variance between steps is allowed. The Particle Filter also has the process noise, measurement noise and time step as hyperparameters. Additionally, the number of particles can be set differently to control the range of possible states to take into account but with a higher amount the computational cost also increases.

# Chapter 4

# Experimental Setup

In order to test the different proposed methods on the data, multiple experiments where set up. The requirements for the experiments will be listed in the following section. Further, the used metrics for evaluating the prediction accuracy and computational efficiency are defined.

## 4.1 Hardware and Software Requirements

For the experiments an AMD Ryzen7 5800H with Radeon Graphics processor and a NVIDIA GeForce RTX 3070 GPU was used. Furthermore, the experiments were conducted in a virtual machine that used Ubuntu 20.04. The implementations are mainly based on TensorFlow as well as numpy and pandas, but a whole requirements list can be found in the corresponding GitHub repository [10].

## 4.2 Evaluation Metrics

The methods should be analyzed according to their accuracy as well as computational efficiency. From that, one can conclude which method performs best from prediction accuracy and which one is the fastest and most real-time applicable.

### 4.2.1 Accuracy Metrics

The most common accuracy metrics for trajectory prediction are the average displacement error (ADE) and final displacement error (FDE) [9]. The ADE is defined by the mean squared error between predictions and ground truth points:

$$\text{ADE} = \frac{1}{T} \sum_{t=1}^{T} \|\mathbf{p}_t - \hat{\mathbf{p}}_t\|_2 \tag{4.1}$$

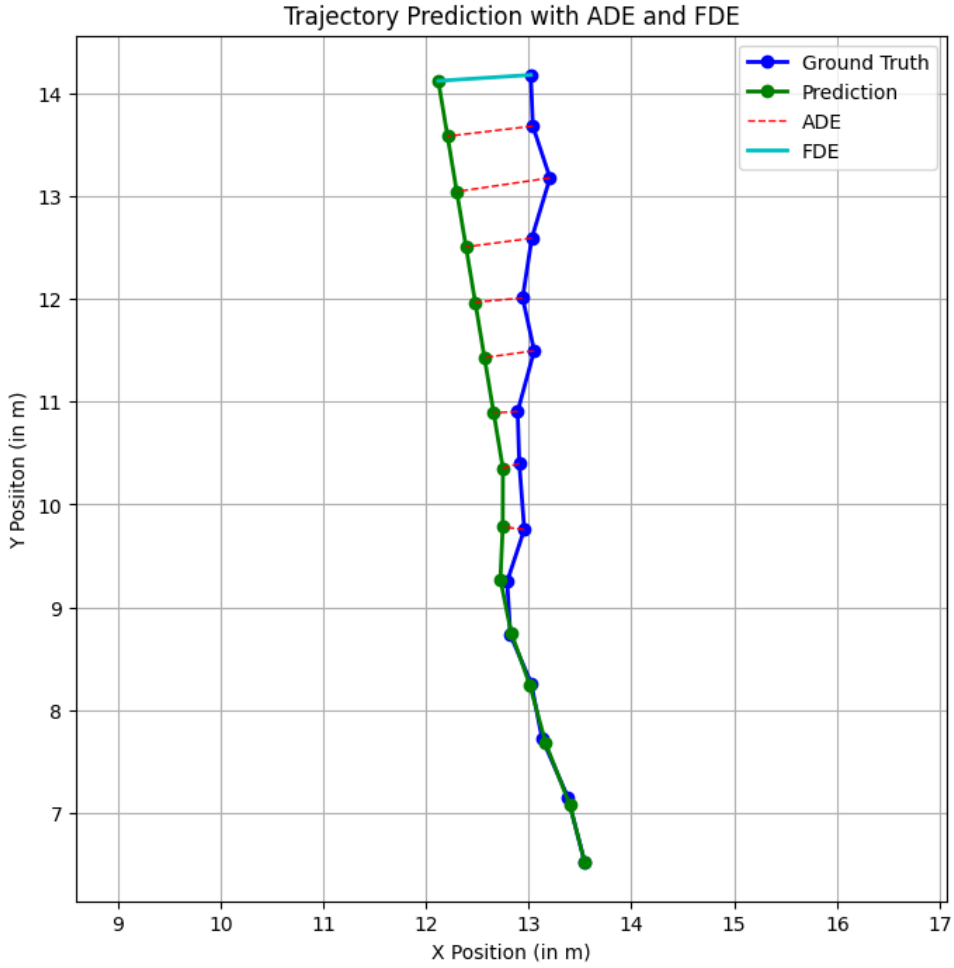


**Figure 4.1:** A visualisation of the average displacement error (ADE) and final displacement error (FDE) between a ground truth trajectory (blue trajectory and dots) and a (with KF) predicted trajectory (green trajectory and dots). The dotted red lines between two points indicate the mean squared error for the ADE and the light blue line between the goal points, the mean square error for the FDE.

The mean square error is therefore taken from the total number of time steps $T$ of the prediction and $\mathbf{p}_t$ as well as $\hat{\mathbf{p}}_t$ represent the ground truth and predicted position at time step $t$ respectively (see Figure 4.1). Therefore for each time step $t$, the euclidean distance $\|\mathbf{p}_t - \hat{\mathbf{p}}_t\|_2$ between the ground truth and predicted position is calculated. The summation then computes the total displacement error. The overall mean is then reached by normalizing it by the total amount of time steps.
The ADE can be seen as a measurement for the overall prediction accuracy where lower values represent a closer match and therefore a better accuracy. Further, for an analysis one has to keep in mind with this metric that through the averaging many different kinds of trajectories can produce the same ADE value and to account for meaningful trajectories one has to also take a look into the outputted trajectory forms compared to the ground truth.

The FDE in contrast focuses only on the last predicted point and measures the distance to the last ground truth point in the given trajectory.

$$\text{FDE} = \|\mathbf{p}_T - \hat{\mathbf{p}}_T\|_2 \tag{4.2}$$

The distance is measured between the ground truth $\mathbf{p}_T$ and predicted final position $\hat{\mathbf{p}}_T$ at the last predicted time step $T$ using the euclidean distance to account for the dimension of the positions (see Figure 4.1). Contrary to the ADE, the FDE focuses solely on the accuracy of the last predicted point and is therefore useful when the destination is important, but can also be used to take a look into the form of trajectory and differentiate between ADE values. If a prediction gets further away from the ground truth over time compared to a prediction that is further away in the beginning but comes closer to the ground truth with each time step, the ADE can have a similar value. In such a case the ADE may average the errors to the same value but the FDE may show that with time the prediction gets better. One still has to investigate the trajectories themselves by hand for such an interpretation and the metrics are only a first indicator of prediction accuracy.
The ADE and FDE are measured on each dataset on the same 1.000 samples of the test dataset split and the average of the metrics over the 1.000 samples is then given. Therefore one could describe it as meanADE and meanFDE.

### 4.2.2 Computational Efficiency

In order to measure computational efficiency, the time to process one trajectory was measured for each approach. This process was done 100 times and the average time to process one trajectory is then given. This kind of metric was chosen since the processing time is the most important aspect besides the accuracy for the analysis of its real-time applicability.

# Chapter 5

# Results and Analysis

In order to compare the different implementations, one needs to analyse them on the given pre-processed datasets with the described metrics and considering different aspects. The implemented methods are the adapted Human Scene Transformer towards single agent trajectories, the Kalman Filter with a constant velocity model, the Interacting Multiple Models method with a constant velocity model and three fixed constant turn rate models as well as the Particle Filter with a constant velocity model. The given pre-processed datasets are the JRDB, Crowdbot and a synthetic dataset. As a metric for accuracy, the average displacement error (ADE) and the final displacement error (FDE) are used. The methods are analysed according to their overall prediction accuracy, as well as on different length of trajectories and the features used. First of all, only the Human Scene Transformer prediction accuracies are analysed on the JRDB and the synthetic dataset. Then, the performance accuracies of all implemented methods are analysed according to the described aspects.

## 5.1 Prediction Accuracy of the HST

The prediction accuracies of the implemented HST are analysed on one real dataset (JRDB) and the synthetic dataset. Though, knowing that the JRDB dataset with 3D positional information only contains 192.000 samples at all and therefore a tenth of the amount of training samples for the JRDB prediction in image space (further called 2D JRDB), I trained my simplified version on it. With a batch size of 32, 200 batches per training step and a total of 4.400 batches, the HST achieves an ADE of 2.89 meters and a FDE of 2.81 meters. For a classification of this value, the JRDB dataset is analysed in terms of total trajectory distance, so how far the pedestrians move in 15 time steps (6 seconds) in Figure 5.1. The figure 5.1 shows that from 70.688 single training samples, more than half (41.340) are in the range of 0 - 50 centimeters. 11.246 samples are in the range of half a meter up to a meter. Therefore approximately 75 percent of the trajectories are covering a distance from 0 to 1 meters in six seconds. With that, the ADE and FDE of 2.8 meters away from the ground truth on average for 75% of the trajectories is far off. This can be seen on some samples of the JRDB test dataset (see Figure 5.2)

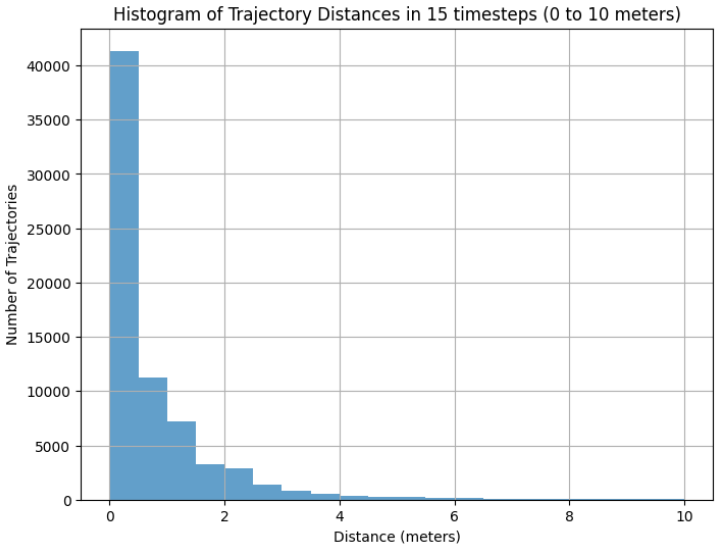


**Figure 5.1:** A histogram of trajectory distances (calculated by the euclidean norm between the first and last position of each trajectory) of all training samples of the JRDB dataset. Every trajectory is 6 seconds long, each bin represents a range of 50 centimeters.

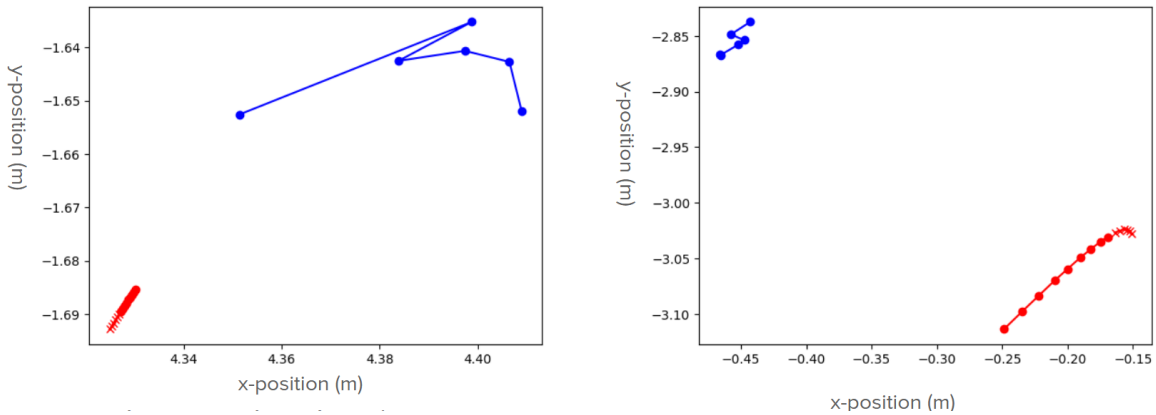


**Figure 5.2:** Two examples of the JRDB test dataset where the red trajectory refers to the ground truth (the red crosses are the first five steps as known history information and the red circles the ten steps to be predicted). The blue trajectories are predicted from the first five ground truth inputs and are connected by their sequential order.

In order to improve the prediction accuracy on the JRDB dataset, the HST as well as the pre-processing of the data were varied but none of the following changes made an improvement in the ADE and FDE and are therefore not described in detail but the idea behind them. Due to the varying scales of the trajectories shown in the figure 5.1, one idea to help the model learn better was to normalize the data by z-score normalisation, so to a mean 0 with standard deviation 1. Normalised data as input to neural networks is in some cases essential to get it

trained, but normally needs a remapping for tasks like the trajectory prediction to output usable positions. That may be the reason why the HST works on unnormalised data but training the model on normalised JRDB data does not make a significant improvement.
If a specific dataset is too small for training a neural network, finetuning is a common solution. The idea is then to train the model on a huge (in the best case, diverse) dataset and then retrain on the specific dataset to adapt it to the intended use case. Therefore, different combinations of finetuning were tried out like firstly on the normalised JRDB dataset and then on Crowdbot or firstly on JRDB and then finetune on the synthetic dataset or the other way around. For easier generalisability, all datasets were normalised with z-score normalisation for the finetuning experiments. Though one cannot compare the metrics between models trained on normalised and unnormalised data, due to different ranges, looking into sample outputs showed that the HST did not learn to produce a reliable trajectory from the given ground truth.
Another idea was to scale the data from meters to centimeters because a lot of trajectories are in the centimeter and millimeter range (see Figure 5.1). With that, differentiating between the steps inside a trajectory should become easier to capture for a neural network, and looking at the intermediate feature embedding of sample trajectories showed that even small steps are clearly distinguishable. Looking at the metrics and sample predicted trajectories, this change did not make an improvement in prediction accuracy. Instead of using the absolute positions for the trajectories, one could also use the relative positions with the first position being at the same starting position (e.g. (0,0,0) for 3D and (0,0) for 2D) and the trajectory is defined as the vector from one step to the next. Intuitively, this may simplify learning for the neural network because it eliminates the variable of the distance relative to the robot and only tracks the direction and speed of the pedestrians but therefore also loses information. Training the HST with relative positions of the JRDB dataset showed similar output trajectories as for the absolute positions and is therefore no improvement. Though the HST did not learn sufficiently to predict trajectories

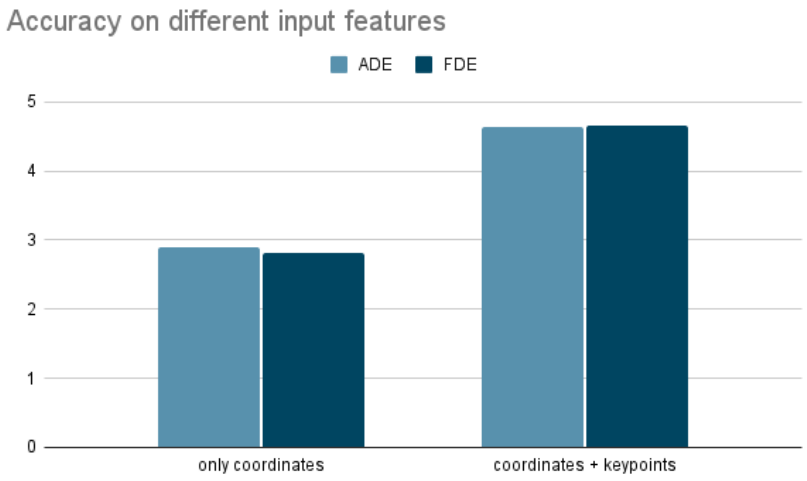


**Figure 5.3:** The accuracy (ADE and FDE in meters) on the JRDB test dataset once trained on only 3D coordinates (left side) and once trained on 3D coordinates and skeletal keypoints (right side).

of the JRDB dataset and one cannot analyse the influence of the skeletal keypoint information on insufficiently predicted trajectories, but analyse if they are an advantage for the current case or not. For the JRDB case this means that the HST is once trained on only 3D coordinates as input feature and once with 3D coordinates plus 3D skeletal keypoints (see Figure 5.3). For the model being only trained on coordinates, the ADE is 2.89 meters and increases with additional skeletal keypoint information to 4.63 meters. The FDE is close to the ADE for both cases (2.81 meters and 4.65 meters respectively).
Now the synthetic dataset consists only of 2D coordinates and therefore the HST can only be trained with positional data as input feature. With a batch size of 32, 400 batches per training step and a total of 8.000 batches, the HST achieves an ADE of 0.59 meters and a FDE of 0.91 meters. Similarly to before, the figure 5.7 shows the average distances of trajectories made in six seconds (15 time steps) and therefore the distribution of pedestrian behaviour. With 75 percent of the trajectories being in the range of 0 to 5.5 meters, the synthetic dataset captures more diverse and longer trajectories than the JRDB dataset and an ADE of 0.59 meters on average is closer to most ground truth trajectories.

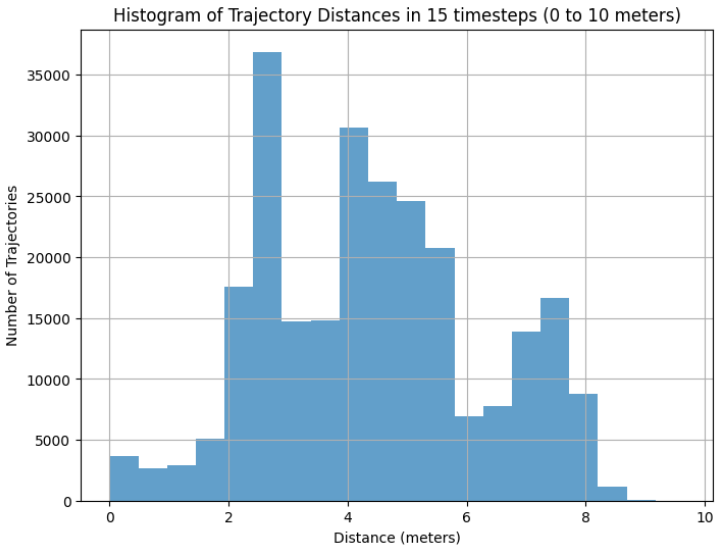


**Figure 5.4:** A histogram of trajectory distances (calculated by the euclidean norm between the first and last position of each trajectory) of all training samples of the synthetic dataset. Every trajectory is six seconds long, each bin represents a range of 50 centimeters.

Visualising some of the predicted trajectories next to the ground truth trajectory (see Figure 5.10) shows how the ADE and FDE values came about and especially how the HST can capture straight trajectories with little noise (left side of the Figure 5.10) better than trajectories with more noise (right side of Figure 5.10) though all trajectories are created with an internal goal that they are heading towards in a mainly straight (but sometimes noisy) manner.

In order to analyse the accuracy on short-time and long-time prediction as well as how the

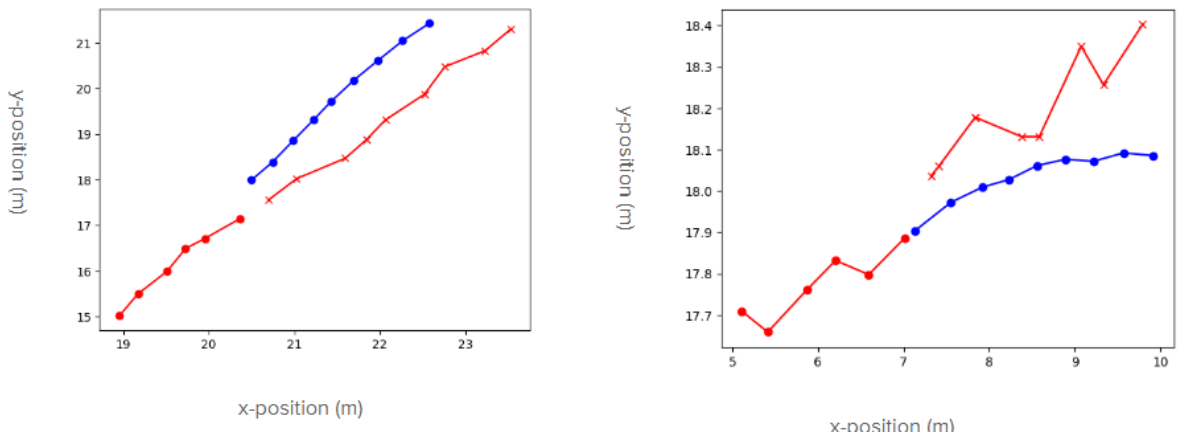


**Figure 5.5:** Two examples of the synthetic test dataset where the red trajectory refers to the ground truth (the red crosses are the first five steps as known history information and the red circles the ten steps to be predicted). The blue trajectories are predicted from the first five ground truth inputs and are connected by their sequential order.

length of history data affects the prediction, the HST was trained on the JRDB and synthetic dataset independently to predict the next seven time steps from the previous eight time steps. Since the pre-processing of the datasets involved a windowing of the sequences to a length of 15 time steps (six seconds), the total sequence length cannot be extended in total.

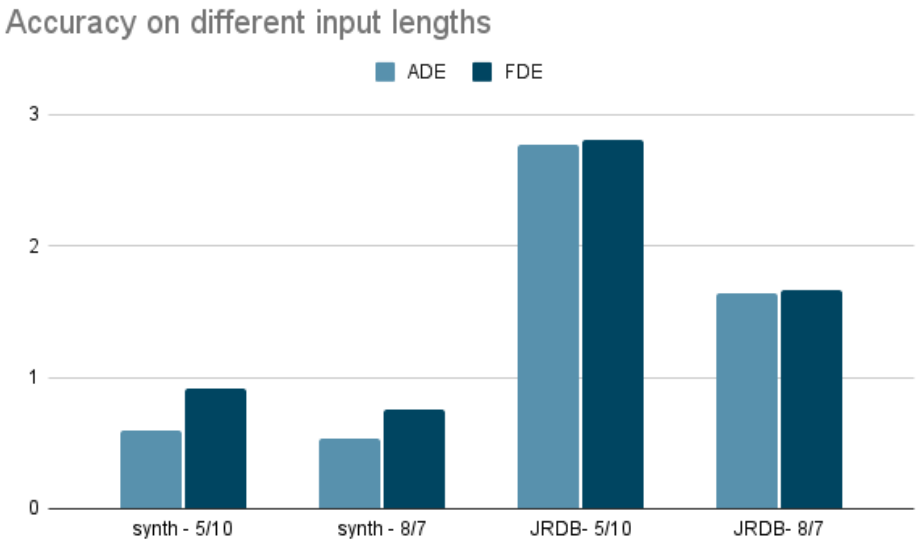


**Figure 5.6:** The prediction accuracy measured by the ADE and FDE in meters on the JRDB and synthetic dataset each once trained on five history time steps and ten time steps to predict (indicated as 5/10) and eight history time steps and seven time steps to predict (indicated as 8/7)

Therefore, to get a first impression of the influence of the history length, a longer history length of eight time steps (3.2 seconds) was chosen but instead of ten time steps, only the next seven can be predicted. Due to the ADE being the average and the FDE only taking the last point into account (disregarding the length of the sequence before), the metrics can be compared between different history lengths.
The HST trained on the synthetic dataset only improves slightly in terms of ADE from 0.59 meters to 0.53 meters and in terms of FDE from 0.91 meters to 0.7 meters, when getting three more time steps as input to predict three fewer points (see Figure 5.6). The HST trained on the JRDB dataset improves more in terms of ADE from 2.77 meters to 1.64 meters and in terms of FDE from 2.81 meters to 1.66 meters, when getting three more time steps as input to predict three fewer points. That shows that especially the FDE improves with more history information. In terms of JRDB the ADE and FDE improve by a large margin even though the predicted trajectories are from their form similar to those outputted with less information but only a meter in average closer to the ground truth.
One last point to mention is, that in every case described before, there are no differences between the modes. That means that the mode probabilities are very small (<0.1) in most cases and the outputted trajectories look the same across the modes. The modes are collapsing and learn to output the same behaviour.

## 5.2 Prediction Accuracy of the classical approaches

Before comparing all approaches against each other, the classical methods are also analysed individually since they do not need learning and can therefore be directly applied onto the real datasets Crowdbot and JRDB. Each method has hyperparameters that are fitted to the specific train split of the dataset and the ADE and FDE can be applied afterwards on the respective test dataset.
Firstly, for the Crowdbot dataset, the Kalman Filter with a constant velocity model is applied with a process noise variance $q$ of 0.1, a measurement noise variance $r$ of 0.1, the initial state covariance $P$ of 1.0 and the time step $\Delta t$ 0.9. This results in an ADE of 0.34 meters and an

| | Kalman Filter | Interacting Multiple Model | Particle Filter |
|---|---|---|---|
| Crowdbot | 0.33\|0.78 | 0.36\|0.83 | 0.34\|0.82 |

**Table 5.1:** The ADE and FDE (separated by a |) on average over 1000 samples of the crowdbot dataset for the different methods.

FDE of 0.8 meters on average for the test split of the Crowdbot dataset. In Figure 5.7, one can see the average distances of trajectories made in six seconds (15 time steps) and therefore the distribution of pedestrian behaviour to classify the calculated metrics. With 75 percent of the trajectories being in the range up to three meters, the Crowdbot dataset captures humans in close proximity to the robot.
The Interacting Multiple Model approach reaches an ADE of 0.36 meters and a FDE of 0.83 meters with a process noise variance $q$ of 0.1, a measurement noise variance $r$ of 0.1, the initial state covariance $P$ of 1.0, the omega variance 1.0 and the time step $\Delta t$ 0.4 . Compared to that,

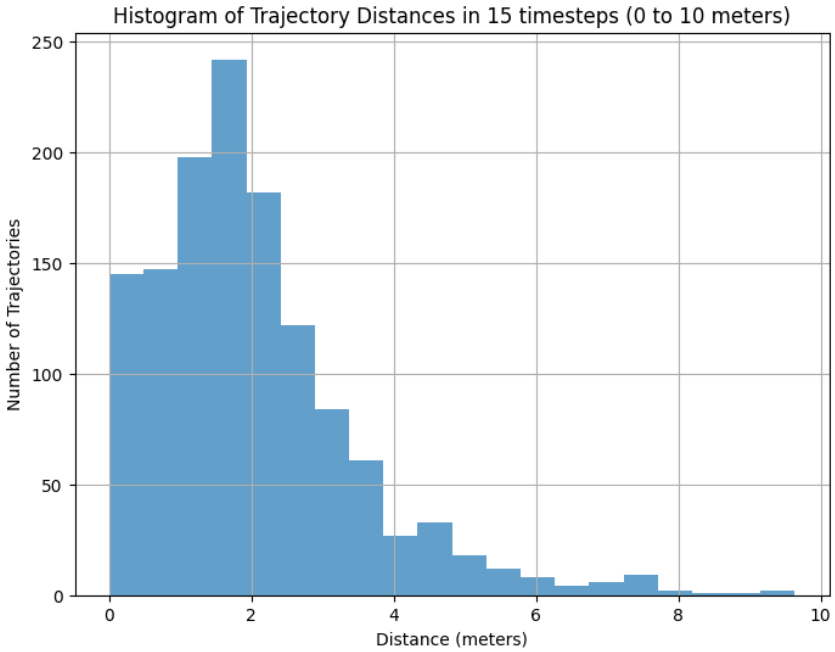


**Figure 5.7:** A histogram of trajectory distances (calculated by the euclidean norm between the first and last position of each trajectory) of all training samples of the Crowdbot dataset. Every trajectory is six seconds long, each bin represents a range of 50 centimeters.

the Particle Filter approach reaches with a process noise variance $q$ of 0.2, a measurement noise variance $r$ of 1.0, 1.000 particles and the time step $dt$ 1.0 an ADE of 0.34 meters and a FDE of 0.82 meters.

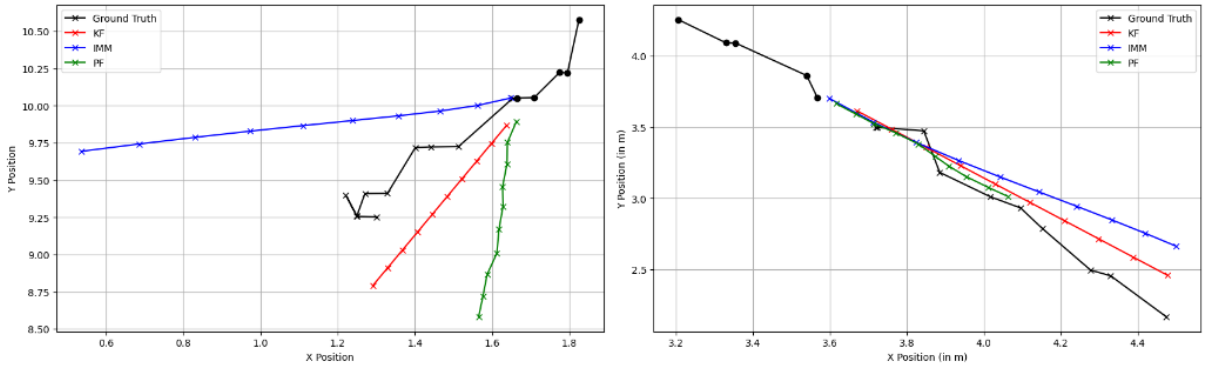


**Figure 5.8:** Two examples of the Crowdbot test dataset where the black trajectory refers to the ground truth (the black points are the first five steps as known history information and the black crosses the ten steps to be predicted). The blue trajectories are predicted from the first five ground truth inputs and are connected by their sequential order with the IMM model, the red ones with the KF model and the green one with the Particle Filter.

Especially in Figure 5.8, one can see that the KF method predicts a trajectory closest to the ground truth, that mirrors the ADE and FDE values. Moreover, the two trajectories of the Crowdbot dataset were chosen because they represent the intended use case best with an overall straight trajectory in an open space from real measurements. More examples can bee seen in the appendix 2.
Similarly to before, one can analyse the influence of different history lengths in order to compare short and long-term predictions. From the figure 5.9 one can see that all methods improve in accuracy with more historical information on a total of 15 time steps. In percentage terms, one can say that the KF improves its predicition accuracy by 47.12 % when having three more steps as history information, while the IMM improves around 49.37 % and the PF only 28.93%.

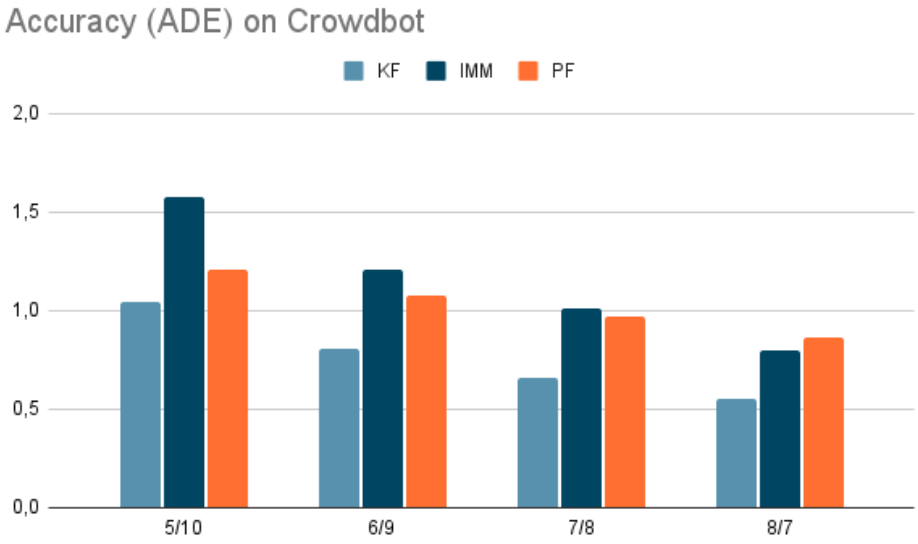


**Figure 5.9:** The prediction accuracy in terms of ADE in meters on average on the test split of the Crowdbot dataset. The methods are indicated as orange for the Particle Filter, dark blue for the Interacting Multiple Models approach and light blue for the Kalman Filter each on the Crowdbot dataset but with different length of history and prediction steps. 5/10 stands for five history steps and ten prediction steps.

## 5.3 Comparison of all Methods

Due to the fact that the HST only learns sufficiently on the synthetic dataset, one can compare all implemented methods against each other only on this dataset. While the distribution of trajectory lengths is already described before (see Figure 5.7, as well as the prediction accuracy in terms of ADE and FDE being 0.59 and 0.91 meters respectively, a comparison to the classical approaches is missing so far. After applying the hyperparameter tuning onto the training split of the synthetic dataset, the KF with a constant velocity model is applied with a process noise variance $q$ of 0.1, a measurement noise variance $r$ of 0.1, the initial state covariance $P$ of 0.5 and the time step $\Delta t$ 0.9. This results in an ADE of 0.34 meters and an FDE of 0.82 meters

| | HST | Kalman Filter | Interacting Multiple Model | Particle Filter |
|---|---|---|---|---|
| Synthetic | 0.59\|0.91 | 0.34\|0.82 | 0.66\|1.67 | 0.76\|1.97 |

**Table 5.2:** The ADE and FDE (separated by a |) on average over 1000 samples of the synthetic dataset for the different methods.

on average for the test split of the synthetic dataset (see table 5.3). With almost the same hyperparameter values and additionally the omega variance of 1.0, the IMM approach yields an ADE of 0.66 meters and a FDE of 1.67 meters (see table 5.3). Lastly, for the PF approach with a constant velocity model for prediction steps, the best results were achieved with a time step of 0.1, a measurement noise of 0.5, the process noise of 0.2 and 5.000 particles in total. This leads to an ADE of 0.76 meters and a FDE of 1.97 meters (see table 5.3).

Figure 5.10 shows that for relatively straight movements the HST and KF produce the best

| | HST | Kalman Filter | Interacting Multiple Model | Particle Filter |
|---|---|---|---|---|
| Synthetic | 0.59\|0.91 | **0.52\|0.82** | 0.93\|1.58 | 1.22\|1.97 |
| JRDB | 2.77\|2.81 | **0.10\|0.26** | 0.12\|0.33 | 0.12\|0.30 |
| Crowdbot | - | **0.33\|0.78** | 0.36\|0.83 | 0.34\|0.82 |

**Table 5.3:** The ADE and FDE (separated by a |) on average over 1000 samples for the synthetic, JRDB and Crowdbot dataset for the different methods. The bold values indicate the best (minimum) value for that dataset.

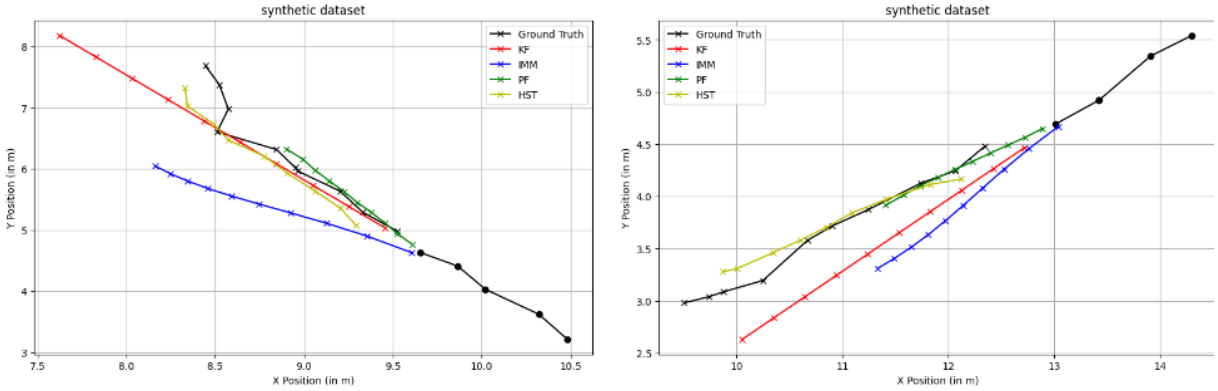


**Figure 5.10:** Two examples of the synthetic test dataset where the black trajectory refers to the ground truth (the black dots are the first five steps as known history information and the black crosses the ten steps to be predicted). All trajectories are predicted from the first five ground truth inputs and are connected by their sequential order, the predictions are visualised with the KF method (red), IMM method (blue), PF method (green) and HST (yellow).

fitting trajectories (more examples are in appendix 1). A total overview of all used datasets on all applicable methods is given in the following table and highlights that in terms of ADE and FDE, the KF achieves the best results on all datasets, though the HST only performs worse by

a very small amount (see table 5.3)

## 5.4 Computational Efficiency

Like described in section 4.2.2 the computational efficiency is measured as average runtime in seconds for one trajectory. Figure 5.11 shows that the slowest methods are the IMM with 0.1462 seconds and the HST with 0.1313 seconds on average. The fastest method is the KF which needs 0.0398 seconds on average followed by the PF with 0.0739 seconds on average.

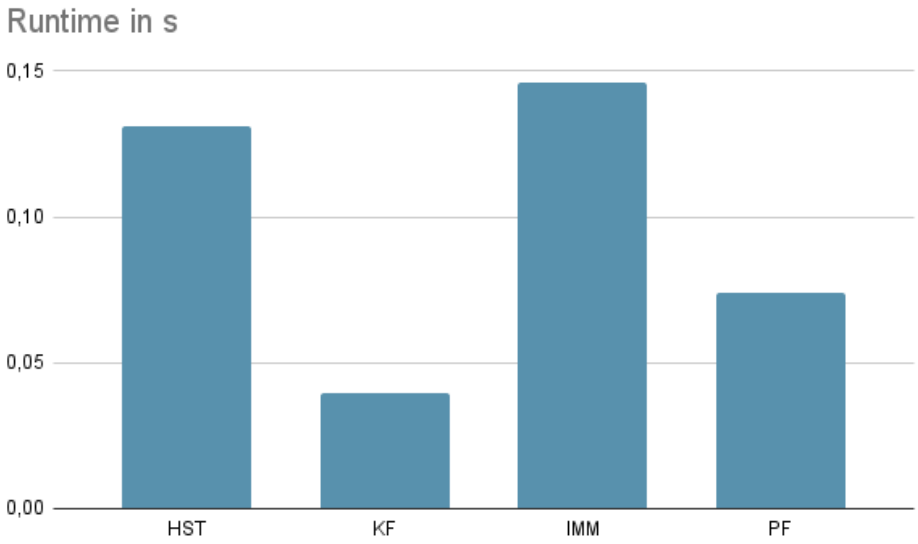


**Figure 5.11:** The runtime is presented as seconds needed to predict the next ten steps from the given five steps. The runtime is given for the HST, KF, IMM and PF method by averaging 100 sample runtimes of the synthetic dataset.

# Chapter 6

# Discussion

The following chapter discusses the aspects found in the analysis of the different implemented methods. Firstly, due to the intensive analysis and work done for the implemented single-agent HST, the results are analysed on their own and afterwards are compared to the results of the classical methods. Then, the results are interpreted, especially in regards to the open space scenario and advantages and limitations of each method are highlighted. Lastly, the insights for future research are mentioned that arise from the analysis and discussion.

## 6.1 Interpretation of HST Analysis

The implementation of the adapted single-agent Human Scene Transformer leads to a lot of interesting insights. The most obvious and huge influence on this whole thesis was that there is a lack of fitting data. Or to be more detailed, despite the fact that there were no datasets fitting exactly the intended use case, there is not enough data available where there are corresponding 2D and 3D features for pedestrian trajectory prediction. This was highlighted by the fact that 192.000 trajectories of the JRDB dataset were not enough to get sufficient predictions from the HST. One reason for that could be of course that the amount of data is not enough but a more reasonable explanation is that the JRDB dataset is too diverse with too many interactions and context, that it cannot be captured by the adapted single-agent HST. The fact that this version of the HST learns on the synthetic dataset with only 33% more data than JRDB, but compared to two million data points still significantly less, shows that the simplified model is still functional and that the training procedure itself works, but but that it needs data fitting to the model.
Moreover, the analysis showed that fine-tuning did not help the HST to better capture the JRDB data. In general, having a sufficiently trained model on some dataset should help the model generalize to another domain or other data since the model does not have to start with random weights but already learned some important aspects (probably linear movements). Due to the fact that the model was not able to generalize from the synthetic dataset to the JRDB dataset shows that the JRDB dataset captures way more complex behaviour that requires insight on neighboring persons to get an accurate prediction. This insight leads to the idea that one does not need two million trajectories like the original HST implementation to train the adapted

single-agent version, but with less complex data like the synthetic set has, fewer training samples lead to sufficient predictions. Therefore it does not need overall diversity like offered from JRDB, but diversity in the range of the intended use case.
Moreover, the idea of capturing different behaviour patterns by learning different modes with the HST showed not to be applicable to the single-agent version. One reason for that mode collapse, where there is no difference between the mode's output, could be that the synthetic dataset does not represent human behaviour realistic enough and due to its simplicity there are no patterns identifiable. This could either be the fact for the intended use case in general or is an artifact of the synthetic dataset on its own. To prove that modes are not beneficial for the open space scenario one would have to collect real data to see if there are different behavioural patterns.
Due to the fact that the synthetic dataset only consists of positional data, the influence of the skeletal keypoints could only be analyzed on the JRDB dataset, which does not represent a learned HST model. From that analysis one could infer that in the special case of having diverse and interacting pedestrians but only considering single trajectories, the skeletal keypoints are not helping the single-agent HST in predicting the future trajectory. This is an interesting artifact because from a naive view I would say that if only single agents are looked at, interactions may be foreseen by their body movements or at least give hints for direction changes and therefore I would assume at least a comparable accuracy to the model without additional information. Since this is not the case, the additional information may cause the model to learn slower. Then, the feature is not an impediment by itself but the model has more data to learn where it has to attend to and what is important for the future trajectory. Again, real data with skeletal keypoint information is needed to really investigate the impact of pose information but the original HST paper [21] showed that for contextual scenes and interactions, pose information is an advantage.

Regarding the hypothesis that the HST as a state-of-the-art deep learning approach is at least better in short-time prediction, the insights from different history lengths as input are interesting. Though it will be compared to classical approaches in a later part of this discussion, the analysis of different history lengths showed so far that with more information the prediction over all used datasets gets better but for the only learned case (synthetic dataset) only by a small amount (ADE improves by 10.17% and FDE by 17.58%). Interestingly, one can see that while more history information increases the prediction accuracy on average only by ten percent in the case where the trajectory prediction is really learned but for the JRDB dataset the ADE increases by 40%. This may indicate that for such complex trajectories with interactions, history information is more relevant than additional pose information. Again, one needs real data fitting to the intended use case to analyse it but from the analysis so far it would be interesting how the variables history length and number of features are correlating since they are influencing the training process as well as end prediction accuracy differently.
Moreover, due to the limitation of all sequences being cropped to 15 time steps (six seconds), the analysis of long-term prediction is difficult. The limitation comes from the training process of the HST that trains one model for one fixed history length and for a fixed number of prediction steps. Therefore all sequences are six seconds long and the focus was laid on the influence of the history length, since a long-term prediction is not possible in six seconds. A

possibility would be to train different HST models on different lengths of trajectories but with enough data I would assume that different lengths in one HST model could be possible due to the masked sequence learning where one could pad or mask out time steps that are not available.

To sum up, the single-agent HST that I implemented is able to learn simple trajectory distributions, like designed with the synthetic dataset, and for further insights needs real world data with less complexity than the JRDB dataset. From the given data and analysis, one can conclude that more available history information leads to a more accurate prediction and the skeletal keypoints and different modes are so far no advantage but due to the mentioned limitations should not in general be excluded.

## 6.2 Interpretation of Comparison

The main insight from the comparison of all implemented methods is that the Kalman Filter with a constant velocity model outperforms the other methods on all regarded datasets. Moreover, this is independent from the amount of history steps and prediction steps. The KF approach performs better in terms of average and final displacement error for five and eight history steps with respectively ten and seven steps to predict. Nevertheless, one has to mention that the HST performs only slightly worse on the synthetic dataset for short time prediction. Though, the difference between both methods increases with more historical information where the HST performs 13.46% worse (0.52 meters ADE on KF and 0.59 meters ADE on HST) than the KF for five history steps and 76.67% worse for eight history steps (0.30 meters ADE on KF and 0.53 meters ADE on HST).
From that, one can conclude that the simplicity assumptions made for the intended use case that are mirrored in the synthetic dataset fit to the linear motion assumption of the KF and that on average the trajectories can be described by a simple constant velocity model. This is not that surprising since the synthetic dataset was generated by the idea of overall linear motion and therefore the only difficulty for the KF was to capture the generated noise velocity and direction. But even though the KF achieves the best results, the comparison showed that the HST is in its adapted version able to capture such behaviour too but only to a certain degree and that more historical information is not that beneficial. This suggests that the HST may struggle with long term dependencies, which is surprising, since the main advantage of attention mechanisms over other methods is that they can capture these long-term dependencies. One reason for that could be that the attention mechanisms are not always focusing on the time domain but on the features (or in the original paper on interactions [21]).
Therefore it would be interesting to train the single-agent HST on real world data to see if the model is overall still too complex for the data and overfits if it gets too much historical information. Again, it would be interesting how real world data for the intended use case is distributed, how close it is to the synthetic dataset and if the simplistic CV model for the KF still outperforms the HST.
The fact that the chosen classical models are all limited by strong linear assumptions makes it unsurprising that their performance is strongly dependent on the simplicity of the human behaviour captured by the data. Similarly to the single-agent HST, the models are not designed

to capture complex behaviour like interactions, context-dependent behaviour or spontaneous changes. That makes the analysis of the methods over real-world data interesting, because Crowdbot and JRDB contain plenty of such artifacts. The fact that the KF even outperforms the IMM and PF in such cases is therefore fascinating because the IMM is designed to handle behaviour changes but probably due to not knowing when this happens, this was not properly modelled and therefore not captured. Moreover, looking at sample trajectory predictions of the IMM (see Figure 2 in the appendix), it seems that the implementation of three different turn models leads the IMM to often choose a turn model, even if the underlying trajectory is more linear. Here it would be interesting if a coordinated turn model, that is then implemented with an Extended Kalman Filter to capture the non-linearity and adapt the turn rate to the data, would fit better to the trajectories.
Another interesting artifact found, especially if looking at sample trajectory predictions (see Figure 1 in the appendix), is that the PF, which in theory should outperform the KF due to its usage of the same underlying model but the PF being better at capturing non-linear noise, is performing worse than the KF. This happens probably because I implemented the CV model for the PF to predict from the first time step on, so the particles are moved according to the model in the first history steps instead of using only the measurements. Therefore the particles are not behaving like the pedestrian in the first steps but only like the assumed model and cannot capture the real behaviour sufficiently. Though it is not possible to test an improved version of the PF in the scope of this master thesis, one can assume that with a correct implementation where the CV model is adapted to the data during the first steps and the PF updates the particles only on the data and not the model, the PF may outperform the KF.

One aspect that could not be shown due to missing real world data is that the HST model is designed to capture occlusions. This is intentionally designed by the masked sequence learning to handle interactions or obstacles where pedestrians are occluded but would be interesting for the intended use case in case that pedestrians are suddenly appearing very close to the robot where parts of the body are then also occluded. Right now, with the available data one cannot clearly say that the HST outperforms the KF in such cases because of a lack of available data but I would assume that it may be beneficial and even if it is only such specific cases, the HST may be better. But nevertheless, the analysis so far showed that on average the KF outperforms the HST, the IMM and the PF with their specific assumptions and limitations, showing that for the simple intended use case, the chosen neural network is not an advantage and despite the higher complexity of the model and the time to learn the model, the KF outperforms also in terms of computational efficiency.
Therefore the proposed research question if a state-of-the-art neural network is for such a simplification still a large benefit over classical non-deep learning methods can be answered with no. The answer is dependent on the current available data and the simplifications and assumptions made by the intended use case. Moreover, one has to highlight that the neural network is no large benefit but also not much worse, at least for short-time predictions. The hypothesis that the deep learning approach is at least better in short time prediction and predicts human behaviour more accurate in general cannot be neglected in total. Its accuracy is comparable to the KF and from an intuitive view most of the predicted trajectories of the HST look more like realistic trajectories due to the non-linearity but on average are as far away as the predictions from the linear KF. Still, one has to mention that the HST like other neural networks sometimes

completely fails to capture the data distribution, especially if the given trajectory is more an outlier than the mean.

## 6.3 Implications for Open Space Scenarios

The intended open space scenario was simplified to single agents operating in a nearly obstacle- and context-free area. The correlating pedestrian trajectories are therefore assumed to be mostly straight or slightly curved, representing human behaviour that is mainly goal-driven and when there are no obstacles or context, humans typically do not show more complex behavioural patterns. With the synthetic dataset designed based on that assumption, it was shown that state-of-the-art deep learning methods like the HST can be adapted towards that use case and are not too complex to capture such simple behaviour. The single-agent HST achieves comparable prediction accuracies to the classical KF approach but due to not learning different behaviour patterns, one can assume that the use case does not include that and models do not need the complexity to handle different behaviour patterns.
Nevertheless, from all analysed aspects, the logical method for predicting pedestrian trajectories in the given use case would be the KF. The KF outperforms the HST on short-term and even more on long-term predictions, is easier and faster to implement, needs only positional data (therefore only 3D data) and is also faster in prediction. This makes it more real-time applicable than the HST that needs more training and pre-processing of data to extract corresponding skeletal keypoints to the positions. Moreover, taking new measurements in real-time into account may be easier with the KF than the HST because so far the HST is trained on a fixed sequence length and therefore neglects prior information if the sequence gets longer.

Though all methods are designed to capture simple pedestrian behaviour, there may occur some special cases that one has to take into account. Despite realistic human behaviour like not directly walking straight or slightly changing velocities, it may happen in the designed use case that the pedestrian may occur suddenly in the field of view from the moving robot. Probably in most cases pedestrians are a few meters away and occurred from the side but in some cases a pedestrian may occur in close proximity to the robot. This may cause problems because the robot needs to rapidly predict what the pedestrian is going to do in order to not collide with the pedestrian and the pedestrian may be occluded if the given sensors cannot capture it in time. From the previous analysis, one would have to choose the KF as best method for the intended use case but though it cannot be tested with the given data, based on the simple model and overall KF algorithm one could assume that such cases may cause a problem. Like it was shown before, the KF predictions get better with more information.
These cases may be rare and the hypotheses that they occur has to be validated with real-world data but one advantage of the original HST is to consider those occlusions and handle such short-time predictions. One idea could be to use the KF in general since it is shown to be the best fitting method, to get a first, fast and sufficiently good approximation of where the pedestrian is going and for the described rare cases of occlusions explicitly train the HST model on it to handle that. Of course this is only necessary if the occlusion cases are really important or dangerous, but in these cases one could consider a lot of different more complex models that are outside the scope of this thesis.

With the given use case and idea to help a moving robot with its path planning, the KF filter appears to be the best solution.

## 6.4 Advantages and Limitations of each approach

Though rarely discussed before, each method has shown advantages and limitations for the given use case. The following aspects are all dependent on the given analysis and highly dependent on the data, which is discussed before.
First of all, the Kalman Filter's advantages are its good performance in prediction accuracy as well as computational cost that makes it real-time applicable. Moreover it performs continuously well, independent of the number of history steps and is able to include new measurements in real-time. On average, the KF can capture most of the simple pedestrian trajectories sufficiently. The method is limited by its strong linear motion assumption and constant velocity assumption that makes it unsuitable for capturing complex human behaviour like non-linear movements or sudden changes. By taking only positional information of one pedestrian it therefore cannot capture context like interactions or environmental changes.
One advantage of the (adapted) single-agent Human Scene Transformer are that with very little historical information it achieves comparable, sufficiently good predictions as the Kalman Filter. Moreover, though it cannot be proven with the given data, but theoretically the HST uses attention mechanisms to model long-term dependencies and therefore should be also able to predict longer trajectories. It uses masked sequence learning to capture occlusions and missing data. Nevertheless, the HST is limited by the data it is trained on. The analysis showed that the data has to fit to the model, and that for example the JRDB dataset has too complex human behaviour that cannot be captured. Moreover, the HST training takes a lot of time and computational cost and the the whole implementation complexity is way higher than for a classical approach like the KF.
The Interacting Multiple Model approach that was implemented in this thesis theoretically handles multiple motion models and therefore multiple behaviour pattern. This should in general be an advantage but like shown with the mode collapse of the HST, it is at least not an advantage for the synthetic dataset. Therefore, it is suited for more complex trajectories but needs more adaption and fine-tuning of the models and parameters to fit to the data. In the way the IMM is implemented here, it neither outperformed nor achieved comparable prediction accuracies.
The implemented Particle Filter was shown to be not perfectly fitting but in general it is capable to handle non-linearity and therefore noisy as well as complex pedestrian trajectories. Its advantage and usefulness has therefore to be validated with real-data and its computational cost is also high in the given implementation due to many particles being moved in one step, but there are possibilities to improve this by parallelization.

## 6.5 Insights for Future Research

The main contribution that future researches can do is to collect more data, especially fitting to an open space scenario in order to validate all the points made before. The main problem with that is that large benchmarks try to capture diverse behaviour and until now are mainly

focusing on complex human interaction behaviour or pedestrian behaviour in traffic scenarios. Other datasets have mainly a specific use case in mind and since there was no interest so far in the given simple open space scenarios, there is a lack of data. Future research should therefore take this scenario into account and collect data or at least a dataset of correlating 3D and 2D data to train the HST with position and skeletal keypoints. Moreover, the takeaway from this thesis should be that the latest, complex neural networks like transformers are not always necessary and especially for simpler tasks like the intended use case here, one should always consider easier models too. Though here the easier models were non-deep learning methods, there are easier sequential deep learning methods like LSTMs out there. Or the other way around, there are more complex models for the KF than the constant velocity model. Therefore, future research should not always consider the latest methods or complex methods but think about how well the output should be and if simpler models may be sufficient.
Other interesting aspects future research could take into account are for example how applicable trained multi-agent neural networks like the HST are on single-agent trajectories. The idea could be derived from the fact that the single-agent HST is not able to capture complex behaviour from the JRDB dataset, but maybe the multi-agent HST would do well on predicting interactions but worse on single agents without interactions. Moreover, the influence of the skeletal keypoints could not be fully analysed in this thesis and an interesting question for further research. Maybe the pose information is only beneficial in the interaction cases and other features like the head direction are more correlated to simple human motion. Depending on the task, the question of long-term prediction is also interesting for future research since the analysis of this thesis showed that the KF can better use more historical information than the single-agent HST. It is questionable how longer prediction steps from the same amount of history steps look like and if with real-world data the HST may better use its attention mechanisms to capture long-term dependencies. Moreover, future research could investigate the way of learning and if the fixed-sized sequences can be made of variable length to be applicable to different length of trajectories without retraining. Further, it would be beneficial for more deep learning methods if the overall computational costs can be minimized. Since the HST is mainly based on transformers, research in this direction would be applicable to many state-of-the-art transformers.
Lastly, the implementations of the Interacting Multiple Model approach and the Particle Filter were not perfectly fitting the given dataset and a deeper investigation of these methods was not possible due to the master thesis being limited but could also be further investigated that if a better implementation or parameter tuning would make the methods comparable or even better than the KF.

# Chapter 7

# Conclusion

The given thesis showed that in simple scenarios like the described open space scenario, classical methods, like the Kalman Filter with a simple constant velocity model can outperform complex state-of-the-art deep learning models like Human Scene Transformer in a single-agent trajectory prediction task. While one main point is that the whole thesis handles the lack of missing data for such a use case, the thesis highlights the use of synthetic data to get an approximation of the methods without real world data. Further, the analysis showed that the classical methods KF outperforms the HST in accuracy, effort and efficiency what makes it real-time applicable and best fitting for the use case.
Moreover, this thesis shows that human behavior is more complex than simple linear motion, making it clear that we can't fully capture or generalize it with basic models. That is one of the reasons why the HST did not learn on the JRDB dataset. There are complex methods trying to capture most of this diverse behaviour but for the simplistic assumed motion, other methods and features need to be taken into account. The skeletal keypoints were no advantage with the adapted model and the given data, bit cannot be excluded as a benefit in general as well as it is questionable if other features like the head direction are more related to simple pedestrian motions.
Further, the question of the different methods performance in long-term as well as occluded scenarios needs to be further investigated and future research should therefore focus on creating fitting datasets to explore that sufficiently.

# Bibliography


[1] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282, 1995.

[2] Haziq Razali, Taylor Mordan, and Alexandre Alahi. Pedestrian intention prediction: A convolutional bottom-up multi-task approach. *Transportation research part C: emerging technologies*, 130:103259, 2021.

[3] Stefan Becker, Ronny Hug, Wolfgang Hübner, and Michael Arens. An evaluation of trajectory prediction approaches and notes on the trajnet benchmark. *arXiv preprint arXiv:1805.07663*, 2018.

[4] Mahsa Golchoubian, Moojan Ghafurian, Kerstin Dautenhahn, and Nasser Lashgarian Azad. Pedestrian trajectory prediction in pedestrian-vehicle mixed environments: A systematic review. *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[5] Raphael Korbmacher and Antoine Tordeux. Review of pedestrian trajectory prediction methods: Comparing deep learning and knowledge-based approaches. *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[6] Mark Nicholas Finean, Luka PetroviÄ, Wolfgang Merkt, Ivan MarkoviÄ, and Ioannis Havoutis. Motion planning in dynamic environments using context-aware human trajectory prediction. *Robotics and Autonomous Systems*, 166:104450, 2023.

[7] Jiquan Ngiam, Benjamin Caine, Vijay Vasudevan, Zhengdong Zhang, Hao-Tien Lewis Chiang, Jeffrey Ling, Rebecca Roelofs, Alex Bewley, Chenxi Liu, Ashish Venugopal, et al. Scene transformer: A unified architecture for predicting multiple agent trajectories. *arXiv preprint arXiv:2106.08417*, 2021.

[8] Jiangbei Yue, Dinesh Manocha, and He Wang. Human trajectory prediction via neural social physics. In *European Conference on Computer Vision*, pages 376–394. Springer, 2022.

[9] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 961–971, 2016.

[10] Annalena Bebenroth. Pedestrian trajectory prediction, 2024. `https://github.com/anbebe/Pedestrian-Trajectory-Prediction`.

[11] Chargepal, 2024. `https://chargepal.de/` [Accessed: (25.01.2024)].

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[13] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

[14] Roger Labbe. Kalman and bayesian filters in python. *Chap*, 7(246):4, 2014.

[15] Nicolas Schneider and Dariu M Gavrila. Pedestrian path prediction with recursive bayesian filters: A comparative study. In *german conference on pattern recognition*, pages 174–183. Springer, 2013.

[16] X Rong Li and Vesselin P Jilkov. Survey of maneuvering target tracking. part i. dynamic models. *IEEE Transactions on aerospace and electronic systems*, 39(4):1333–1364, 2003.

[17] Xianghui Yuan, Feng Lian, and Chongzhao Han. Models and algorithms for tracking target with coordinated turn motion. *Mathematical Problems in Engineering*, 2014(1):649276, 2014.

[18] Hao Wu, Wenjun Xu, Bitao Yao, Yang Hu, and Hao Feng. Interacting multiple model-based adaptive trajectory prediction for anticipative human following of mobile industrial robot. *Procedia Computer Science*, 176:3692–3701, 2020.

[19] Roberto Conde, A Ollero, and JA Cobano. Method based on a particle filter for uav trajectory prediction under uncertainties. In *40th International Symposium of Robotics. Barcelona, Spain*. Citeseer, 2009.

[20] Pek Hui Foo and Gee Wah Ng. Combining imm method with particle filters for 3d maneuvering target tracking. In *2007 10th International Conference on Information Fusion*, pages 1–8. IEEE, 2007.

[21] Tim Salzmann, Hao-Tien Lewis Chiang, Markus Ryll, Dorsa Sadigh, Carolina Parada, and Alex Bewley. Robots That Can See: Leveraging Human Pose for Trajectory Prediction. *IEEE Robotics and Automation Letters*, 2023.

[22] Tim Salzmann, Boris Ivanovic, Punarjay Chakravarty, and Marco Pavone. Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVIII 16*, pages 683–700. Springer, 2020.

[23] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. Social gan: Socially acceptable trajectories with generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2255–2264, 2018.

[24] Cunjun Yu, Xiao Ma, Jiawei Ren, Haiyu Zhao, and Shuai Yi. Spatio-temporal graph transformer networks for pedestrian trajectory prediction. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XII 16*, pages 507–523. Springer, 2020.

[25] Stanford Vision and Learning Laboratory JackRabbot Team. Sensor setup in jrdb, the jackrabbot dataset and benchmark. `https://download.cs.stanford.edu/downloads/jrdb/Sensor_setup_JRDB.pdf`.

[26] Roberto Martin-Martin, Mihir Patel, Hamid Rezatofighi, Abhijeet Shenoi, JunYoung Gwak, Eric Frankel, Amir Sadeghian, and Silvio Savarese. Jrdb: A dataset and benchmark of egocentric robot visual perception of humans in built environments. *IEEE transactions on pattern analysis and machine intelligence*, 2021.

[27] Diego Paez-Granados, Yujie He, David Gonon, Lukas Huber, and Aude Billard. 3d point cloud and rgbd of pedestrians in robot crowd navigation: Detection and tracking. *IEEE DataPort*, 12, 2021.

[28] Diego F Paez-Granados, Hideki Kadone, Modar Hassan, Yang Chen, and Kenji Suzuki. Personal mobility with synchronous trunk–knee passive exoskeleton: Optimizing human–robot energy transfer. *IEEE/ASME Transactions on Mechatronics*, 27(5):3613–3623, 2022.

[29] Dan Jia, Mats Steinweg, Alexander Hermans, and Bastian Leibe. Self-supervised person detection in 2d range data using a calibrated camera. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13301–13307. IEEE, 2021.

[30] Xinshuo Weng, Jianren Wang, David Held, and Kris Kitani. 3d multi-object tracking: A baseline and new evaluation metrics. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10359–10366. IEEE, 2020.

[31] Alex Bewley, Peter Hawkins, and Tim Salzmann. Human scene transformer, 2023. `https://github.com/google-research/human-scene-transformer`.

[32] Saeed Saadatnejad, Yang Gao, Hamid Rezatofighi, and Alexandre Alahi. Jrdb-traj: A dataset and benchmark for trajectory forecasting in crowds. *arXiv preprint arXiv:2311.02736*, 2023.

[33] wmuron and Maciej Budys. motpy - simple multi object tracking library, 2021. `https://github.com/wmuron/motpy`.

[34] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3645–3649. IEEE, 2017.

[35] Nir Aharon, Roy Orfaig, and Ben-Zion Bobrovsky. Bot-sort: Robust associations multi-pedestrian tracking. *arXiv preprint arXiv:2206.14651*, 2022.

[36] Qi Chu, Wanli Ouyang, Hongsheng Li, Xiaogang Wang, Bin Liu, and Nenghai Yu. Online multi-object tracking using cnn-based single object tracker with spatial-temporal attention mechanism. In *Proceedings of the IEEE international conference on computer vision*, pages 4836–4845, 2017.

[37] V Bazarevsky. Blazepose: On-device real-time body pose tracking. *arXiv preprint arXiv:2006.10204*, 2020.

[38] JRDB. Jrdb-pose 2022, 2023. [Online; accessed September 20, 2024].

[39] Rishabh Bajpai and Deepak Joshi. Movenet: A deep neural network for joint profile prediction across variable walking speeds and slopes. *IEEE Transactions on Instrumentation and Measurement*, 70:1–11, 2021.

[40] Ronny Votel and Na Li. Next-generation pose detection with movenet and tensorflow.js, 2021. `https://blog.tensorflow.org/2021/05/next-generation-pose-detection-with-movenet-and-tensorflowjs.html`.

[41] Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. Training deep networks with synthetic data: Bridging the reality gap by domain randomization. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 969–977, 2018.

[42] Matteo Fabbri, Guillem Brasó, Gianluca Maugeri, Orcun Cetintas, Riccardo Gasparini, Aljoša Ošep, Simone Calderara, Laura Leal-Taixé, and Rita Cucchiara. Motsynth: How can synthetic data help pedestrian detection and tracking? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10849–10859, 2021.

[43] Muhammad Naveed Riaz, Maciej Wielgosz, Abel García Romera, and Antonio M López. Synthetic data generation framework, dataset, and efficient deep model for pedestrian intention prediction. In *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*, pages 2742–2749. IEEE, 2023.

[44] Douglas Reynolds. *Gaussian Mixture Models*, pages 659–663. Springer US, Boston, MA, 2009.

[45] Roger Labbe and 33 more contributors. Filterpy - kalman filters and other optimal and non-optimal estimation filters in python., 2015. `https://github.com/rlabbe/filterpy`.

[46] Seham Mouawad Aly, Raafat El Fouly, and Hoda Braka. Extended kalman filtering and interacting multiple model for tracking maneuvering targets in sensor netwotrks. In *2009 Seventh Workshop on Intelligent solutions in Embedded Systems*, pages 149–156. IEEE, 2009.

[47] Fahime Farahi and Hadi Sadoghi Yazdi. Probabilistic kalman filter for moving object tracking. *Signal Processing: Image Communication*, 82:115751, 2020.

[48] Vaibhav Malviya and Rahul Kala. Trajectory prediction and tracking using a multi-behaviour social particle filter. *Applied Intelligence*, 52(7):7158–7200, 2022.

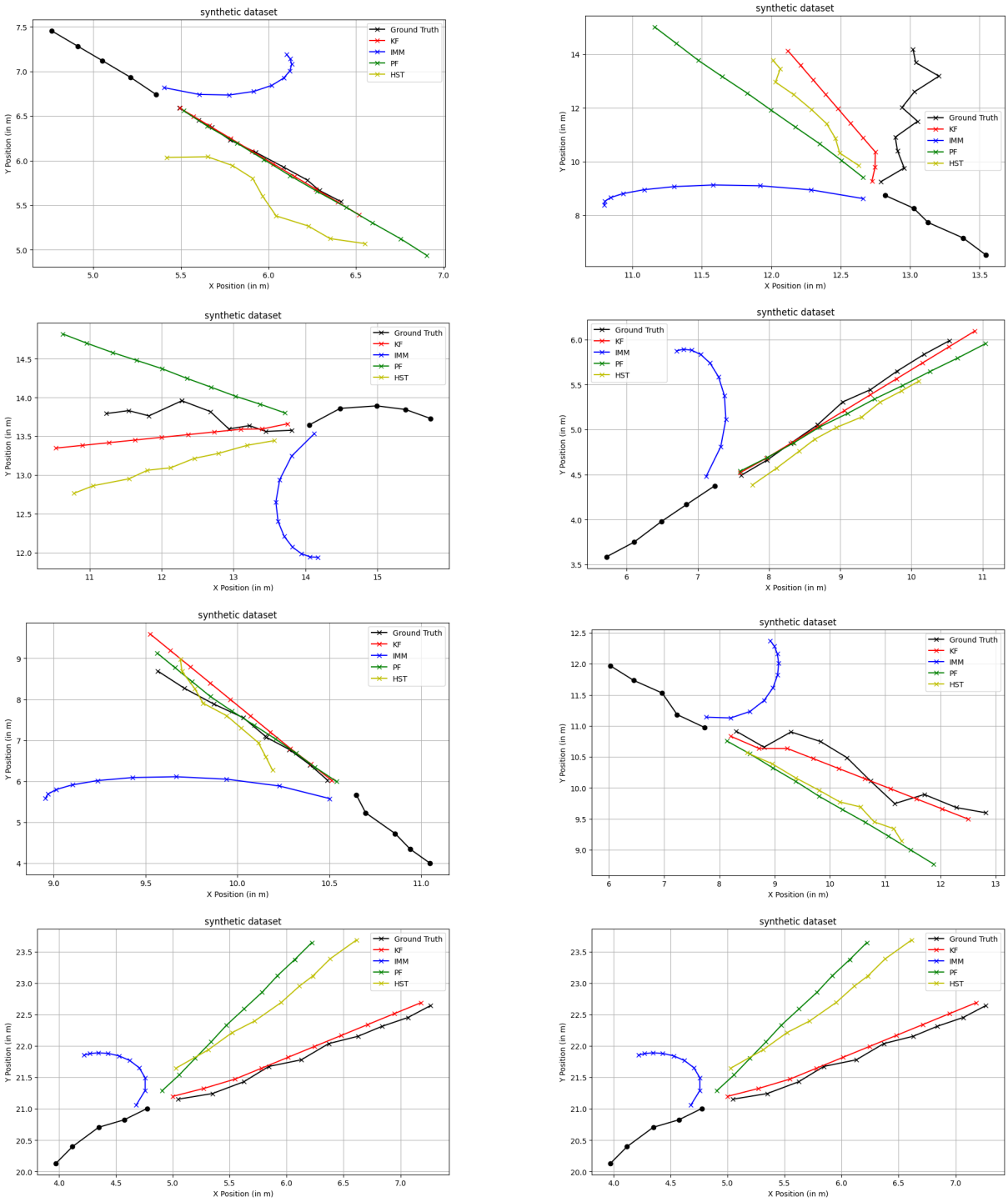


**Figure 1:** Eight example predictions of the synthetic test dataset where the black trajectory refers to the ground truth (the black points are the first five steps as known history information and the black crosses the ten steps to be predicted). The blue trajectories are predicted from the first five ground truth inputs and are connected by their sequential order with the IMM model, the red ones with the KF model, the green one with the Particle Filter and the yellow one with the single-agent Human Scene Transformer.

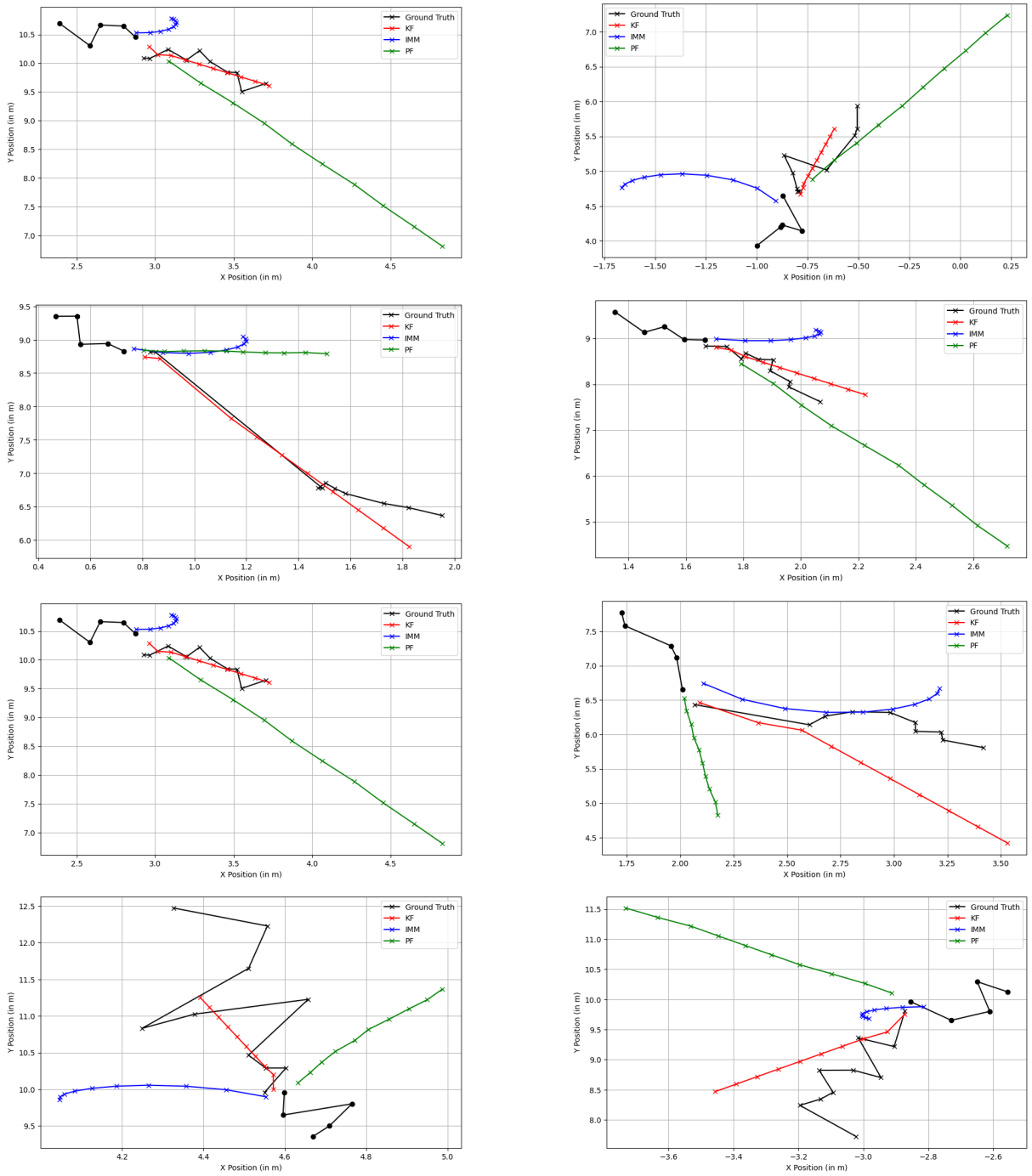


**Figure 2:** Eight example predictions of the Crowdbot test dataset where the black trajectory refers to the ground truth (the black points are the first five steps as known history information and the black crosses the ten steps to be predicted). The blue trajectories are predicted from the first five ground truth inputs and are connected by their sequential order with the IMM model, the red ones with the KF model and the green one with the Particle Filter.

# Declaration of Authorship

I hereby certify that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at this or any other university.

A. Bebenroth

signature

Osnabrück, 07.10.2024

city, date