University of Freiburg
Department of Computer Science
Autonomous Intelligent Systems Group
Prof. Dr. Wolfram Burgard

Learning Reactive Robot Navigation Policies from Predictive Trajectory Planning

Master-Thesis

Philipp Ruchti
April 1, 2013

Supervisors: Prof. Dr. Wolfram Burgard
Henrik Kretzschmar
Markus Kuderer

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich diese Master-Arbeit mit dem Titel

Learning Reactive Robot Navigation Policies from Predictive Trajectory Planning

selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet wurden und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen sind, als solche kenntlich gemacht wurden. Darüber hinaus erkläre ich, dass diese Master-Arbeit nicht, auch nicht auszugsweise, bereits anderweitig verwendet wurde.

Philipp Ruchti
Freiburg im Breisgau, den 1. April 2013

Zusammenfassung

Menschen sind es gewohnt sich in einer Umgebung mit anderen Menschen zu bewegen. Aus diesem Grund fällt es ihnen leichter den Pfad eines Roboters vorherzusagen, wenn dieser menschenähnliche Pfade fährt. In dieser Arbeit stellen wir einen reaktiven Ansatz zur Navigation eines mobilen Roboters vor. Hierbei erzeugt das hier vorgestellte Verfahren menschenähnlichen Pfade. Es gibt zwei grundlegende Ansätze sozialer Navigationsalgorithmen für mobile Roboter. Die erste Gruppe dieser Algorithmen sind die reaktiven Methoden. Diese Methoden erzeugen die Kommandos zur Steuerung eines Roboters mit Hilfe von Potenzialen. Die reaktiven Methoden zeichnen sich meistens durch geringen Rechenaufwand aus, erzeugen jedoch für gewöhnlich keine menschenähnlichen Pfade. Prediktive Methoden hingegen sind in der Lage menschenähnliche Pfade zu erzeugen, haben jedoch in der Regel einen höheren Rechenaufwand. Das Ziel dieser Arbeit ist es die Vorteile beider Verfahren zu kombinieren. Wir stellen eine reaktive Methode vor, die unter Verwendung von Locally Weighted Regression und Beispielpfaden eines prädiktiven Pfadplanungsverfahrens Beschleunigungskommandos erzeugt, um einen mobilen Roboter zu steuern. Die Verwendung einer nichtparametrischen Regressionsmethode lässt mehr Freiheiten bei der Repräsentation der Trainingsdaten, als eine parametrische Repräsentation der Daten. Um Locally Weighted Regression effizient verwenden zu können, reduzieren wir den Zustandsraum, welchen wir für die Repräsentation der Potenziale verwenden. Hierzu repräsentieren wir das Verhalten eines Roboters mit Hilfe zweier Potenziale. Das erste dieser Potenziale lenkt den Roboter in Richtung seines Ziels, während das zweite für das gegenseitige Ausweichen der Agenten verantwortlich ist. Wir reduzieren den Zustandsraum des Weiteren unter Verwendung von Symmetrien und zusätzlichen Annahmen. In den Experimenten demonstrieren wir, dass es dem Algorithmus, trotz dieser Annahmen, möglich ist die Pfade des prädiktiven Verfahrens zu reproduzieren. Unter Verwendung eines Pioneer III Roboters zeigen wir, dass die hier vorgestellte Methode in der Lage ist, einen mobilen Roboter erfolgreich durch eine Umgebung mit Fußgängern zu steuern.

Abstract

Humans are used to move in environments, where other humans are present. Due to this fact it is easier for humans to predict the path of a robot, if the robot navigates human-like. In this thesis we present a reactive navigation method to create human-like paths to navigate mobile robots. There are two basic approaches for social navigation of mobile robots. The first group are reactive approaches which create commands based on potentials. On the one hand reactive methods have low computational costs, but one the other hand they often do not create human-like paths. The second group are predictive approaches. Most of the predictive methods better reproduce human paths, but they tend to be computational more involved. In this thesis we aim to combine the benefits of both approaches. We propose a reactive method, which uses locally weighted regression and training data from a predictive path planning method to create acceleration commands to navigate a mobile robot. The use of a non-parametric regression method results in more flexibility to represent the training data compared to the use of a parametric representation. To use locally weighted regression efficiently we reduce the state space, which we use to represent the potentials. In order to do this, we first we represent the behavior, which we aim to learn, by to individual potentials, one for approaching the target and one which represents the cooperative collision avoidance between two agents. We further reduce the state space by taking advantage of symmetries and additional assumptions. In the experiments we demonstrate, that despite of these reductions, we are able to reproduce the paths generated by the predictive path planning method, which we use to create training data. In our experiments with a Pioneer III robot and multiple humans we demonstrate that this method can be used to drive a mobile robot successfully through an environment with pedestrians.

Contents

1 Introduction

In the last years robots learned to execute more and more tasks to aid humans in daily life, for example fetching drinks from the refrigerator (see Bohren et al. [9]), unloading items from a dishwasher (see Saxena et al. [38]) or folding towels (see Maitin-Shepard et al. [30]). In the future robots will be able to execute a growing number of daily tasks. If robots move in an environment with pedestrians, they need to move in a human compatible way. Humans are used to interact with other humans, as a result it is easier for humans to predict the path of a robot if the robot navigates human-like. One possibility to create a path through an area with pedestrians is to predict the paths of all other agents involved and afterward choose a path avoiding them. However, this is not the way humans move. If we move, we do not know all the targets or paths of all other humans or robots in a room. Instead we start walking and react to situations we encounter. The first of the approaches described above is a predictive approach, because it predicts whole paths. Paths created by predictive approaches often reproduce human paths quite well, but the number of agents, which reactive algorithms can simulate within reasonable time, often limits the use of such methods for navigation of mobile robots. The second approach described above is denoted as a reactive approach. Reactive methods create commands to drive a robot using potentials. Such methods are often faster to compute than predictive methods, but usually do not create human-like paths. In this thesis we propose a reactive navigation method combining both, the accuracy and human-like paths from the predictive method and the low computational costs on run-time arising from the reactive methods. In order to achieve this, we learn a reactive navigation policy using regression and demonstration paths created by a predictive path planning method. We model the potentials so that the paths, created by our method, resemble the demonstration paths. We measure the similarity of paths using features, as travel time, acceleration, velocity and distance to other agents. The method, which we use to create demonstration paths, is learned from real human motion. By reproducing these paths we create trajectories, which inherit the same human-like characteristics. In this thesis we use two kinds of potentials to create human-like paths. The first potential drives the agents towards its target. The second potential is created by a sum of contributions from repulsive potentials of other agents, which represent the cooper-

ative collision avoidance between two agents. By combining both potentials we create human-like paths towards the agent's target while evading other agents.

1.1 Contribution of this Thesis

In this thesis we present a reactive social navigation method for mobile robots. There are two basic approaches to the task of navigating mobile robots. The first approach is the reactive approach, where the commands to drive a robot are created using potentials. A well known reactive social navigation method is the social force method proposed by Helbing and Molnár. Reactive methods are fast to compute, but they often does not create human-like paths. The second approach is the predictive approach. Predictive methods generate commands to navigate a mobile robot by computing trajectories for all involved agents. These methods are capable of creating human-like paths, but most of these methods are computationally more involved. In this thesis we combine the benefits of both approaches. We present a reactive social navigation method for a mobile robot, which is learned from human-like paths generated by a predictive method. We use locally weighted regression, a non-parametric regression method, to represent potentials to reproduce the paths of a predictive path planning method. To use locally weighted regression efficiently we reduce the state space used to represent these potentials. We use two independent potentials to represent the behavior of the robot, an attractive potential to represent the target approaching behavior of the robot and a repulsive potential which represents the cooperative collision avoidance between two agents. We additionally take advantage of symmetries and make assumptions to further reduce the state spaces representing the potentials. In the experiments carried out in this thesis we demonstrate that, despite of these reductions, the proposed algorithm is able to reproduce the paths generated by the repulsive method. In our experiments with a real robot we demonstrate that we are able to navigate a Pioneer III robot successfully through an environment with multiple pedestrians.

1.2 Outline

In the next section we discuss related approaches including social path planning methods and reactive models. In Chapter 3 we discuss some basic principles that are useful to understand this thesis. Chapter 4 presents our approach to learn a reactive navigation policy, especially how we model the proposed potentials and how we combine them to

navigate a mobile robot. We also take a look on the state spaces, which we used to represent the potentials. In Chapter 5 we evaluate the learned policy and compare it to a state-of-the-art social path planning method, the social force model from Helbing and Molnár and to the predictive path planning method proposed by Kuderer et al., which we use to create training data. Chapter 6 summarizes what we have achieved in this thesis and discusses how this work can be extended.

1.3 Notation

In this thesis an agent (robot or human) is described by a two-dimensional position, a two-dimensional velocity and a two-dimensional target. For better readability we use the following notation in the remainder of this thesis:

$\alpha, \beta, \ldots$	Names for agents
$\vec{p}^{\alpha}$	Position of agent α
$\vec{v}^{\alpha}$	Velocity of agent α
$\vec{t}^{\alpha}$	Target of agent α
$\vec{p}^{\alpha\beta} = \vec{p}^{\alpha} - \vec{p}^{\beta}$	Offset between the agents α and β

2 Related Work

In this chapter we give an overview of related work. First we discuss some approaches regarding supervised learning and learning from demonstration. Afterward, we take a look at some applications of locally weighted regression followed by a summary of different techniques for social path planning. In the last section we present some applications of reactive methods in the context of robotics.

2.1 Supervised Learning

As robots learn to accomplish more and more tasks, they become increasingly useful in daily life. By sharing their environment with humans it is necessary that robots move in a human compatible way. In this thesis we present an algorithm where we learn paths from human-like demonstrations. We therefore use a supervised learning approach to learn a reactive path planning algorithm. Supervised learning is a machine learning technique where the training data consists of a set of examples and corresponding labels. In robotics, numerous applications use techniques of learning from demonstration, where training data is generated from teacher's demonstrations (see [1, 6, 8, 25, 29]). In the following, we discuss some applications of learning from demonstration in more detail. Pomerleau [35] uses human demonstrations to train a van to drive autonomously. He equips the van with a camera facing to the ground and collects images and steering commands while driving the van manually. By using these demonstrations he trains a neural network to learn a mapping from video images to steering commands. This method enables him to drive the van with a speed up to 20 miles per hour on different road types. An algorithm proposed by Ratliff et al. [36] uses hand drawn example paths on a 2d map of an environment to train a robot's path planning. From these examples they learn features describing the paths and use them to learn a feature-to-traversal-cost mapping. The authors use their method to train a path planning algorithm for an outdoor robot which produces paths with similar characteristics, even in unseen areas. For finding out more about applications, different techniques and different design choices of learning from demonstration the reader is encouraged to look into the survey from Argall et al. [3].

In this thesis we aim to reproduce demonstration paths from an existing technique that has been learned from human motion. Learning a mapping from examples to continuous outputs is denoted as regression. There are multiple different regression techniques (see Bishop [7]). The regression method used in this thesis is locally weighted regression (see Cleveland and Devlin [13]). By combining locally weighted regression and k-nearest neighbor search Li et al. [28] are able to predict the short-term traffic flow on road junctions. They use k-nearest neighbor search to find the k most similar data points. To receive the traffic forecast they then use locally weighted regression to weight the data points depending on the daytime the data is recorded. Koropouli et al. [26] teach a robot to manipulate compliant tools and compliant objects with an appropriate force. For this task it is necessary not only to control the position of the device but also to reproduce the grasping force over time. They use locally weighted regression to learn a position and a control policy from human demonstrations. Locally weighted regression can also be used to teach a robot to play ping pong. Matsushima et al. [31] therefore use locally weighted regressions to predict the time until the ball hits the paddle, the velocity change by hitting the ball and the time the ball will need to return. On these predictions they build a feed-forward control scheme to control the paddle. The controlled robot was able to play successfully against an “easy” playing human opponent. Schaal et al. [39] use locally weighted regression to teach a robot devil-sticking. They learn the mapping from the state of an incoming stick on one hand to the state describing the stick when reaching the other hand. With use of this regression they were able to successfully teach their robot a “left-right-left-etc. juggling”. For more information on locally weighted regression we refer to the survey from Atkeson et al. [5] which discusses different types of locally weighted regression and points out approaches using these techniques.
In this thesis we use locally weighted regression to learn a state to action mapping from demonstrations of human-like paths. As in Li et al. [28] we limit the used demonstrations in the prediction, but we use a kd-tree to find all points within a defined distance. Matsushima et al. [31] use regression to learn multiple different mappings. As in their approach, we use locally weighted regression to model different state to action mappings (see 4.1).

2.2 Social Path Planning

This thesis presents a method for human-like navigation for a mobile robot using a reactive method that is based on potential fields. Much effort has gone into the development

of algorithms for driving a robot without collisions through an environment with moving obstacles. Many path planning algorithms do not explicitly try to create human-like paths, but rather focus on avoiding collisions (see [10, 12, 14, 17]). Fiorini and Shiller [15] propose an algorithm for collision avoidance of robots in dynamic environments. They classify the velocities of the robot at a time into collision-free velocities and into velocity obstacles, i.e., velocities which will lead to a collision. They use a tree-based heuristic search within these chosen velocities to find a collision-free path to the target. An algorithm suited for collision avoidance for fast traveling robots in dynamic environments is proposed by Fox et al. [16]. Fox et al. carry out their planning in the velocity space. They limit the admissible velocities to the ones that are safe and within a dynamic window. In this context safety means that the robot can stop before reaching the next obstacle. The dynamic window contains all the velocities which are reachable within a short time interval accounting for the limited acceleration of the robot. From the admissible velocities they choose the command that maximizes an objective function. With this approach they can drive a robot fast and safely through a changing dynamic environment. Other algorithms explicitly account for humans as "obstacles". Ziebart et al. [40] suggest an algorithm to predict human movement within an environment. They use this knowledge to plan robot paths which avoid humans. For learning the human behavior, the authors compute features from the surroundings to describe the environment the human moves through. By using these features they first compute a map representing the likelihood to encounter a human, which is then used to plan a path for the robot. If they encounter a moving human along the way they predict its path and alter the cost map along the humans predicted path. Using the updated cost map the robot replans its path to avoid the human.

While the algorithms discussed so far enable a robot to move without collisions, Müller et al. [33] propose an algorithm that uses moving humans to drive collision free and efficiently. They create an algorithm which enables a robot to classify people into possibly moving obstacles or "leaders" to follow. This enables the robot to move in a social compatible way within human groups through populated environments. The authors combine a people tracking system with an iterative A* planner to guide the robot to its goal moving with people whenever possible.

Other authors focus on creating paths which look like human paths to move in a human compatible way. In the following we will discus a selection of different reactive social path planning algorithms. One of the most famous methods for social path planning is the social force model from Helbing and Molnár [22]. The authors propose to use different attractive potentials to move towards the target, special places or friends

and repulsive potentials to avoid other agents or walls. This method performs quite well (see Johansson et al. [23]) and, when applied to a large number of agents, also produces some behavior, which is shown by real crowds, like lane formation or an alternating flow of people at a door. To calibrate the parameters of the different potentials in the social force model Johansson et al. [23] use tracking of pedestrians in video recordings. They vary the parameters of the model to fit simulated pedestrians to tracked data and investigate the use of different forms for the repulsive interaction potential. Using the social force model and video data Helbing and Johansson [21] investigate the forming of self-organized spatio-temporal patterns in crowded areas, like the forming of lanes or oscillatory flows at doorways. The authors set their focus on extremely crowded areas and pedestrians in panic. They propose the use of some additional forces when simulating panic situations because humans in panic do not maintain a minimum distance. Helbing and Johansson introduce some additional forces like body force or sliding friction force to simulate these situations. Karamouzas et al. [24] present an algorithm for collision avoidance for simulated pedestrians. Their simulated pedestrians scan their surroundings for possible future collisions. If a pedestrian predicts a collision he computes the most efficient motion to avoid this collision. This results in an evasion force for each possible collision. After applying these forces, they recheck for new possible collisions. In their experiments the authors show that this method creates smooth avoidance behavior and evasion in a “natural way”. A similar technique for long-term collision prediction enables Paris et al. [34] to build a reactive pedestrian simulation which is suited for the simulation of crowds. The authors try to create realistic paths by an additional calibration of their algorithm using motion capture data from walking humans. Based on this human-fitted long-time collision prediction they are able to create human-like paths. When simulating crowds they are also able to avoid typical problems of microscopic approaches like oscillations and jams.

Guy et al. [19] create a model for human-like collision avoidance. They therefore extend the principle of velocity obstacles (see Fiorini and Shiller [15]) so that each agent only resolves half of a collision. This leads to perfect collision avoidance if all agents follow this strategy. Additionally, they incorporate different human behaviors like reaction time, physical constraints, and a personal space around every agent. To fit their parameters and to test their algorithm they use motion capture data of human crossings. The authors are able to reproduce trajectories similar to the trajectories traveled by humans. Brogan and Johnson [11] propose an algorithm for generating human-like walking paths. For each agent they compute at each time step the maximum velocity and the heading. These values are then used to integrate position, velocity and heading

forward in time. For fitting their parameters they capture motion data from different persons on different walking tasks.
While reactive algorithms use a state-to-action mapping to generate a command to move a robot, other social path planning algorithms predict whole trajectories. Kuderer et al. [27] present an algorithm to create human-like motion. They use motion capture data from walking pedestrians and features to capture the properties of human walking paths. While generating human paths they also account for different decisions to pass left or right. On the one hand the prediction of whole trajectories using features learned from human motion enables them to generate human-like paths. On the other hand the generation of all the decisions to pass left or right limits the algorithm in the number of agents it can simulate within reasonable time. The authors demonstrate successfully these human-like paths on a real robot. Arechavaleta et al. [2] assume that humans minimizes a cost function while walking. The authors record multiple human walking paths and try to find the underlying function a human strives for. They found out that human paths minimize the time derivative of the curvature of the path. Based on this result they propose an approximation of human trajectories.
In this thesis we propose a reactive method for creating human-like paths, similar to the social force model proposed by Helbing and Molnár [22]. In contrast to this method we do not try to model the different potentials in closed form, but we use locally weighted regression to represent these potentials using human-like paths. To create training data we use paths from the model proposed by Kuderer et al. [27]. In contrast to Guy et al. [19] we do not explicitly model human behaviors, but we aim to reproduce them implicitly by learning from human-like paths.

2.3 Reactive Models

Reactive methods have been used widely within the area of robotics to perform all sorts of tasks. Roberts et al. [37] combine different known methods to implement a reactive navigation system on an underground mining vehicle to operate it at full-speed in a production mine. They use a laser scanner to detect walls and feed this data to the reactive navigation. In addition to the reactive local navigation they are using a global, topological map to select the next global sub-goal. Haddad et al. [20] present a reactive navigation algorithm for cross-country navigation using a pair of stereo cameras. With help of the stereo cameras they estimate a probability that cells in sight are traversable. Based on these estimates they generate a potential field to direct the robot towards its goal and

around obstacles. Arkin [4] integrates behavioral, perceptual, and world knowledge in a reactive navigation approach without the explicit use of a global world model. In his experiments he is able to incorporate different types of grounds, like grass, gravel or concrete and to use this knowledge while navigating. One problem of reactive potentials for navigation is the integration of the robot's kinematic constraints. Minguez et al. [32] propose a special transformation to transform either the robot's workspace or the configuration space to account for the kinematic constraints of a non-holonomic robot. They map each point within the robot's coordinate frame, that is reachable within one motion command, into the ego-kinematic space. While using this transformation one can treat the robot as a free-flying-object, while indirectly account for the robot's dynamics. This allows to incorporate the dynamics of a robot into arbitrary reactive methods without changing the method. Ge and Cui [18] tackle another problem of using potential field methods for navigation - the problem that the target is not reachable if it is too close to an obstacle. In this case the obstacle's repulsive potential is stronger than the target's attractive potential. To solve this problem the authors propose a repulsive potential for obstacles which includes the distance between the agent and its target.

2.4 Summary

In this thesis we use training data generated by a predictive path planning method to learn a policy to generate human-like paths for a mobile robot. To generate training data we use a predictive method which is proposed by Kuderer et al. [27]. While this algorithm predicts whole trajectories, we learn a reactive state to action mapping. In contrast to the well known social force model proposed by Helbing and Molnár [22] we do not try to describe the potentials in closed form and then fit the parameters from human demonstrations (compare [21, 23]), instead we use a non-parametric regression method. To be more precise, we use locally weighted regression to model an attractive and a repulsive potential from the given demonstrations (see 4.1).

3 Fundamentals

This chapter explains some of the basic principles, which are helpful to understand this thesis. First, we discuss locally weighted regression, a supervised learning method which predicts functional values by weighting known data. We use this method to represent the potentials used in this thesis. We then explain the idea of the kd-tree, a data structure to store and fast query multi-dimensional data. This data structure is used in this thesis to store the training data for the regression. Afterward, we present the social force method proposed by Helbing and Molnár [22] as well as the predictive path planning method proposed by Kuderer et al. [27]. In the evaluation of this thesis we compare paths created by our algorithm with paths created by these methods. We additionally use the second method to create training data for our algorithm. In the end of this chapter we explain box plots, a method to visualize the distribution and variation of numeric data.

3.1 Locally Weighted Regression

Regression is a supervised learning technique which learns a mapping from input data to real valued output. We do not know the function which creates this mapping but we have been given some training data created by this function. This training data consists of a set of inputs $\mathbf{X} = \{x_1, \ldots, x_N\}$ with the corresponding outputs $\mathbf{Y} = \{y_1, \ldots, y_N\}$. There are different approaches to solve this task. On the one hand there are parametric approaches. These methods aim to model the underlying function $f(x_i) = y_i$ using the training data. An example for a parametric method is polynomial regression, where a polynomial function is fitted to the data. On the other hand there are non-parametric methods. These methods do not explicitly model the underlying function, but predict outputs with the use of the data of the training set. An example for such a method is Locally weighted regression (see Bishop [7]), explained here. If a demonstration arises, this data is not used to improve some function, but is just added to the training set. If one queries a data point the data in the training set is weighted by the distance to the query, and the weighted output is returned. This technique avoids the possibly involved creation of an explicit model.

To predict the output y_{N+1} of a data point with input x_{N+1} we weight the outputs of the points in the training set $\mathbf{Y} = \{y_1, \ldots, y_N\}$ depending on the distance of their inputs $\mathbf{X} = \{x_1, \ldots, x_N\}$ to the input value x_{N+1} of the query. The weighting of the data is done by using a kernel $k(x_i, x_{N+1})$. The function value y_{N+1} for an input x_{N+1} is computed as follows:

$$y_{N+1} = f(x_{N+1}) = w \sum_{i=1}^{N} k(x_i, x_{N+1})\, y_i, \tag{3.1}$$

where $w = \sum_{i=1}^{N} k(x_i, x_{N+1})$ is the normalization, so that the weights sum up to one.
If we for example want to predict the price of a house depending on the square meters and the number of rooms we could use locally weighted regression. The inputs would be vector valued and would look like $\mathbf{X} = \{(120m^2,\ 3 \text{ rooms}), (281m^2,\ 5 \text{ rooms}), \ldots\}$. The outputs would be the corresponding costs $\mathbf{Y} = \{120k\ €, 340k\ €, \ldots\}$. To predict the value of a house with $237m^2$ and 4 rooms we would use the vector $x_{N+1} = (237m^2,\ 4 \text{ rooms})$ as query. We then would predict the price y_{N+1} using Equation 3.1.
To predict the output of a query we need to calculate the kernel between each of the training inputs and the query point. If we use a huge amount of data this calculation can get computationally involved. To speed up locally weighted regression using a Gaussian kernel we predict a value by only weighting training data within the proximity of the query point. This is possible because the exponential function within a Gaussian kernel leads to weights which decrease fast with growing distance between the weighted points. To efficiently implement this approximation one needs fast access to all points within the training data. One possibility is to store the training data using a k-d tree.

3.2 k-d Tree

A k-d tree is a data structure which stores points with dimensionality k and allows for fast access to all points within a defined volume. A k-d tree therefore recursively divides the volume in which the data is stored using hyperplanes. Each hyperplane is set that half of the data in the volume lies on each side of the plane. The dimension on which the tree divides the volume is changed in each step. An example of a k-d tree with dimensionality $k = 2$ is shown in Figure 3.1. The volume is first divided on the x-axis (line x_0), which leads to the root-node. The volume is then further subdivided by y_0 respective y_1. Note that while the first and third (x_1) division is done on the x-axis, the second division is carried out on the y-axis. To query all points within a defined volume we recursively look if the space on the left, right of or on both sides of the hyperplane is included within the search volume. We then continue our search only on the sub-

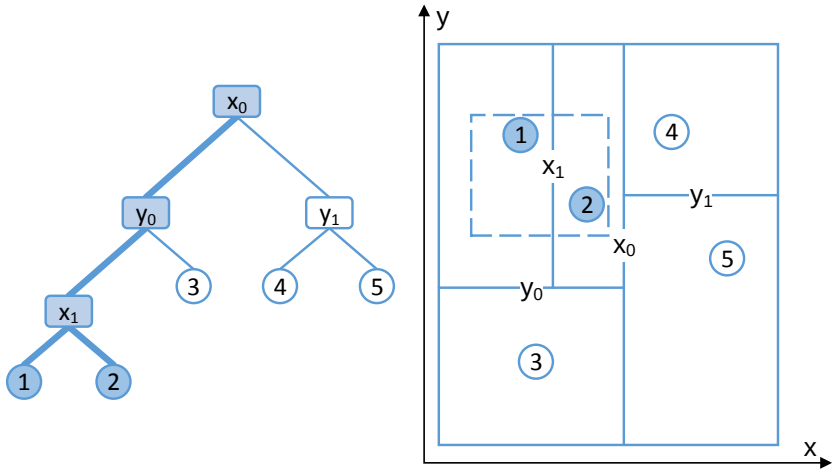

Figure 3.1: Example of a k-d Tree. Left: Tree based representation of an example k-d tree. The boxes represent the hyperplanes on which the tree is divided. The bold path through the tree shows the search for the points within the dashed box. Right: The points within two-dimensional space together with the lines dividing the volume. Also shown is the dashed box for which we query all points.

volumes which are included within the search volume. If the volume of the k-d tree is huge and the search volume is small, the search space decreases rapidly. Figure 3.1 shows the query for all points within a small area (visualized by the dashed box in the right plot). The left side of the plot shows the search within the tree (bold lines). In the first step the total volume lies left of x_0, so the right half of the k-d tree must not be searched further. In the second step the total volume lies above y_0. In the last step both sides of the tree must be searched further. As these volumes are not further divided the found points must be checked against the search volume.

3.3 Social Force Method

In this thesis we compare the paths created by the proposed method with paths created by the social force method and with paths created by the predictive method we use to create training data. In this section we shortly explain the social force method proposed by Helbing and Molnár. For a detailed explanation of this method see Helbing and Molnár [22]. The main idea of the social force approach is that humans are driven by some sort of potential. This potential is the sum of different types of potentials. On the one hand there are attractive potentials driving humans towards their targets or some

interest points. On the other hand there are repulsive potentials driving humans away from other humans or walls. Mathematically the social forces acting on a pedestrian can be computed as follows.
First we compute the desired traveling direction as the direction towards the pedestrian's target:

$$\vec{e}^{\alpha}(t) := \frac{\vec{t}^{\alpha} - \vec{p}^{\alpha}(t)}{\left\|\vec{t}^{\alpha} - \vec{p}^{\alpha}(t)\right\|}. \tag{3.2}$$

From this direction the attracting target potential can be computed as follows:

$$\vec{F}_0^{\alpha}\left(\vec{v}^{\alpha}, v_0^{\alpha}\vec{e}^{\alpha}\right) := \frac{1}{\tau^{\alpha}}\left(v_0^{\alpha}\vec{e}^{\alpha} - \vec{v}^{\alpha}\right), \tag{3.3}$$

where v_0^{α} is the desired traveling speed of pedestrian α and τ^{α} controls the acceleration. If there are other pedestrians present, each of them creates a repulsive potential:

$$\vec{f}^{\alpha\beta}\left(\vec{p}^{\alpha\beta}\right) := -\nabla_{\vec{p}^{\alpha\beta}} V^{\alpha\beta}\left[b\left(\vec{p}^{\alpha\beta}\right)\right]. \tag{3.4}$$

$V_{\alpha\beta}$ describes an elliptic monotonic decreasing potential with semi minor axis b. This potential is described by an exponential function with parameters σ:

$$V^{\alpha\beta}(b) := V_0^{\alpha\beta} e^{\frac{-b}{\sigma}}. \tag{3.5}$$

This potential has its largest extend into the desired traveling direction of the pedestrian β. The parameter b is computed depending on the step width $s^{\beta} := v^{\beta}\Delta t$ of pedestrian β:

$$2b := \sqrt{\left(\|\vec{p}^{\alpha\beta}\| + \|\vec{p}^{\alpha\beta} - v^{\beta}\Delta t\vec{e}^{\beta}\|\right)^2 - \left(v^{\beta}\Delta t\right)^2}. \tag{3.6}$$

Another source for repulsive potentials are borders. From the vector $\vec{p}^{\alpha B} = \vec{p}^{\alpha} - \vec{p}^{\alpha B}$, where $\vec{p}^{\alpha B}$ denotes the point on the border closest to pedestrian α, the force acting on the pedestrian is computed as follows:

$$\vec{F}^{\alpha B}\left(\vec{p}^{\alpha B}\right) := -\nabla_{\vec{p}^{\alpha B}} U^{\alpha B}\left(\left\|\vec{p}^{\alpha B}\right\|\right). \tag{3.7}$$

Helbing and Molnár propose to use the following potential $U^{\alpha B}$ to push a pedestrian away from a wall:

$$U^{\alpha B}\left(\left\|\vec{p}^{\alpha B}\right\|\right) = U_0^{\alpha B} e^{-\|\vec{p}^{\alpha B}\|/R}. \tag{3.8}$$

Attractive places like friends, street artist or window displays create attractive potentials as follows:

$$\vec{f}^{\alpha i}\left(\left\|\vec{p}^{\alpha i}\right\|, t\right) := -\nabla_{\vec{p}^{\alpha i}} W^{\alpha B}\left(\left\|\vec{p}^{\alpha i}\right\|, t\right). \tag{3.9}$$

The potential $W^{\alpha B}\left(\left\|\vec{p}^{\alpha i}\right\|, t\right)$ is supposed to decrease with time, because pedestrians lose interest. A pedestrian will only consider attractive places or other pedestrians in traveling direction. Therefore Helbing and Molnár compute a direction dependent weight:

$$w\left(\vec{e}, \vec{f}\right) := \begin{cases} 1 & \text{if } \vec{e} \cdot \vec{f} \geq \left\|\vec{f}\right\| \cos\varphi \\ c & \text{else} \end{cases}. \tag{3.10}$$

This weighting applies to the forces from other pedestrians $\vec{f}^{\alpha\beta}\left(\vec{p}^{\alpha} - \vec{p}^{\beta}\right)$ as well as to the attractive place forces $\vec{f}^{\alpha i}\left(\vec{p}^{\alpha} - \vec{p}^{i}, t\right)$:

$$\vec{F}^{\alpha\beta}\left(\vec{e}^{\alpha}, \vec{p}^{\alpha} - \vec{p}^{\beta}\right) := w\left(\vec{e}^{\alpha}, -\vec{f}^{\alpha\beta}\right) \vec{f}^{\alpha\beta}\left(\vec{p}^{\alpha} - \vec{p}^{\beta}\right) \tag{3.11}$$

$$\vec{F}^{\alpha i}\left(\vec{e}^{\alpha}, \vec{p}^{\alpha} - \vec{p}^{i}, t\right) := w\left(\vec{e}^{\alpha}, \vec{f}^{\alpha i}\right) \vec{f}^{\alpha i}\left(\vec{p}^{\alpha} - \vec{p}^{i}, t\right) \tag{3.12}$$

All these forces computed so far sum up to a total force drive the pedestrian into the desired direction:

$$\begin{aligned} \vec{F}^{\alpha}(t) := \vec{F}_0^{\alpha}\left(\vec{v}^{\alpha}, v_0^{\alpha}\vec{e}^{\alpha}\right) + \sum_{\beta} \vec{F}^{\alpha\beta}\left(\vec{e}^{\alpha}, \vec{p}^{\alpha} - \vec{p}^{\beta}\right) \\ + \sum_{B} \vec{F}^{\alpha B}\left(\vec{e}^{\alpha}, \vec{p}^{\alpha} - \vec{p}^{\alpha B}\right) + \sum_{i} \vec{F}^{\alpha i}\left(\vec{e}^{\alpha}, \vec{p}^{\alpha} - \vec{p}^{i}, t\right). \end{aligned} \tag{3.13}$$

Using this force we can update the desired velocity $\vec{w}^{\alpha}$:

$$\frac{d\vec{w}^{\alpha}}{dt} := \vec{F}^{\alpha}(t) + \text{fluctuations}. \tag{3.14}$$

The social force model does not allow the velocity of a pedestrian to exceed a defined maximal velocity $v_{\max}^{\alpha}$, so the velocity is constrained as follows:

$$\frac{d\vec{p}^{\alpha}}{dt} = \vec{v}^{\alpha}(t) := \vec{w}^{\alpha}(t)\, g\left(v_{\max}^{\alpha}, \|\vec{w}^{\alpha}\|\right), \tag{3.15}$$

where $g(\cdot,\cdot)$ is defined as:

$$g\left(v_{\max}^{\alpha}, \|\vec{w}^{\alpha}\|\right) := \begin{cases} 1 & \text{if } \|\vec{w}^{\alpha}\| \leq v_{\max}^{\alpha} \\ \frac{v_{\max}^{\alpha}}{\|\vec{w}^{\alpha}\|} & \text{else} \end{cases} \tag{3.16}$$

In summary the social force method first computes attractive potentials for the target and attractive places and repulsive potentials for all other pedestrians and walls. These forces are then summed up to a total force. With this force the desired velocity of the pedestrian is updated. The velocity of the pedestrian is then restricted not to exceed a maximal velocity. At the end this restricted velocity is used to update the position of the traveling pedestrian.

3.4 Predictive Path Planning Method

In this thesis we aim to reproduce paths generated by a given predictive social path planning method with use of a reactive method. This section explains shortly the predictive method from Kuderer et al., which we use to create our demonstration paths. For a detailed explanation see Kuderer et al. [27].
Kuderer et al. learn a method which predicts whole trajectories for all involved agents. These trajectories are optimized to match features learned from human navigation, which capture the characteristics of the trajectories and include travel time, velocity, acceleration and distance between agents.
For representing the trajectories they use a continuous, spline-based representation. For an agent α the trajectory x^{α} is defined as a mapping from time t to a configuration $x \in \mathcal{X}$:

$$t \mapsto x^{\alpha}(t) \in \mathcal{X}. \tag{3.17}$$

This configuration x includes a two-dimensional state describing the position of agent α. If more than one agent is involved the joint trajectory for all N agents is defined as the following Cartesian product:

$$\mathrm{x}(t) = x^{1}(t) \times x^{2}(t) \times \ldots \times x^{N}(t) \in \mathcal{X}^{N}. \tag{3.18}$$

If two agents pass each other they have two possibilities to avoid a collision, either both agents pass left or right. These decisions result in topological variants of the joint trajectory. Each topological variant φ is a subset of $\mathcal{X}^{N}$ where each two agents, which cross each other, choose the same side to pass. Kuderer et al. assume that for each topological variant there is a probability that humans would choose this variant. This probability distribution $p(\mathrm{x})$ is assumed to be a function of some features. For a joint

trajectory they define features as being a mapping from the joint trajectory to a real number:

$$f_i : \mathcal{X}^N \to \mathbb{R}. \tag{3.19}$$

Using these features they optimize the most likely trajectory to match the features $\mathbf{f}_{\mathcal{D}}$ they measured empirically from human navigation:

$$\mathbb{E}_{p(\mathrm{x})}\left[\mathbf{f}(\mathrm{x})\right] = \mathbf{f}_{\mathcal{D}} = \frac{1}{|\mathcal{D}|} \sum_{\mathrm{x}_k \in \mathcal{D}} \mathbf{f}(\mathrm{x}_k). \tag{3.20}$$

From this approximation of the distribution of joint trajectories they can then choose the topological variant that best matches the empirical features $\mathbf{f}_{\mathcal{D}}$ and run it on a real robot. The authors use four different features. First they use as travel time the time the agent needs to reach its target:

$$f^{\alpha}_{\text{travel time}} = t^{\alpha}_{\text{travel time}}. \tag{3.21}$$

The acceleration of the agent is computed via the second derivative of the state:

$$f^{\alpha}_{\text{acceleration}} = \int_t \|\ddot{x}^{\alpha}(t)\|^2 \, dt. \tag{3.22}$$

For matching the desired traveling speed they compute a velocity feature:

$$f^{\alpha}_{\text{velocity}} = \int_t \|\dot{x}^{\alpha}(t)\|^2 \, dt \tag{3.23}$$

and for avoiding collisions they use a velocity dependent distance feature for each two agents:

$$f^{\alpha}_{\text{distance}} = \sum_{\beta \neq \alpha} \int_t \frac{v^{\alpha}_t}{\|x^{\alpha}(t) - x^{\beta}(t)\|^2} dt. \tag{3.24}$$

These features together form the feature vector $\mathbf{F}$. The authors use a weighted sum of these features to choose the most likely trajectory. The weights to compute this cost value are learned from human navigation paths.

The number of possible topological variants grows exponentially with the number of agents. The computation of all these paths for multiple agents in reasonable time and memory is a limitation of this predictive method.

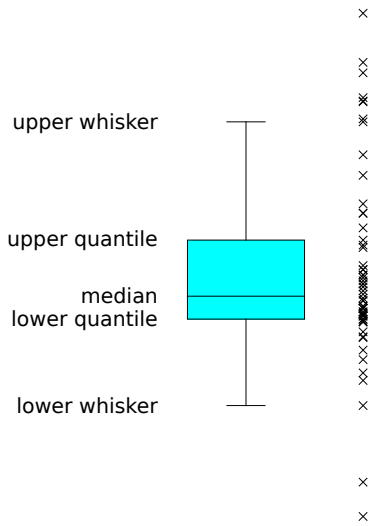

Figure 3.2: A box plot together with the data the box plot represents (right). We can see that, as the data points just below the median are cumulated the distance between the median and the lower quantile is smaller than the distance between the median and the upper quantile.

3.5 Box Plot

A box plot is a plot that visualizes the distribution of numerical data. It shows the data with the use of five values: the median, two quantiles and two whiskers. An exemplary box plot, together with the data whose distribution it visualizes, is shown in Figure 3.2. The median divides the data so that half of the data has a greater value than the median and half of the data has a value less than the median. Between the median and the quantiles lies each 25% of the data values. The whiskers are plotted from the quantiles with a length of 1.5 times the range between the upper and lower quantile, shrunken back to the nearest data value. A box plot shows more expressive information than a mean together with the standard deviation because it also visualizes the skewness of the distribution described by the data. The data used to create the box plot in Figure 3.2 accumulate just below the median of the box plot. This is represented in the box plot by a smaller distance between the lower quantile and the median than the distance between the median and the upper quantile.

4 Learning Reactive Robot Navigation Policies from Predictive Trajectory Planning

This chapter presents our approach to learn a reactive navigation policy for social navigation of a mobile robot. There are two main approaches to solve this task: The predictive and the reactive approach. Predictive methods compute commands to drive a robot by generating paths for all involved agents from their positions to their targets. These methods are able to create human-like paths, however, they are often computational expensive. Reactive methods use potentials to learn a mapping from the state of all involved agents to commands for the robot. These methods are mostly fast, but they often do not lead to human-like paths. In this thesis we try to combine both approaches. We present a reactive approach which is learned from paths generated by a predictive method. Most of the reactive approaches for social navigation of a mobile robot aim to learn parametric potentials. In this thesis we use a non-parametric regression method. In contrast to approaches which learn parametric potentials the non-parametric regression leaves more freedom to represent the data. We also do not need to know the structure of the data we want to represent beforehand. The predictive method, used in this thesis to create training data, generates whole trajectories for all involved agents using features learned from human navigation. We learn our model by first sampling a set of situations for which we generate acceleration commands using the predictive method. We then use locally weighted regression to generate commands for arbitrary situations. To efficiently use locally weighted regression we need to reduce the state space describing multiple agents moving towards their targets while evading each other into a lower dimensional state space. We therefore focus on the situation where two agents evade each other while moving towards their targets. We then assume that the evasion of multiple agents can be approximated by evading each agent individually. To further reduce the state space we split the potential, driving an agent towards its target while evading an other agent, into two separate potentials. The first of these potentials is an attractive potential driving an agent towards its target while the second potential is a repulsive potential representing the cooperative collision avoidance between two agents. To generate training data for the attractive potential we create demonstration paths for one agent, which

is moving towards its target. To create the repulsive potential we generate demonstration paths for two agents. From these paths we compute the relative state between the agents and an acceleration command for the first agent. To generate the training data for the repulsive potential our method reduces the twelve-dimensional joint state space of two agents' positions, velocities and targets into a five-dimensional state space by taking advantage of symmetries and by making assumptions about situations we encounter. By applying acceleration commands from both potentials we are able to drive a robot towards its target while evading other agents.
In the next section, we describe the two potentials in more detail and how we generate the training data. We also explain the reduction of the twelve-dimensional joint state space into the used five-dimensional one together with the conversion between these states. In the end of this chapter we explain how we use the learned policy for navigation.

4.1 Potentials

The policy learned in this thesis uses two potentials, one for approaching the target and one for evading other agents. In the following, we explain how we generate training data for these potentials. We also explain the state spaces used to parametrize the two potentials and we take a look at their dimensionality. We then make use of symmetries and make some assumptions to reduce these state spaces in order to use them for locally weighted regression.

4.1.1 Approaching the Target

The intend to move towards a target can be modeled by a potential $P_{\text{target } \alpha}$ which accelerates the agent α into the direction of the target. We assume that the agent wants to reach its target and stop there. Due to this assumption we could simply parametrize the potential using the offset between the position of the agent $\vec{p}^{\alpha}$ and its target $\vec{t}^{\alpha}$ together with the agent's current velocity $\vec{v}^{\alpha}$. This results in the following potential:

$$P_{\text{target } \alpha} = P_{\text{target } \alpha}\left(p_x^{\alpha} - t_x^{\alpha}, p_y^{\alpha} - t_y^{\alpha}, v_x^{\alpha}, v_y^{\alpha}\right) \tag{4.1}$$

If the paths used as training data are independent with respect to a global position and orientation we are able to use this invariance to reduce the state space. This assumption holds for the predictive method, which we use to create training data. The predictive

method does not use a map with different traversal cost, but optimizes the paths by minimizing costs composed of weighted features. The features used to generate the paths are travel time, velocity, acceleration and distance between agents. All these features are independent of a global position and are therefore not influenced by a rotation or movement of the coordinate system. To reduce the state space we move the coordinate system in such a way that the agent's target lies on the origin. Furthermore, we turn the coordinate system so that the agent's position lies on the y-axis. We parametrize the potential with the distance between the agent's position and its target $d^{\alpha t^{\alpha}}$ and its velocity $\vec{v}^{\alpha}$. By using this transformation we are able to reduce the state space, which is used to parametrize the potential by one dimension. In this thesis we use the following attractive potential:

$$P_{\text{target }\alpha} = P_{\text{target }\alpha}\left(d^{\alpha t^{\alpha}}, v_x^{\alpha}, v_y^{\alpha}\right). \tag{4.2}$$

To learn this potential we use demonstration paths with one agent. We first sample different states to create training data for the regression. Without loss of generality we set the target of the agent onto the origin and place the agent with different distances to its target and different velocities onto the y-axis. For each of the sampled states we then generate a demonstration path $\vec{p}^{\alpha}(t)$. From these paths we compute the acceleration using finite differences:

$$a_{\text{target }\alpha}\left(d^{\alpha t^{\alpha}}, v_x^{\alpha}, v_y^{\alpha}\right) = \frac{\dot{\vec{p}}^{\alpha}(t_0 + \delta t) - \dot{\vec{p}}^{\alpha}(t_0)}{\delta t}, \tag{4.3}$$

where t_0 is the start time of the path. The resulting potential is composed of acceleration commands pushing an agent towards its target accounting for its displacement to the target and its velocity.

4.1.2 Avoiding Collisions

To account for more than one agent, we must incorporate interactions of multiple agents. We therefore learn a repulsive potential that captures the cooperative collision avoidance between two agents α and β: $P_{\text{repulsive }\alpha\beta}$. In our framework each agent α is described by its two-dimensional position $\vec{p}^{\alpha}$, two-dimensional velocity $\vec{v}^{\alpha}$ and its two-dimensional target $\vec{t}^{\alpha}$. Therefore the interaction of two agents α and β can be described by a joint state space S of twelve dimensions:

$$S = \left(p_x^{\alpha}, p_y^{\alpha}, v_x^{\alpha}, v_y^{\alpha}, t_x^{\alpha}, t_y^{\alpha}, p_x^{\beta}, p_y^{\beta}, v_x^{\beta}, v_y^{\beta}, t_x^{\beta}, t_y^{\beta}\right). \tag{4.4}$$

To predict a value, locally weighted regression weights known training data within the proximity of the query point. To get a valid prediction we need to fill the volume of the state space, from which we plan to query points, with training data. It is not feasible to fill a sufficient volume of the twelve-dimensional state space of agents' positions, velocities and targets with enough training data to get a precise regression. In the following we therefore take a closer look at this state space and reduce it by taking advantage of symmetries and make some simplifications.

4.1.2.1 Taking Advantage of Symmetries

If moving and turning the coordinate system does not change the interaction between agents, we can use this transformation to reduce the state space for the repulsive potential. As mentioned above the predictions of the predictive method are invariant with respect to a global rotation and movement. To reduce the state space we first use one of the positions, without loss of generality $\vec{p}^{\alpha}$, together with the two-dimensional offset between the agents $\vec{p}^{\alpha\beta} = \vec{p}^{\alpha} - \vec{p}^{\beta}$ instead of both agents' positions. We then move the coordinate system so that the target of agent α lies on the origin. We then rotate the coordinate system around the target of agent α so that the position of agent α lies on the y-axis. For this rotation we first convert the offset $\vec{p}^{\alpha\beta}$ and the moved position of agent α: $\vec{p'}^{\alpha}$ into polar coordinates:

$$\vec{p}^{\alpha\beta} = \left(\varphi^{\alpha\beta}, r^{\alpha\beta}\right) \tag{4.5}$$

$$\vec{p'}^{\alpha} = \left(\varphi'^{\alpha}, r'^{\alpha}\right). \tag{4.6}$$

To rotate the coordinate system we set $\varphi'^{\alpha} = \frac{\pi}{2}$ and rotate all other quantities accordingly. The state space resulting from these changes is composed of the distance from agent α to its target: r^{α}, the rotated two-dimensional offset between the agents: $\left(\varphi'^{\alpha\beta}, r'^{\alpha\beta}\right)$, four dimensions of rotated velocities: $v'^{\alpha}_{x}, v'^{\alpha}_{y}, v'^{\beta}_{x}, v'^{\beta}_{y}$ and two dimensions of the moved and rotated target of agent β: t'^{β}. By combining these symmetries we are able to reduce the state space by three dimensions.

4.1.2.2 Assumptions to Reduce the State Space

To further reduce the state space, we assume a fixed distance between the agents' positions and the agents' targets d_{target}. This assumption fixes the value of $r^{\alpha} = d_{\text{target}}$.

Then, we convert the moved and rotated offset between agent β and its target into polar coordinates:

$$\vec{p}^{\beta t^{\beta}} = \left(\varphi^{\beta t^{\beta}}, r^{\beta t^{\beta}}\right) \tag{4.7}$$

and fix this distance to the target $r^{\beta t^{\beta}} = d_{\text{target}}$.
The second assumption is that agent β travels directly towards its target. This results in a velocity into the direction of $\varphi^{\beta t_{\beta}}$. We additionally assume that agent β travels at its fixed desired velocity v_{desired}. With this assumption we can discard $v'^{\alpha}_{x}, v'^{\alpha}_{y}$.

Using the symmetries and assumptions together we are able to reduce the state space for the repulsive potential from twelve to five dimensions. The used dimensions in the reduced state space are: the rotated velocity of agent α: $\left(v^{\alpha}_{x}, v^{\alpha}_{y}\right)$, the offset between the agents in polar coordinates $\left(\varphi^{\alpha\beta}, r^{\alpha\beta}\right)$ and the rotated velocity direction of agent β: $\varphi^{\beta t_{\beta}}$. The rotation results from rotating the coordinate system so that the position of agent α lies on the y-axis.

We then use this reduced state space to represent the repulsive potential:

$$P_{\text{repulsive } \alpha\beta} = P_{\text{repulsive } \alpha\beta}\left(v^{\alpha}_{x}, v^{\alpha}_{y}, \varphi^{\alpha\beta}, r^{\alpha\beta}, \varphi^{\beta t_{\beta}}\right) \tag{4.8}$$

4.1.2.3 Converting Between the State Spaces

In this thesis we use a reduced state space to learn the repulsive potential. We use symmetries and make some assumptions to reduce the twelve-dimensional joint state space of two agents into a five-dimensional state space. We parametrize the repulsive potential by using the reduced five-dimensional state space. To create training data we need to have a full description of two agents, which is given by the twelve-dimensional state. While using the learned policy for navigation we need to generate commands for a defined situation. This situation is described by the state of two agents. For both tasks we need to convert states. We therefore describe how to convert states from one of the state spaces into the other.
If we have given a twelve-dimensional state with positions, velocities and targets for two agents we must take the following steps to get the corresponding reduced state. The first

step is to move the coordinate system so that the target of agent α: $\vec{t}^{\alpha}$ lies on the origin. This is achieved by moving the positions of both agents and the target of agent β:

$$\vec{p}^{\alpha} \leftarrow \vec{p}^{\alpha} - \vec{t}^{\alpha} \tag{4.9}$$

$$\vec{p}^{\beta} \leftarrow \vec{p}^{\beta} - \vec{t}^{\alpha} \tag{4.10}$$

$$\vec{t}^{\beta} \leftarrow \vec{t}^{\beta} - \vec{t}^{\alpha} \tag{4.11}$$

$$\vec{t}^{\alpha} = (0, 0) \tag{4.12}$$

After moving the coordinate system we rotate it so that the position of agent α lies on the y-axis. We therefore rotate all quantities by an angle θ around the origin:

$$\theta = \operatorname{atan2}\left(p_y^{\alpha}, p_x^{\alpha}\right) - \frac{\pi}{2} \tag{4.13}$$

$$\vec{p}^{\alpha} \leftarrow \left(p_x^{\alpha} \cdot \cos(\theta) - p_y^{\alpha} \cdot \sin(\theta), p_x^{\alpha} \cdot \sin(\theta) + p_y^{\alpha} \cdot \cos(\theta)\right) \tag{4.14}$$

$$\vec{v}^{\alpha} \leftarrow \left(v_x^{\alpha} \cdot \cos(\theta) - v_y^{\alpha} \cdot \sin(\theta), v_x^{\alpha} \cdot \sin(\theta) + v_y^{\alpha} \cdot \cos(\theta)\right) \tag{4.15}$$

$$\vec{p}^{\beta} \leftarrow \left(p_x^{\beta} \cdot \cos(\theta) - p_y^{\beta} \cdot \sin(\theta), p_x^{\beta} \cdot \sin(\theta) + p_y^{\beta} \cdot \cos(\theta)\right) \tag{4.16}$$

$$\vec{v}^{\beta} \leftarrow \left(v_x^{\beta} \cdot \cos(\theta) - v_y^{\beta} \cdot \sin(\theta), v_x^{\beta} \cdot \sin(\theta) + v_y^{\beta} \cdot \cos(\theta)\right) \tag{4.17}$$

$$\vec{t}^{\beta} \leftarrow \left(t_x^{\beta} \cdot \cos(\theta) - t_y^{\beta} \cdot \sin(\theta), t_x^{\beta} \cdot \sin(\theta) + t_y^{\beta} \cdot \cos(\theta)\right) \tag{4.18}$$

We are now able to compute the offset between the agents in polar coordinates $(\varphi^{\alpha\beta}, r^{\alpha\beta})$ and the direction of target and velocity of agent β:

$$\varphi^{\alpha\beta} = \operatorname{atan2}\left(p_y^{\alpha} - p_y^{\beta}, p_x^{\alpha} - p_x^{\beta}\right) \tag{4.19}$$

$$r^{\alpha\beta} = \sqrt{\left(p_x^{\alpha} - p_x^{\beta}\right)^2 + \left(p_y^{\alpha} - p_y^{\beta}\right)^2} \tag{4.20}$$

$$\varphi^{\beta t_{\beta}} = \operatorname{atan2}\left(p_y^{\beta} - t_y^{\beta}, p_x^{\beta} - t_x^{\beta}\right) \tag{4.21}$$

The offset between the agents, the direction of target and velocity of agent β together with the transformed velocity of agent α yield the five-dimensional state S:

$$S = \left(v_x^{\alpha}, v_y^{\alpha}, \varphi^{\alpha\beta}, r^{\alpha\beta}, \varphi^{\beta t_{\beta}}\right) \tag{4.22}$$

If we have given a five-dimensional state $S = \left(v_x^\alpha, v_y^\alpha, \varphi^{\alpha\beta}, r^{\alpha\beta}, \varphi^{\beta t_\beta}\right)$ we can convert this state into a corresponding twelve-dimensional state using the assumptions we do in Chapter 4.1.2.2. The state of agent α is defined by the assumption and the velocity parameters v_x^α and v_y^α:

$$\vec{p}^\alpha = \left(0, d_{\text{target}}\right) \tag{4.23}$$

$$\vec{v}^\alpha = \left(v_x^\alpha, v_y^\alpha\right) \tag{4.24}$$

$$\vec{t}^\alpha = (0, 0) \tag{4.25}$$

$$\tag{4.26}$$

The position of agent β can be computed by adding the offset between the agents in Cartesian coordinates to the position of agent α:

$$\vec{p}^\beta = \vec{p}^\alpha + \left(r^{\alpha\beta} \cdot \cos\left(\varphi^{\alpha\beta}\right), r^{\alpha\beta} \cdot \sin\left(\varphi^{\alpha\beta}\right)\right)$$

The direction of the velocity of agent β is defined by parameter $\varphi^{\beta t_\beta}$. The norm of its velocity is defined by one of the assumptions:

$$\vec{v}^\beta = \left(v_{\text{desired}} \cdot \cos\left(\varphi^{\beta t_\beta}\right), v_{\text{desired}} \cdot \sin\left(\varphi^{\beta t_\beta}\right)\right)$$

We assume that the direction of the target of agent β is defined by its velocity. So the target can be computed by adding the normalized velocity times the distance between the agent's target and its position:

$$\vec{t}^\beta = \vec{p}^\beta + d_{\text{target}} \cdot \frac{\vec{v}^\beta}{||\vec{v}^\beta||}$$

These quantities together define the twelve-dimensional state fully describing both agents.

4.1.2.4 Learning the Potential

In this thesis we aim to represent the behavior of an agent evading other agents while moving towards a target using two independent potentials. The repulsive potential only represents the evasion of a second agent and does not drive an agent towards its target. This separation of the potential reduces the amount of training data which is needed. To learn the repulsive potential we first sample states from the reduced five-dimensional state space. These states do not describe the full states of two agents. To get a fully defined state for both agents, we convert these states into the twelve-dimensional de-

scription for two agents (see Chapter 4.1.2.3). This representation includes positions, velocities and targets for two agents. For each of these twelve-dimensional states we create a demonstration path which involves paths for two agents towards their targets. From each of these paths we compute an acceleration command a_{both} for agent α. Such a command accounts for evading agent β, but also drives agent α towards its target. We then generate a second path which only involves agent α and compute an acceleration command a_α from this path. Afterward, we subtract the second command from the first: $a = a_{\text{both}} - a_\alpha$. Using this command agent α only evades agent β and does not drive agent α towards its target at the same time. To represent the distance between the agents we use polar coordinates (see Chapter 4.1.2.1). Choosing uniformly an angles and a distances to represent the offset between the agents, we obtain more samples for states where the agents are close to each other. This advantageous property allows for more precise actions when agents are close to each other. If the agents are further apart a small error in the evasion does less damage.
The problem which arises if we use the reduced five-dimensional state space for regression is the discontinuity of angles. Points which have a small distance in Cartesian space can have a large distance in the angular component of the polar coordinates. We therefore need to account for this in the distance measure used in the regression. For measuring the similarity between states we reconvert the offset between the agents into Cartesian coordinates. We also need to take care of the angle $\varphi^{\beta t_\beta}$. We therefore use $\varphi^{\beta t_\beta}$ together with a fixed value for r to convert the vector $(\varphi^{\beta t_\beta}, r)$ into Cartesian coordinates.

4.2 Using the Learned Policy for Navigation

To drive a robot through an environment with other agents we combine the attractive potential P_{target} with the repulsive potential $P_{\text{repulsive}}$. We first use the attractive potential to compute an acceleration towards the robot's target. The robot's distance towards its target together with its velocity is used to look up all points within a defined range. For fast access to all points used in the regression we use a k-d tree. The commands of the points within the range are then weighted according to their state's distance to the input state. This weighted output of the regression is then used as attractive target command for this situation. In the next step we compute a repulsive command for each other agent within the environment. From the state of the robot, together with each of the other agents' states we compute the five-dimensional state describing the relative placement between the agents and their velocities (see Chapter 4.1.2.3). We then use

the same procedure described above to compute a command using the regression. Afterward, we sum up all of these repulsive commands and add the attractive command to the repulsive commands. Using the acceleration command $\vec{a}^{\text{robot}}$ we computed so far together with the robots current velocity $\vec{v}^{\text{robot}}$ we compute a desired velocity:

$$\vec{v}^{\text{robot}} \leftarrow \vec{v}^{\text{robot}} + \delta t \; \vec{a}^{\text{robot}} \tag{4.27}$$

To drive a robot we use a controller which uses this desired velocity together with the robot's actual velocity to compute angular and linear velocities to move the robot (see Chapter 5.7). If we want to compare paths created using the predictive method with paths we created using our approach we take the following steps to simulate paths. First, we compute the desired velocity as above. We then assume that the simulated robot is able to execute this velocity so we can update its position as follows:

$$\vec{p}^{\text{robot}} \leftarrow \vec{p}^{\text{robot}} + \delta t \; \vec{v}^{\text{robot}}. \tag{4.28}$$

We then compute a new command for the new situation. This procedure is repeated for all involved agents until they reached their targets.
In this thesis we aim to reproduce the paths generated using the predictive method. To measure the similarity between two paths we use the features proposed in the predictive method (travel time, velocity, acceleration and distance between agents). If two paths are given, which we want to compare, we first compute the feature vectors for both paths $\mathbf{F}_i$, $\mathbf{F}_j$. We then take the norm of the difference of these vectors $||\mathbf{F}_i - \mathbf{F}_j||$ as a measure of similarity.

5 Evaluation

This chapter presents experiments which evaluate the performance of the proposed algorithm. In the first section we discuss plots of the learned potentials. In this thesis we aim to reproduce paths of a predictive path planning method using a reactive method. We therefore start our evaluation with an experiment which shows that the method proposed in this work results in paths similar to the paths of the predictive method. Therefore, we evaluate the accuracy of the regression. To evaluate the quality of the paths generated by our learned policy we create paths for different situations and compare these paths against paths generated by the predictive method and against paths generated by the social force model. We therefore continue by presenting the social force method used in this thesis. We start our evaluation of the quality of the generated paths by analyzing paths involving a single agent moving towards a target. These paths are created using the attractive potential. We then turn to paths including two interacting agents. To create these paths we also use the repulsive potential, which accounts for the cooperative collision avoidance between two agents. To evaluate the performance of our method applied to multiple agents, we show paths for an example situation with five agents crossing each other. We use a Pioneer III robot to show that we are able to use the learned policy for navigation of a real robot in an environment with pedestrians. For a fair comparison of the three methods, which we use to generate paths, we also evaluate the times needed to compute commands to drive a robot. Finally, we investigate the problems occurring by the reduction of the state space, used to represent the repulsive potential.

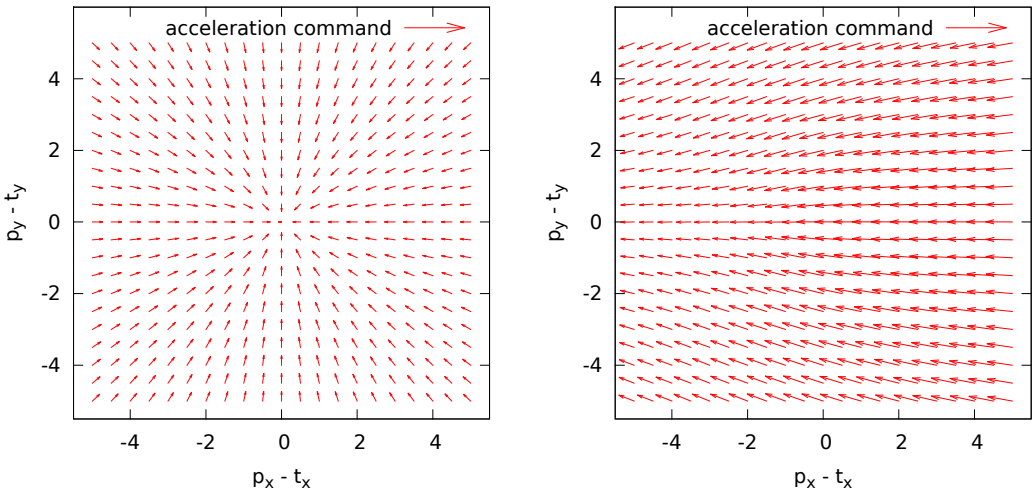

Figure 5.1: Attractive potentials plotted for different velocities. For better visualization the length of the arrows is scaled by 0.5. The potential accelerates an agent towards its target accounting for its velocity. Left: Attractive potential for $\vec{v}^{\alpha} = (0,0)$. Right: The same potential for $\vec{v}^{\alpha} = (1,0)$.

5.1 Learned Potentials for Modeling Reactive Navigation

In this section we illustrate the learned potentials. Figure 5.1 shows two plots of the attractive potential for different velocities $\vec{v}^{\alpha}$. We can see that, if the robot has zero velocity, the potential accelerates the robot towards its target (see Figure 5.1 [left]). If the robot drives with a nonzero velocity, the acceleration command generated by the attractive potential, pushes the agent's velocity towards the target (see Figure 5.1 [right]). Figure 5.2 [left] shows the learned repulsive potential $P_{\text{repulsive } \alpha\beta}\left(v_x^{\alpha}, v_y^{\alpha}, \varphi^{\alpha\beta}, r^{\alpha\beta}, \varphi^{\beta t_{\beta}}\right)$ plotted for different values of the displacement between the agents and for fixed velocities $\vec{v}^{\alpha} = (0,-1)$ and $\vec{v}^{\beta} = (0,2)$. This means agent α travels downwards and agent β travels upwards. Figure 5.3 shows a smaller region of this potential together with an example for the positions of the agents. The potential is always centered at the position of agent β and the command found at the offset position will be executed by agent α. Figure 5.2 [left] shows that agent α evades agent β when approaching it (upper half of the plot). When the agents already passed each other, agent α will no longer evade the second agent. Figure 5.2 [right] shows the same potential for a different velocity of agent β. The second agent now travels towards the left. In the states described by the upper middle of the plot agent α can cross behind agent β because agent β will have moved further into this direction.

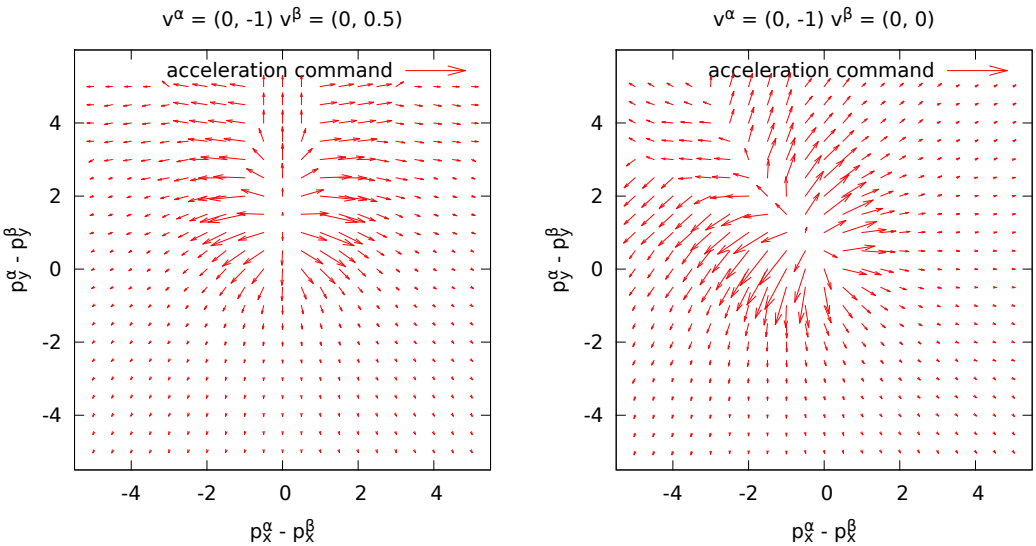

Figure 5.2: Plot of the repulsive potential that drives agents away from each other. The potential is plotted against the displacement between the agents for different fixed velocities. For better visualization the length of the arrows is scaled by 10. Left: repulsive potential for $\vec{v}^{\alpha} = (0, -1)$, $\vec{v}^{\beta} = (0,2)$, right: the same potential for $\vec{v}^{\beta} = (-2,0)$

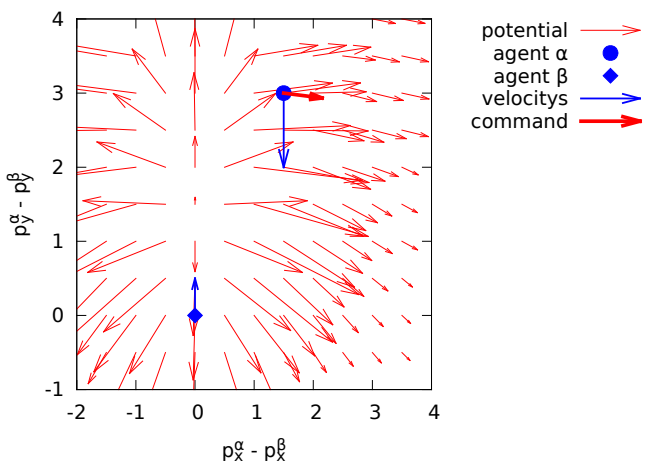

Figure 5.3: This plot shows the use of the repulsive potential using an example. The potential shown here is a smaller region of the potential shown in Figure 5.2 [left]. The potential is centered at the position of agent β. The command which can be found on the position of agent α is highlighted with a thick red arrow and will be executed by this agent.

5.2 Regression and Costs

The experiment described in this section evaluates the regression. The parameter of the locally weighted regression (see Chapter 3.1) is the standard derivation σ of the used kernel $k(\cdot,\cdot)$. In this thesis we use an isotropic Gaussian kernel:

$$k(\vec{x},\vec{y}) = \prod_i \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x_i-y_i)^2}{2\sigma}}. \tag{5.1}$$

The smaller σ the more the outcome of the regression depends on points close to the input. The bigger σ the smoother the regression. In this experiment we evaluate two error measures for different values of σ. The first error measures the difference between predicted commands using the regression and commands computed by the predictive method. The second error is based on features of whole trajectories created using the learned policy which in turn uses the regression. Our experiments suggest that the regression error coincide with the error in the feature values.

We refer to regression error as the difference of predictions done using the regression to their true values. We therefore sample a set of test data and generate the corresponding command using the predictive method. We then use the regression to compute a command for the points in this test set. We use the mean of the difference between the true value Y_i generated by the predictive method and the outcome of the regression $f(X_i,\sigma)$ over the whole test set as error:

$$\mathrm{e}_{\text{regression}} = \frac{1}{I}\sum_i^I \|Y_i - f(X_i,\sigma)\|. \tag{5.2}$$

We then compute an error based on features of whole trajectories. In this thesis we aim to reproduce the paths generated by the predictive method. To measure the similarity of paths we use the features used in the predictive method. These features describe the characteristics of the path. To compute a feature based error we use our learned policy to create a set of trajectories for single agents using parameter σ for regression (see Chapter 4.2). We also generate the same trajectories using the predictive method. For both trajectories we compute the feature vectors $\mathbf{F}_{\text{reactive}}$ and $\mathbf{F}_{\text{predictive}}$ (see Chapter 3.4). The mean of the norm of the feature vector difference over the set of example trajectories is as feature-based error:

$$\mathrm{e}_{\text{feature}} = \frac{1}{I}\sum_i^I \left\|\mathbf{F}^i_{\text{reactive}} - \mathbf{F}^i_{\text{predictive}}\right\|. \tag{5.3}$$

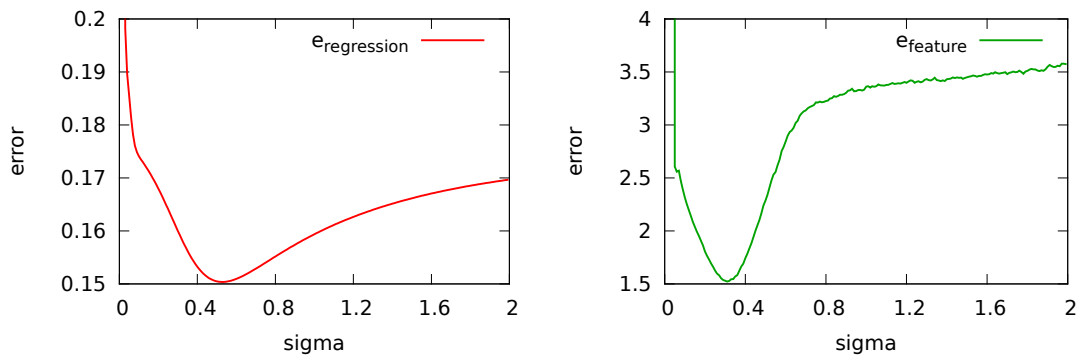

Figure 5.4: The two presented regression related errors computed for different values of the parameter σ.

We compute the presented errors for different values of the standard derivation σ. The result is visualized in Figure 5.4. The figure shows that both errors have a minimum around a value of $\sigma = 0.4$. This suggests that a value of σ, which leads to a good reproduction of the regression data, also creates paths reproducing the feature values of the predictive method. We therefore can approximately optimize our proposed method by minimizing the parameter σ via the regression error to achieve good performance with respect to the feature values.

5.3 The Social Force Model

In this thesis we compare paths generated using the learned policy to corresponding paths generated by the predictive method (see Kuderer et al. [27]) and to paths generated using the social force model (see Helbing and Molnár [22]). This section presents the social force model used in this thesis. In contrast to Helbing and Molnár [22], we did not take into account borders (see Equation 3.7) or attractive places (see Equation 3.9) because this is not supported in the predictive method used to create training data. Furthermore, the viewpoint dependent weight (see Equation 3.10) was not used. Since we do not assume known target positions for all agents we can not compute the proposed potential (see Equation 3.5 and Equation 3.6), thus we use a simplified potential. As we does not know the desired traveling direction of other pedestrians, the potential used in this thesis is circular, in contrast to an elliptic potential used by Helbing and Molnár. The radius of the potential used in this thesis is influenced by parameter b:

$$b\left(\vec{p}_{\alpha\beta}\right) := \left\|\vec{p}_{\alpha\beta}\right\|^2 . \tag{5.4}$$

To allow for a fair comparison to our method, we additionally extended the social force model to decelerate when approaching the target. We assume a linear deceleration after the agent reached a certain distance to its target. To adapt this behavior in the social force model, we change the maximum velocity so that it depends on the distance to the target:

$$v_{\max}^{\alpha} = v_{\max}^{\alpha}\left(\left\|\vec{t}_{\alpha} - \vec{p}_{\alpha}(t)\right\|\right) \tag{5.5}$$

In total the potential used in this social force model is the sum of the attracting target potential and the simplified repulsive potentials from other agents (compare to Equation 3.13):

$$\vec{F}_{\alpha}(t) := \vec{F}_{\alpha}^{0}\left(\vec{v}_{\alpha}, v_{\alpha}^{0}\vec{e}_{\alpha}\right) + \sum_{\beta} \vec{f}_{\alpha\beta}\left(\vec{p}_{\alpha\beta}\right). \tag{5.6}$$

To optimize the parameters used in the social force model we minimize the norm of feature vector differences (see Equation 5.3). We first generate a set of trajectories using the predictive method and compute the corresponding feature vectors for these. For a chosen parameter set we generate the same paths using the social force model and compute the feature vectors. We then take the mean of the norm of the feature vector differences. We compute the parameters that minimize this quantity using a gradient descent method.

5.4 Paths for Situations with One Agent

To test the learned policy, we first generate paths for single agents moving towards their targets. We compare the generated paths to paths created using the social force model and to paths generated using the predictive method. As there are no other agents present, these paths only involve the attractive potential. We learn the attractive potential by generating over 6000 training points.

To test the attractive potential, we sample a set of start positions and velocities and target positions. For each of these states we generate a path using the learned policy, the social force model and the predictive method. A selection of such paths generated by all three methods is shown in Figure 5.5. It suggests that regardless of the starting velocity the agents reach their targets. Furthermore, the paths created using the learned policy are similar to the paths created by the predictive method. The paths of the social force method do not coincide with the predictive paths. We generate a set of 1000 random states for which we generate paths using each of the three methods. For all the paths generated we compute the corresponding feature vector (see Chapter 3.4). Figure

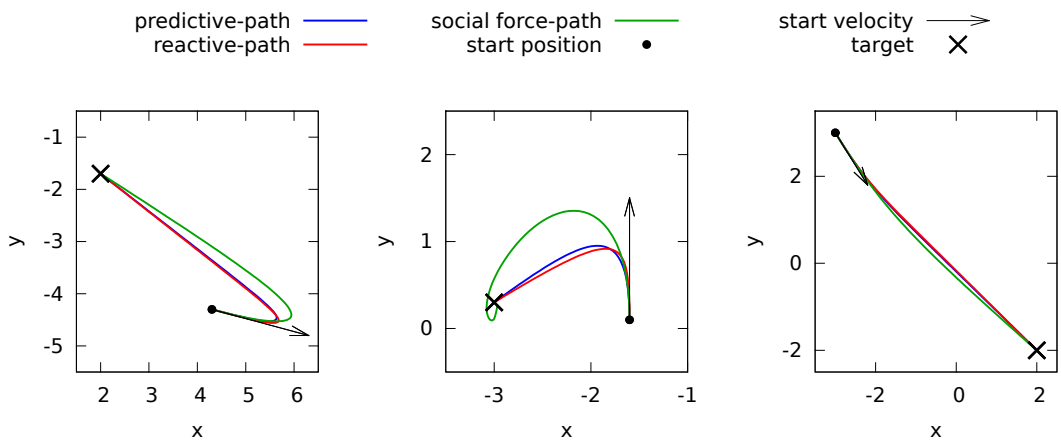

Figure 5.5: Multiple paths of agents traveling towards a target generated using the learned policy (red), the predictive method (blue) and the social force method (green). In addition to the starting positions, the targets and the starting velocities are shown.

5.6 [left] shows the mean of the individual feature vector over all 1000 paths and for all three methods. The features shown in Figure 5.6 [left] are acceleration (f_{acc}), travel time (f_{time}) and velocity (f_{vel}). The last quantity shown in the plot is the mean of the norm of the feature vector difference (see Equation 5.3). If this value is low the paths generated using this method and the paths optimized using the predictive method are similar with respect to the features, meaning both paths show the same characteristics. Figure 5.6 [right] shows a box plot (see Chapter 3.5) of the individual norms of the feature vector differences. According to the features the learned policy generates paths which are more similar to the predictive method than the paths generated by the social force method.

5.5 Paths for Situations with Two Agents

In the experiment described in this section we generate paths for situations with two agents. In contrast to paths of single agents (see Chapter 5.4), the agents additionally account for evading each other while moving towards the target. This evasion maneuver is modeled by the commands of the repulsive potential. To learn the repulsive potential, we generate 1.7 million acceleration commands as training data. We choose a distance between the agents and their targets of d_{target} =7m, and we fix the desired velocity of the second agent to $v_{\text{desired}} = 0.5\frac{\text{m}}{\text{s}}$. Figure 5.7 shows three example situations at different points in time. We notice that the predictive and the reactive method create similar

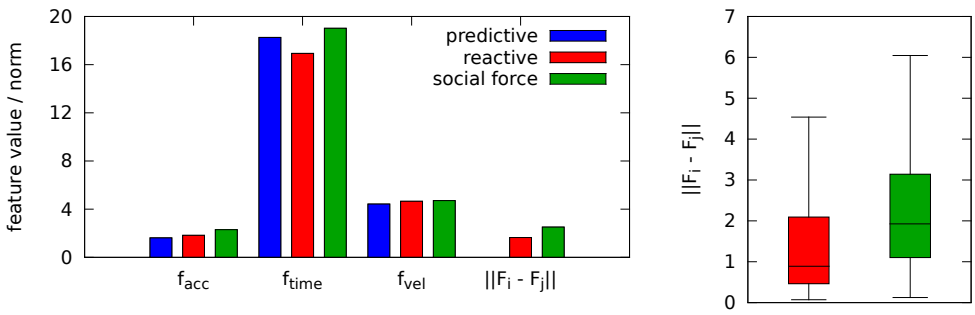

Figure 5.6: Left: Mean feature values and mean of the norm of the feature vector differences of 1000 single agent paths generated using the different methods. The first three groups of bars shows the mean features (acceleration (f_{acc}), travel time (f_{time}) and velocity (f_{vel})), the last group shows the mean of the norm of the feature vector differences ($\|\mathbf{F}_i - \mathbf{F}_j\|$) (see Equation 5.3). Right: Box plot of the norm of the feature vector differences between 1000 single agent paths generated using the reactive method (red) respectively the social force method (green) and single agent paths generated using the predictive method.

paths whereas the social force method results in paths that do not resemble the predictive paths well. In the first row of Figure 5.7 we see that the reactive method generates paths which start to evade from the beginning of the path and thus generates smooth behavior. The social force method also does not create collisions, but unnatural paths. The feature values (acceleration (f_{acc}), distance between agents (f_{dist}), travel time (f_{time}) and velocity (f_{vel})) together with the norm of the feature vector difference ($\|\mathbf{F}_i - \mathbf{F}_j\|$) for the individual paths are shown in Figure 5.8. We see that the social force method has a high value for the travel time feature (f_{time}), especially for the first and third situation. This is because the social force method generates lower velocities (see f_{vel}) and detours (see Figure 5.7). The dissimilarity between the paths generated using the social force method and the paths optimized by the predictive method leads to high values of the norm of feature vector differences.

Figure 5.9 visualizes the commands generated by the two potentials together with the agents' velocities. The plot shows the same situation as in the first row of Figure 5.7. The commands are plotted at the position where the agent is located at this point in time. We see that in the beginning the repulsive potential (green arrows) pushes the agents apart. After the agents passed each other this command vanishes. This is due to the fact that agents do not evade each other anymore once they have passed and travel in opposite directions (see also Figure 5.2 [left]). The red arrows visualize the attractive potential. In the beginning this potential pushes the agent directly towards

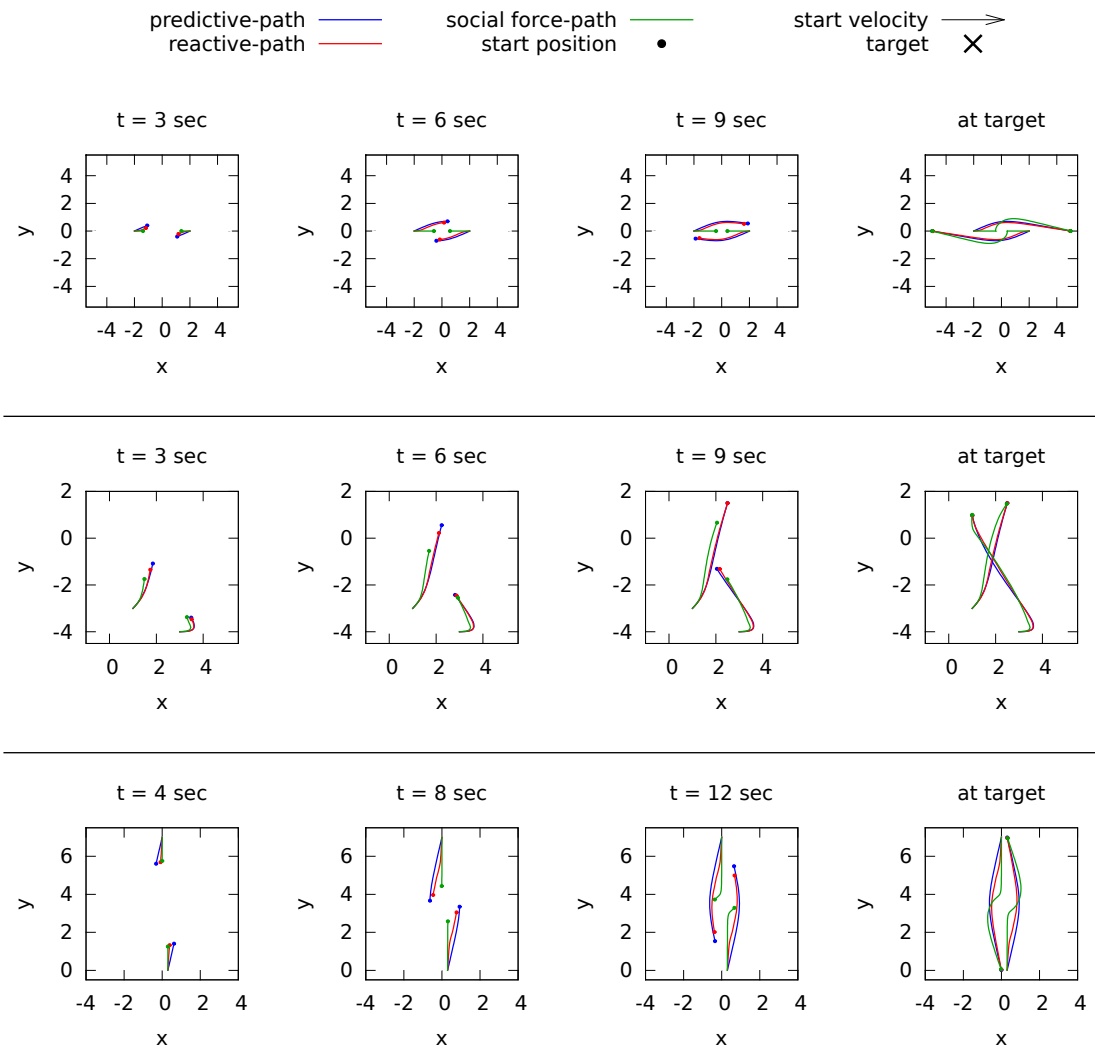

Figure 5.7: Three examples of paths for two agents at different points in time, generated using the three different methods. The paths generated by the learned policy (red) and the predictive paths (blue) are similar, whereas the social force method (green) creates unnatural paths (see first and third row).

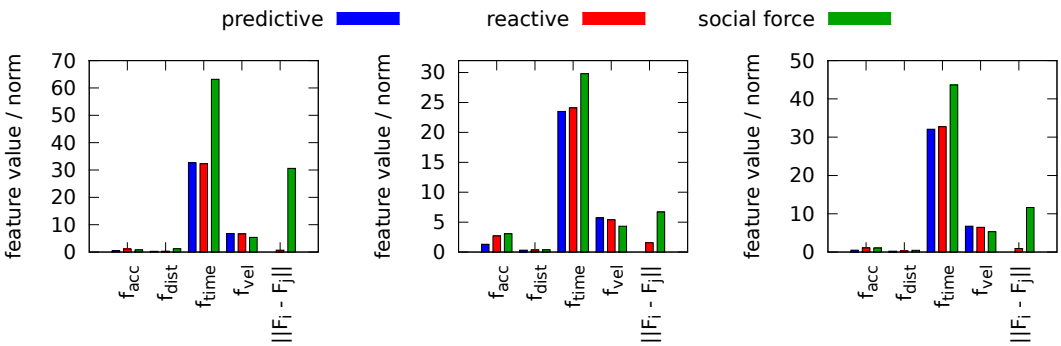

Figure 5.8: Feature vector values (acceleration (f_{acc}), distance between agents (f_{dist}), travel time (f_{time}) and velocity (f_{vel})) together with the norm of the feature vector difference ($||\mathbf{F}_i - \mathbf{F}_j||$) of the paths shown in Figure 5.7. Note that a much higher value of f_{time} for the paths created using the social force method leads to high values of $||\mathbf{F}_i - \mathbf{F}_j||$, meaning that these paths does not show the same characteristics as the paths optimized by the predictive method.

their targets, because the agents start with zero velocities. After the agents reached their desired velocities the attractive potential tries to drive the agents back on their paths. After the repulsive potential vanished the attractive potential pushes the velocity back into direction towards the target.

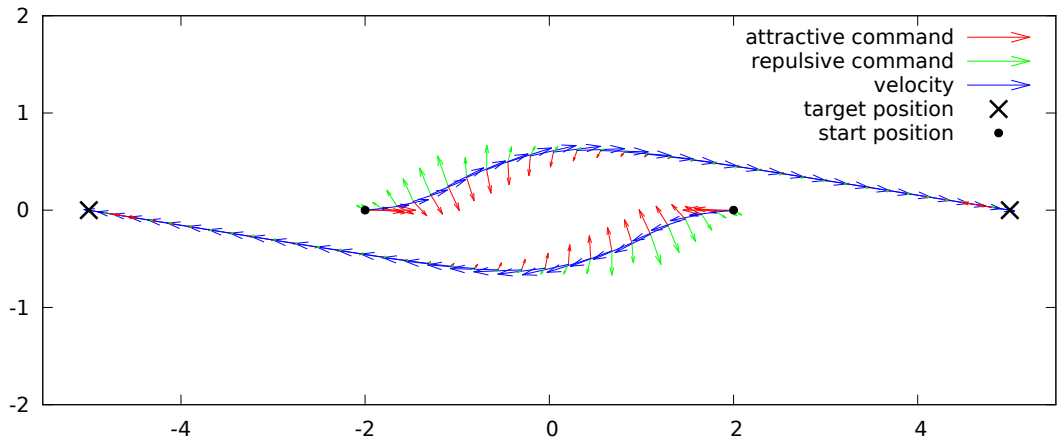

Figure 5.9: This plot shows commands of the attractive potential (red), the repulsive potential (green) and velocities (blue) of the agents at different points in time for the situation shown in the first row of Figure 5.7. The commands and velocities are plotted at the position the agents is located at this point in time.

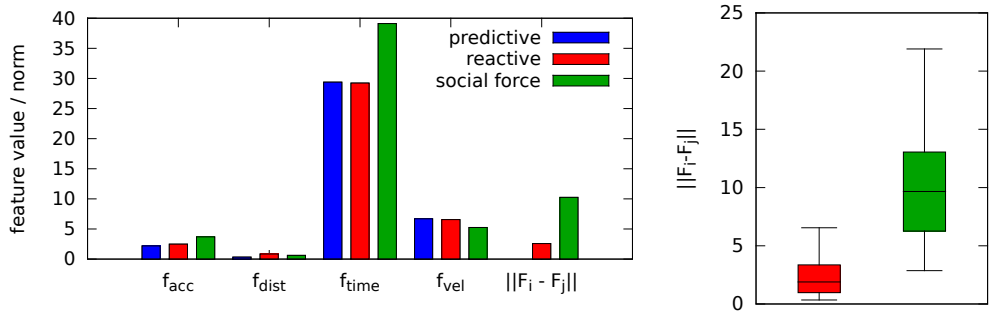

Figure 5.10: Left: Mean of the feature vector values and mean of the norm of the feature vector differences of 1000 paths involving two agents, plotted for the different methods. The first four groups of bars show the mean of the feature vector values (acceleration (f_{acc}), distance between agents (f_{dist}), travel time (f_{time}) and velocity (f_{vel})), the last group shows the mean of the norm of the feature vector differences. Right: Box plot of the norm of feature vector differences between the costs of 1000 paths generated using the reactive method (red) respectively the social force method (green) and paths generated using the predictive method.

To test the proposed algorithm for generating paths for two agents, we generate a set of 1000 situations with two agents. For this experiment we first generate paths for one agent using the predictive method as in the experiment with a single agent (see Chapter 5.4). From this set of paths we then choose a set of combinations of two paths, where the agents at start and target have a defined minimum distance and get close or cross each other on their paths. The first condition generates situations which are possible within real situations, because pedestrians have some extension in space they cannot stand on the same position. The second condition forces situations where avoiding the other agent is necessary. Figure 5.10 [left] shows the mean of the individual feature values and the mean of the norm of the feature vector difference. The reactive method generates paths with almost the same feature values as the predictive method. The social force method generate high values for the travel time because it often generates detours. Figure 5.10 [right] shows a box plot of the norm of feature vector differences between the paths generated by the predictive method and the paths generated by the other two methods (reactive (red) and social force method (green)). We see that the social force model generates paths with a higher value for the norm of the feature vector difference compared to the learned policy. This experiment suggests that our method is able to better reproduce the paths of the predictive method compared to the social force method.

5.6 Paths for a Situation with Multiple Agents

In this thesis we learn an attractive potential together with a repulsive potential for two agents to drive an agent towards a target while avoid other agents. We assume that the avoidance of more than one agent can be approximated by the sum of acceleration commands to avoid each agent individually. In this section we discus how our method performs in a symmetric situation with five agents. Figure 5.11 shows the paths generated for this situation. We can see that the predictive method simplifies the situation as it accelerates one agent, so that the evasion for the other agents gets easier. The paths generated by the predictive method result from a joint prediction at the start of this situation. The learned policy as well as the social force method generates their paths by moving each agent, in each time step individually. Figure 5.11 shows that the learned policy creates a similar set of paths as the predictive method, but does not show this special behavior of accelerating one agent. The social force method first drives all agents towards their targets until the agents have a defined range. After this the method needs around 34 seconds until it drives the agents in a circular path towards their targets. Figure 5.12 [left] illustrate the costs (sum of weighted feature vector values) which are optimized in the predictive method for this situation. We see that the learned policy and the reactive method generates paths with similar costs, while the social force method generates much higher costs. Figure 5.12 [right] shows that this high value comes from the fact that the social force method creates a high value of travel time, because the agents stop for some time (see Figure 5.11 between second eight and around second 42). Figure 5.13 visualizes commands of the learned policy generated from the different potentials together with the velocities of the agents at different points in time. We see that on the first half of the agents' paths the agents are pushed apart (repulsive potential shown in green). After the agents crossed the inner circle the repulsive potentials vanishes and the agents drive towards their targets. The simulation of these paths with ten commands per second needed less than one second using the social force method, around 2:29 minutes using the reactive method and around 2:20 hours using the predictive method. The simulation of these paths we done using a computer with an Intel i7 quad-core (2.2Ghz). The paths of the predictive method are created by one prediction at the start of the simulation, while the other methods generate commands in every time step. Note that to create this plot using the learned policy we compute commands for all five agents. If we plan to create this situation with a robot and people walking we only need to compute commands for the robot resulting in a computation time that allows real time execution (see also Figure 5.24).

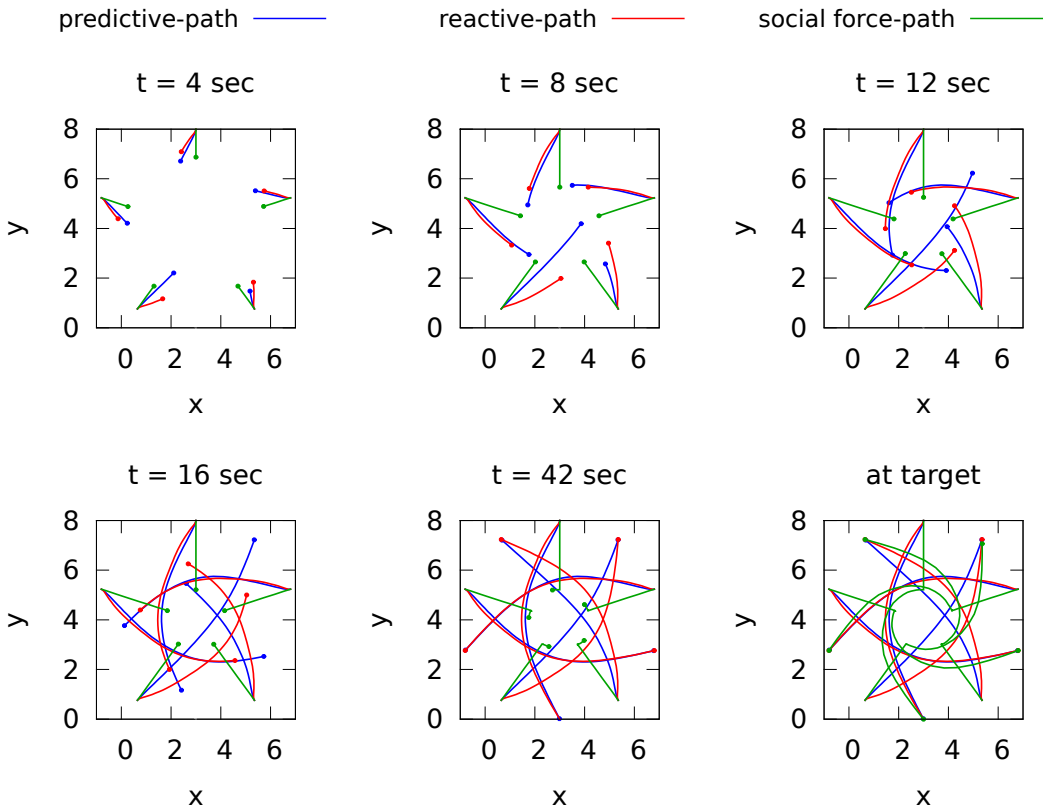

Figure 5.11: Paths of five agents starting on a circular setting, shown at different points in time. Note that the social force method generates paths which need additional time to determine how to evade (no movement between second eight and second 42).

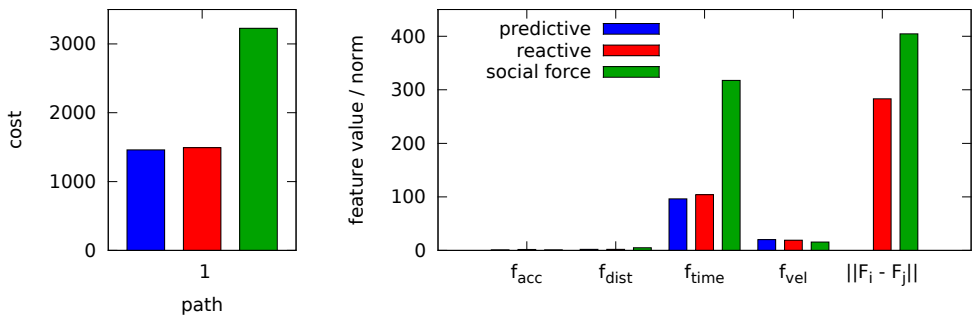

Figure 5.12: Left: Costs of the paths shown in Figure 5.11 generated by the predictive method (blue), the reactive method (red) and the social force method (green). Right: Features and norm of the feature vector differences for these paths. The first four groups of bars show the feature values (acceleration (f_{acc}), distance between agents (f_{dist}), travel time (f_{time}) and velocity (f_{vel})), the last group shows the norm of the feature vector differences ($||\mathbf{F}_i - \mathbf{F}_j||$) (see Equation 5.3).

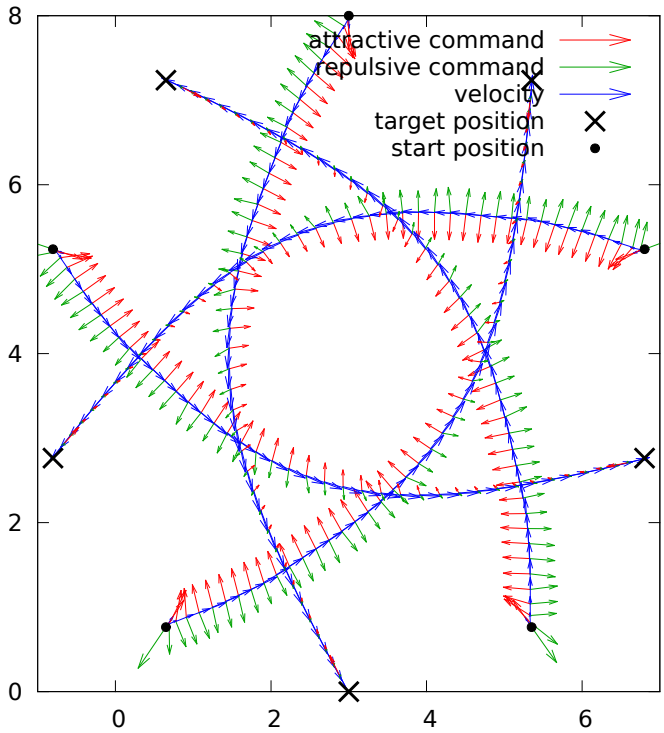

Figure 5.13: This plot shows commands of the attractive (red) and repulsive potential (green) together with velocities (blue) for the five agents situation shown in Figure 5.11. The commands and the velocity are plotted at the position the agent is located at this point in time.

5.7 Experiments on a Real Robot

In this chapter we describe the experimental setup and show the results of the experiments with a real robot. To evaluate the performance of the proposed method on a robot we use our method to navigate a Pioneer III robot in the presence of pedestrians. A graphical overview over the experimental setup is shown in Figure 5.14. We use a motion capture system to determine the position and velocity of the robot and all involved people. Using the position and velocity of the robot we create a command from the attractive potential. We then use the positions and velocities of all other agents to generate a repulsive command for each agent. The sum of all these commands is then used to compute a desired velocity for the robot, which is sent to the controller. The controller converts this desired velocity into a linear and an angular velocity changing the robot's current velocity towards the desired velocity.

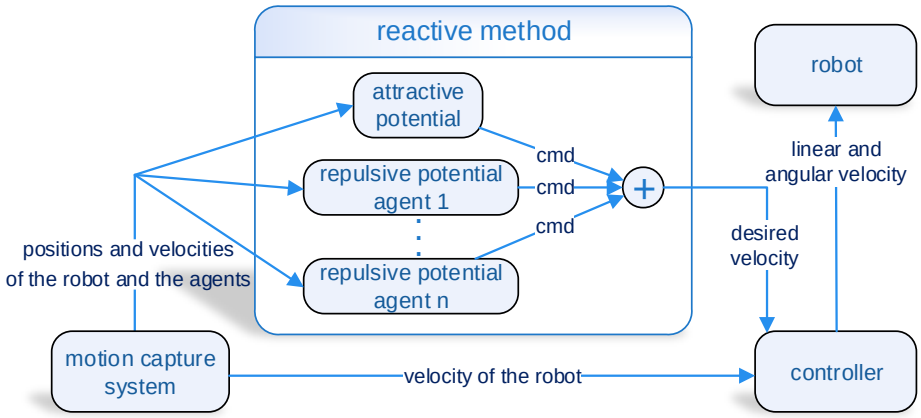

Figure 5.14: Overview over the setup for the experiments using a real robot. We use a motion capture system to determine the position and velocity of the robot and all pedestrians. We then generate commands from the attractive and repulsive potential. From these we compute a new desired velocity, which is sent to the controller. The controller converts this velocity into a linear and an angular velocity driving the robots current velocity towards the desired velocity.

To test the learned policy on a real robot in presence of pedestrians we equip people with helmets which can be tracked by the motion capture system. We first test the behavior of the robot when the second agent does not evade the robot. The resulting path of the person and the robot is shown in Figure 5.15. We see that the robot evades the person. After they cross each other the robot drives straight towards its target. The oscillating position of the person's path is caused by the left and right movement of the head of a person walking on a straight line.

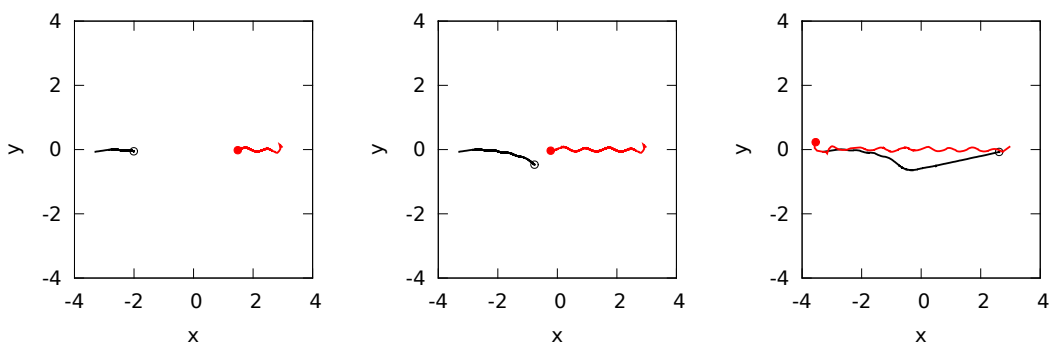

Figure 5.15: Path of the robot (black) evading a person (red) which walks a straight line.

In the next experiment we test the robustness of the evasion behavior of the robot. A person first walks as he would cross the robot on one side. After the robot started to evade the person the human crosses the robot on the other side. The robot is able to notice this change and react accordingly. The paths of the person and the robot are shown in Figure 5.16.

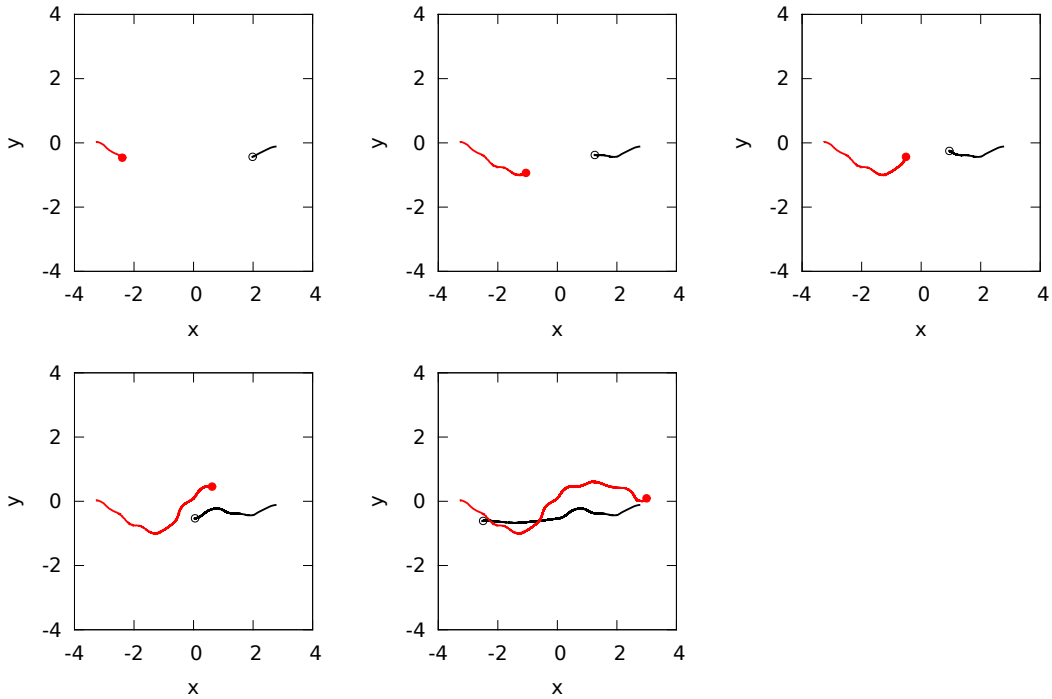

Figure 5.16: Path of the robot (black) evading a person (red) changing the side he wants to cross the robot.

Furthermore, we conduct experiments with two persons. We first test the behavior of the robot when the persons walk straight towards the robot. The behavior of the robot depends on the distance between the agents. If the persons walk close side by side the robot evades both persons. This situation is shown in Figure 5.17. If the persons leave a slightly bigger gap between each other, the robot is able to pass between the people, as shown in Figure 5.18.

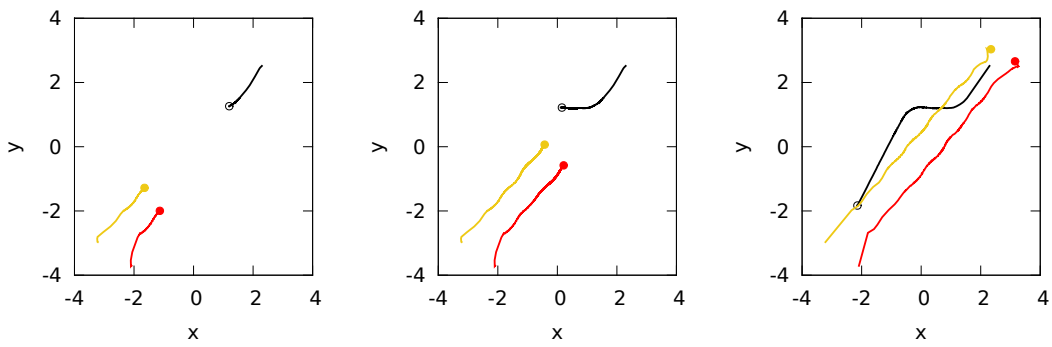

Figure 5.17: Two persons walking close side by side. The robot (black) drives around them.

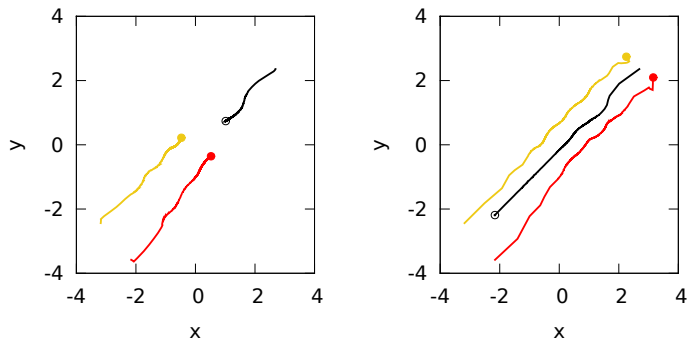

Figure 5.18: A similar situation as in Figure 5.17, but the persons leave a slightly bigger gap between each other so the robot (black) is able to pass between the persons.

The next experiment evaluates the behavior of the robot when people are crossing its path. We set up an experiment where two people cross the robot's path at the same time. The first of the resulting situations is shown in Figure 5.19. In this situation the persons temporarily blocking the robot's path. The robot decelerates and drives with a lower velocity until its path is free again. Figure 5.20 shows a similar situation. The persons meet each other in a slightly different way, so the robot is able to drive around them.

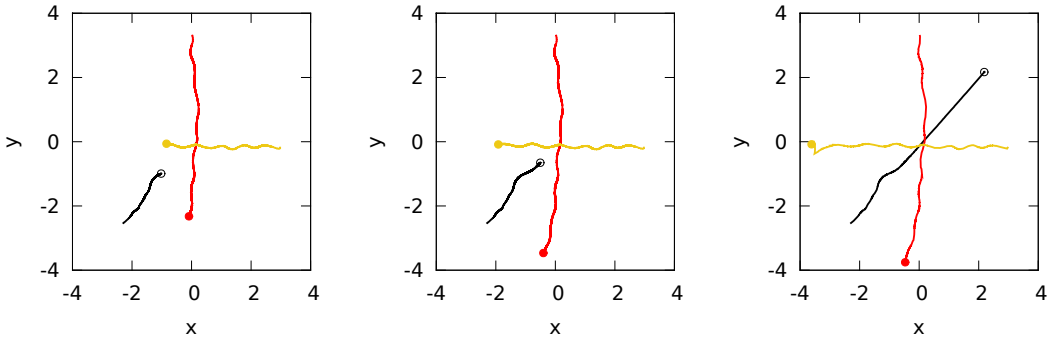

Figure 5.19: The path of the robot (black) is blocked. The robot decelerates until its direct path towards its target is free again.

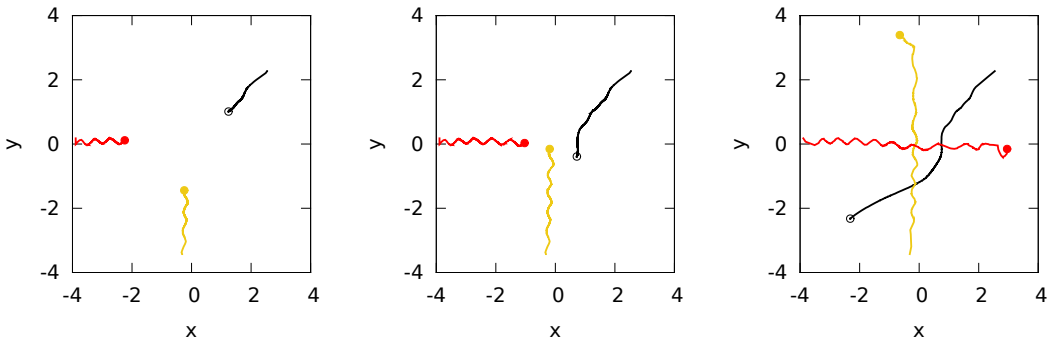

Figure 5.20: This plot shows a similar situation as in Figure 5.19 but the robot (black) is able to evade the persons.

As mentioned before, the computation-time of the predictive method grows exponentially with the number of involved agents. Using a reactive approach we are able to reduce the computation time on execution so that we can simulate a larger number of agents within reasonable time. In the following we show some experimental results for situations including four people and the robot. We first repeat the experiments shown in Figure 5.17 and Figure 5.18. The situations with four agents are shown in Figure 5.21 and Figure 5.22. The robot is able to create similar paths as in the experiments involving two persons and the robot.

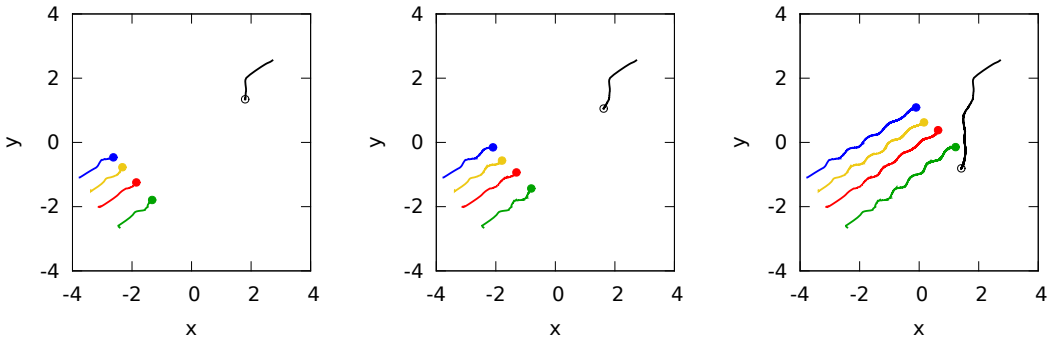

Figure 5.21: This plot shows a similar situation as in Figure 5.17, but with four agents. The robot (black) takes the path around the people as he has no space to drive through.

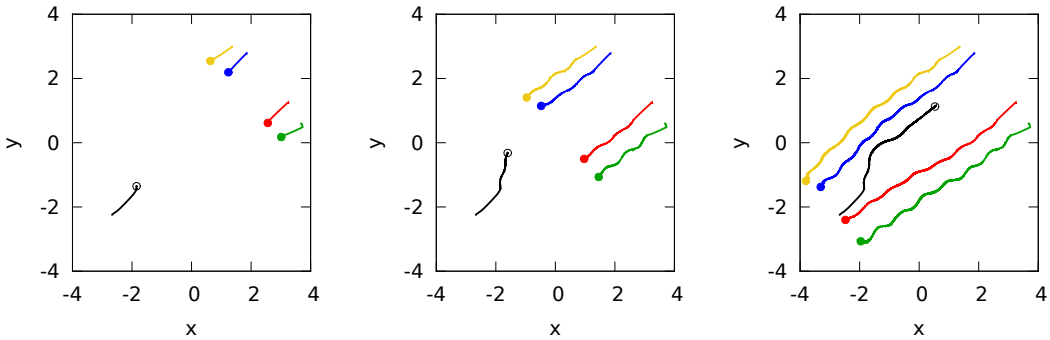

Figure 5.22: As in the two agent case (see Figure 5.18) the robot (black) drives through the gap between the persons.

Furthermore, we set up a situation which violates the assumptions made when reducing the state space of the repulsive potential (see Chapter 4.1.2.2). In the reduction we assume that all other agents travel with a fixed velocity which we set to $0.5\frac{\mathrm{m}}{\mathrm{s}}$. We set up a situation where four humans stand still. The situation together with the resulting path of the robot (black) and the simulated path using the learned policy (blue) is shown in Figure 5.23. The path of the robot and the simulated path are mostly dissimilar in the beginning, as the simulation does not takes into account the orientation of the robot at start. Despite of the fact that we violate the assumptions, the robot creates a reasonable path around all of the pedestrians. As the head of a person who stands still moves slightly and the motion capture system is able to capture even movements of millimeters, we are able to compute a direction of velocity for each person. These small movements leads to the fact that the algorithm assumes target positions which are located at a circle of 7m radius around the individual persons (see Chapter 4.1.2.2). Through the movement of the heads the targets of the pedestrians will fast move around and lead to the wiggly path of the robot.

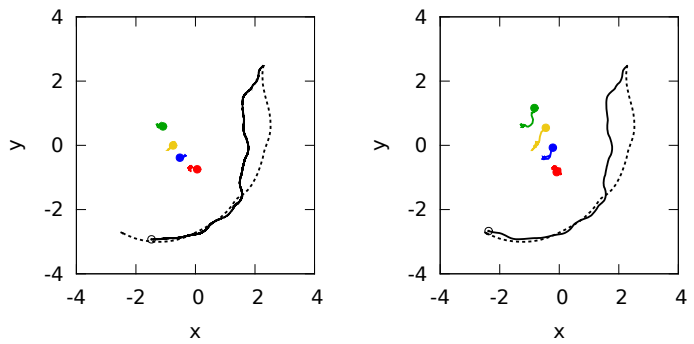

Figure 5.23: The robot (black) drives around four waiting persons. The path simulated using the learned policy is shown drawn with a dashed line. This path does not take into account the orientation of the robot at start. The situation shown here is particularly interesting as this violates the assumption that other agents travels at a fixed velocity.

In the simulation experiments we set up a situation where all agents are located on a circle and their targets lies on the opposite side of this circle (see Figure 5.11). We try to recreate this initial situation in the experiments using the robot. This situation is shown in Figure 5.24. The robot starts to drive towards its target, but after all people accumulate in the center the robot decides to take the path around them. After the people walked on, the direct path to the robot's target is free and the robot drives direclty towards its target. In this experiment the robot is able to compute new desired velocities at a rate of 14Hz.

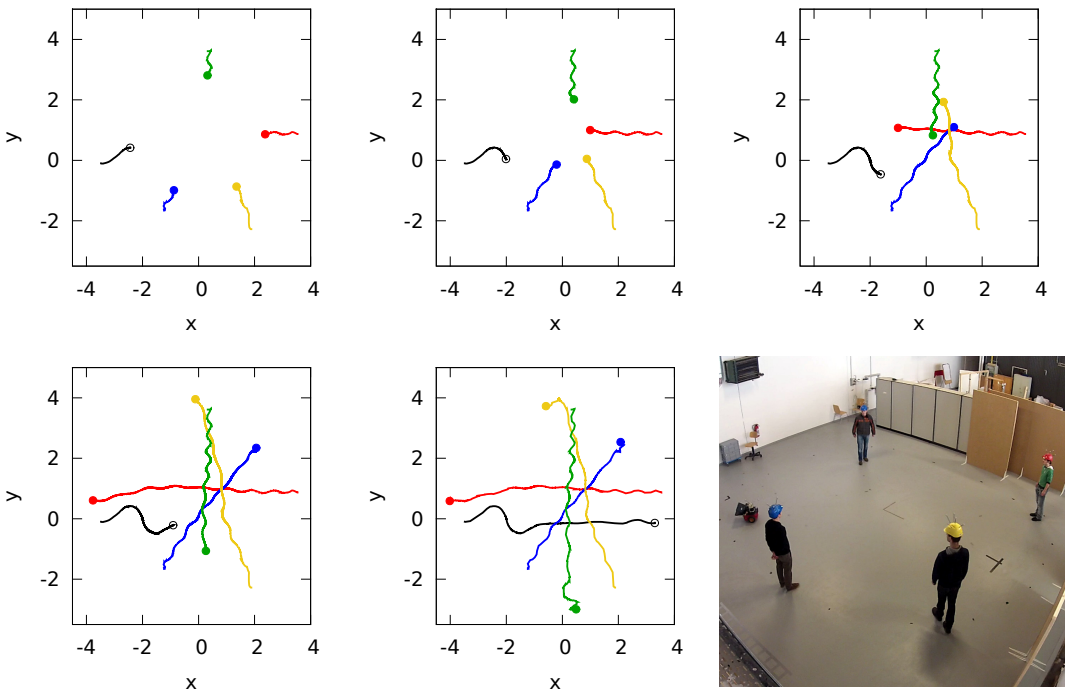

Figure 5.24: All agents want to reach the opposite side of the room. The robot (black) first tries to drive around the persons. After the direct path from the robot's position to its target is free again, the robot drives straight towards its target. The image shows an image of this situation. The robot is located at the left side of the image.

5.8 Computation-times

In this experiment we take a look at the computation-time of the different methods as this is one of the main motivations for this thesis. To drive a robot we need to compute at each time step a desired velocity. To compute a velocity using the reactive or social force method we first evaluate the attractive potential and add the contributions of each of the other agents from the repulsive potential. Using these acceleration commands we update the desired velocity of the robot. To compute this velocity using the predictive method we need to generate a trajectory for each of the involved agents. As the other agents do not necessarily drive or walk the path predicted, the paths must be generated in each time step. Figure 5.25 shows box plots of the computation-times to compute such velocities for 1000 random situations. We can see that the computation-time per velocity computation for the predictive method grows exponentially with the number of agents (see Figure 5.25 [left]). Figure 5.25 [right] shows that the computation-time for the reactive method grows linear with the number of agents. The social force method evaluates closed-form functions to compute the velocity. This computation needs, for situations with six or less agents we investigate, less than one millisecond. Note that the scales of the y-axis in Figure 5.25 are noticeably different. The optimization of the predictive paths stops if the gradient of the error function is less than a predefined value (in this experiments we set this value to 10^{-4}). The optimization is also stopped if the computation time excites a certain amount of iterations. This stop was triggered frequently. This means the real computation-time, until all gradient values are lower than the predefined threshold, is possibly even worse.

5.9 The Influence of the Assumptions on the Generated Paths

In this experiment we investigate the error, which we introduce by the assumptions we make to reduce the state space used to parametrize the potentials (see Chapter 4.1.2.2). While we test the error introduced by these assumptions, we also investigate the amount of error we introduce by using the regression to predict acceleration commands for arbitrary situations. We then discuss a situation where the use of the reduced state space leads to a suboptimal path.

To be able to generate commands with and without the use of the assumptions and to separate the error introduced by the assumptions and by the regression, we replace the generation of the commands. We replace the generation of commands using the regression by directly generating the commands using the predictive method. To incorporate

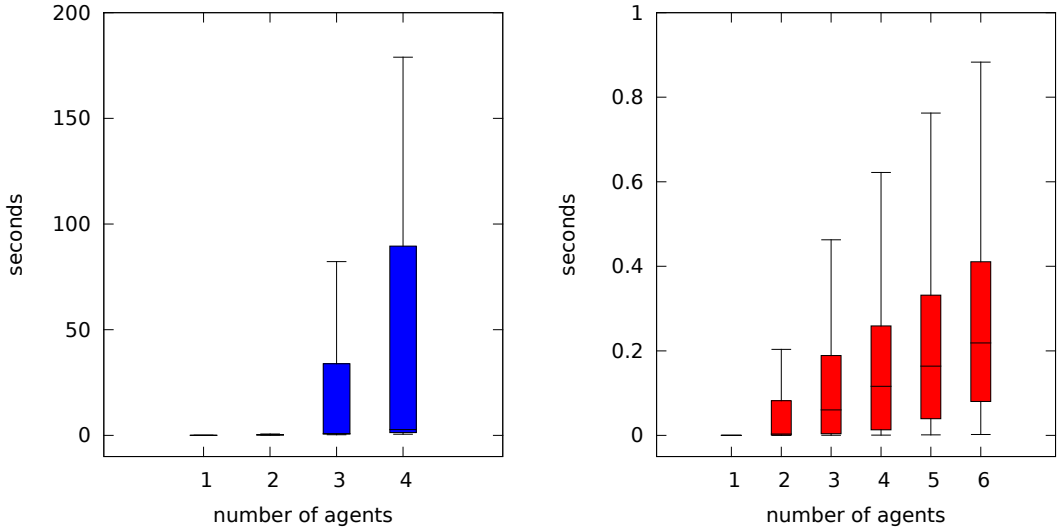

Figure 5.25: Box plot of the time needed to compute a velocity command to drive a robot. The predictive method (blue) shows an exponential growth with the number of agents, while the reactive method (red) shows a linear growth.

the assumptions into the predictive method we move and turn the positions, velocities and targets of the agents so that the resulting state meets the assumptions (see Chapter 4.1.2.3).

For this experiment we generate a set of 100 random situations with two agents, where both agents need to evade each other (as described in Chapter 5.5). We then simulate paths for each agent by generating the acceleration commands directly using the predictive method. We generate the paths with and without the use of the assumptions. We also generate paths for the same situations using the regression. As the regression is parametrized using the reduced state space these paths include the influence of the assumptions as well as the influence of the regression method. For each of the generated situations, we compute two norms of feature vector differences. First, we compute the differences between the paths generated without assumptions and the paths generated with the use of the assumptions but without regression. We then compute the differences between the paths generated without assumptions and the paths generated using the regression. Figure 5.26 shows a box plot of these norms of feature vector differences. Note that the error between the paths generated without assumptions and the paths generated using the regression (shown in green) includes errors introduced by the assumptions as well as introduced by the regression. As a result of this experiment we can conclude that the error which is introduced by reducing the state space with use of

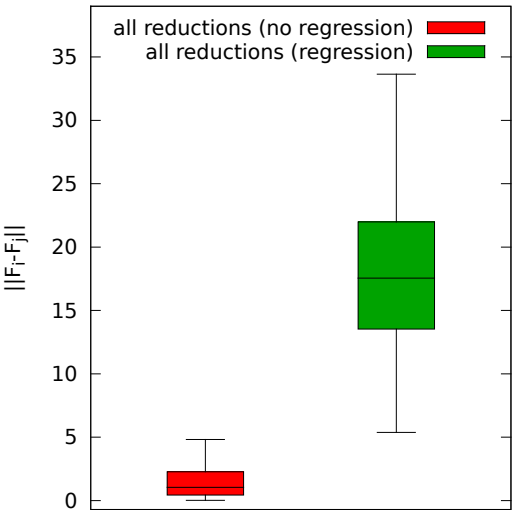

Figure 5.26: A box plot of the errors introduced by the assumptions and introduced by the regression. The red part shows the norms of feature vector difference between the paths created without the assumptions and paths generated with the assumptions but without regression. These differences include the error introduced by the assumptions. The green part of the plot shows the norms of feature vector difference between the paths created without the assumptions and the paths created using the regression which additionally include errors introduced by the use of the regression.

the assumptions is smaller then the error introduced by the regression we use in this thesis.

Figure 5.27 shows a situations where the use of the assumptions leads to a suboptimal path executed on the robot. In this situation the pedestrian (red) walks really slow on the robot's direct path to its target. The robot (black) assumes that the pedestrian walks with a fixed desired velocity of $0.5\frac{\mathrm{m}}{\mathrm{s}}$. This assumption leads to the fact that the robot does not consider to move around the human, as the learned policy assumes that the pedestrian move at least with the same velocity as the robot. In this experiment the assumption of a fixed velocity of other agents leads to the fact that the robot does not overtake the pedestrian. As a result the robot drives into the human's legs.

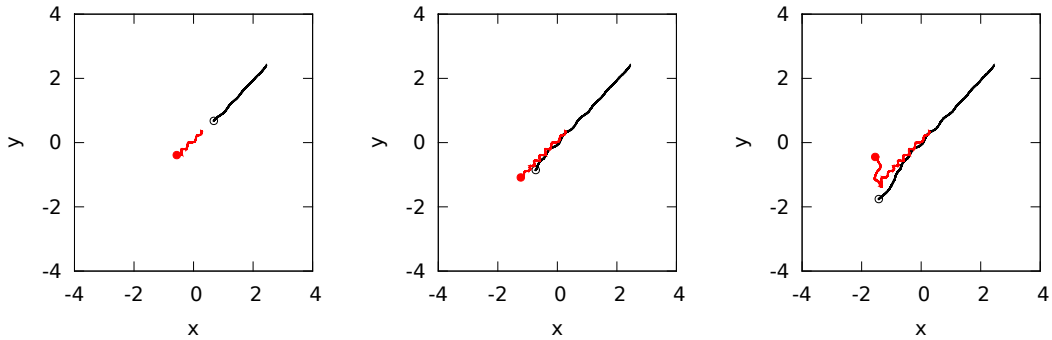

Figure 5.27: This plot shows a situation where the assumptions, used to reduce the state space used to represent the repulsive potential, lead to a suboptimal path driven by the robot. The robot (black) is not able to overtake the slowly moving pedestrian (red).

6 Discussion

We start this chapter with a short overview of our algorithm and outline the contribution of this thesis. Then, we take a look at some possible improvements of the proposed algorithm.

6.1 Conclusion

In this thesis we present a reactive method to create human-like paths for social navigation. We use locally weighted regression together with training data, created using a predictive path planning method proposed by Kuderer et al., to learn a reactive navigation policy. We reduce the state spaces, used to represent the potentials, in multiple steps. First, we split the potential driving an agent towards its target while evading other agents into two individual potentials. The first of these potentials is an attractive potential that moves a robot towards its target. The second potential is a repulsive potential that drives the robot away from other agents. We further reduce the state space by use of symmetries and additional assumptions. In the experiments we demonstrate that, despite of these reductions, the learned policy is able to reproduce the paths generated by the predictive method. Furthermore, we demonstrate that we are able to approximate the potential evading multiple agents with the sum of potentials evading each agent individually. We also demonstrate that the algorithm scales linearly with the number of agents, while the predictive method scales exponentially. In the experiments with a real robot we demonstrate that the proposed method can be used to navigate a mobile robot successfully through an environment with multiple pedestrians. The method proposed in this thesis combines successfully a reactive navigation method with training data of a predictive method. Based on the usage of a non-parametric regression we are able to represent the human-like behavior of the training data. The method we present in this thesis scales better with the number of agents than the predictive method. In contrast to paths created by the social force model the learned policy creates human-like paths. Our learned policy can be used to navigate a mobile robot through environments with multiple agents while creating human-like paths.

6.2 Future Work

In this section we present some points where the proposed algorithm can be enhanced. To be able to use locally weighted regression efficiently we reduce the state space used to represent the repulsive potential from twelve to five dimensions. This reduction leads in some cases to suboptimal paths. For example, a robot cannot overtake a slowly moving pedestrian on the same path. This is due to the fact that our algorithm assumes that all other agents move with a fixed velocity. This problem can be tackled by undoing this part of the reduction. Note that this adds an additional dimension to the state space of the repulsive potential. The predictive path planning algorithm, which we use to create training data, was recently extended to account for walls and obstacles. By adding a global map to the predictive method the generated paths are no longer invariant to movements and rotations of the coordinate system. In order to incorporate obstacles and walls into our proposed method policy, we can continue to assume that this invariance holds and add additional repulsive potentials representing the avoidance of obstacles and walls.

Bibliography

[1] R. Amit and M. Matari. Learning movement sequences from demonstration. In *Development and Learning, 2002. Proceedings. The 2nd International Conference on*. IEEE, 2002.

[2] G. Arechavaleta, J.P. Laumond, H. Hicheur, and A. Berthoz. An optimality principle governing human walking. *Robotics, IEEE Transactions on*, 24(1), 2008.

[3] B.D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5), 2009.

[4] R.C. Arkin. Integrating behavioral, perceptual, and world knowledge in reactive navigation. *Robotics and autonomous systems*, 6(1), 1990.

[5] C.G. Atkeson, A.W. Moore, and S. Schaal. Locally weighted learning. *Artificial intelligence review*, 11(1), 1997.

[6] D.C. Bentivegna and C.G. Atkeson. Learning from observation using primitives. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 2. IEEE, 2001.

[7] C.M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., 2006.

[8] M. Blokzijl-Zanker and Y. Demiris. Multi robot learning by demonstration. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[9] J. Bohren, R.B. Rusu, E.G. Jones, E. Marder-Eppstein, C. Pantofaru, M. Wise, L. Mosenlechner, W. Meeussen, and S. Holzer. Towards autonomous robotic butlers: Lessons learned with the pr2. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011.

[10] S. Bouraine, T. Fraichard, and H. Salhi. Provably safe navigation for mobile robots with limited field-of-views in unknown dynamic environments. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012.

[11] D.C. Brogan and N.L. Johnson. Realistic human walking paths. In *Computer Animation and Social Agents, 2003. 16th International Conference on*. IEEE, 2003.

[12] P. Brunn. Robot collision avoidance. *Industrial Robot: An International Journal*, 23 (1), 1996.

[13] W.S. Cleveland and S.J. Devlin. Locally weighted regression: an approach to regression analysis by local fitting. *Journal of the American Statistical Association*, 83 (403), 1988.

[14] N.E. Du Toit and J.W. Burdick. Robot motion planning in dynamic, uncertain environments. *Robotics, IEEE Transactions on*, 28(1), 2012.

[15] P. Fiorini and Z. Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7), 1998.

[16] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *Robotics & Automation Magazine, IEEE*, 4(1), 1997.

[17] D. Fox, W. Burgard, S. Thrun, and A.B. Cremers. A hybrid collision avoidance method for mobile robots. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 2. IEEE, 1998.

[18] S.S. Ge and Y.J. Cui. New potential functions for mobile robot path planning. *Robotics and Automation, IEEE Transactions on*, 16(5), 2000.

[19] S.J. Guy, M.C. Lin, and D. Manocha. Modeling collision avoidance behavior for virtual humans. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[20] H. Haddad, M. Khatib, S. Lacroix, and R. Chatila. Reactive navigation in outdoor environments using potential fields. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 2. IEEE, 1998.

[21] D. Helbing and A. Johansson. *Pedestrian, Crowd and Evacuation Dynamics*. Swiss Federal Institute of Technology, 2009.

[22] D. Helbing and P. Molnár. Social force model for pedestrian dynamics. *Phys. Rev. E*, 51(5), 1995.

[23] A. Johansson, D. Helbing, and K.S. Pradyumn. Specification of the social force pedestrian model by evolutionary adjustment to video tracking data. *Advances in Complex Systems*, 10(supp02), 2007.

[24] I. Karamouzas, P. Heil, P. van Beek, and M. Overmars. A predictive collision avoidance model for pedestrian simulation. *Motion in Games*, 2009.

[25] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto. Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, 31(3), 2012.

[26] V. Koropouli, D. Lee, and S. Hirche. Learning interaction control policies by demonstration. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011.

[27] M. Kuderer, H. Kretzschmar, C. Sprunk, and W. Burgard. Feature-based prediction of trajectories for socially compliant navigation. In *Proc. of Robotics: Science and Systems*. RSS, 2012.

[28] S. Li, Z. Shen, and G. Xiong. A k-nearest neighbor locally weighted regression method for short-term traffic flow forecasting. In *Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on*. IEEE, 2012.

[29] H.I. Lin and C.C. Lai. Robot reaching movement synthesis by human demonstration. In *Computer Science & Education (ICCSE), 2012 7th International Conference on*. IEEE, 2012.

[30] J. Maitin-Shepard, M. Cusumano-Towner, J. Lei, and P. Abbeel. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010.

[31] M. Matsushima, T. Hashimoto, M. Takeuchi, and F. Miyazaki. A learning approach to robotic table tennis. *Robotics, IEEE Transactions on*, 21(4), 2005.

[32] J. Minguez, L. Montano, and J. Santos-Victor. Reactive navigation for non-holonomic robots using the ego-kinematic space. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 3. IEEE, 2002.

[33] J. Müller, C. Stachniss, K.O. Arras, and W. Burgard. Socially inspired motion planning for mobile robots in populated environments. In *Proc. of International Conference on Cognitive Systems*. Paco plus, 2008.

[34] S. Paris, J. Pettré, and S. Donikian. Pedestrian reactive navigation for crowd simulation: A predictive approach. *Computer Graphics Forum*, 26(3), 2007.

[35] D.A. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1), 1991.

[36] N.D. Ratliff, J.A. Bagnell, and M.A. Zinkevich. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006.

[37] J.M. Roberts, E.S. Duff, P.I. Corke, P. Sikka, G.J. Winstanley, and J. Cunningham. Autonomous control of underground mining vehicles using reactive navigation. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 4. IEEE, 2000.

[38] A. Saxena, L. Wong, M. Quigley, and A.Y. Ng. A vision-based aystem for grasping novel objects in cluttered environments. In *Robotics Research*. Springer, 2011.

[39] S. Schaal, C.G. Atkeson, and S. Vijayakumar. Real-time robot learning with locally weighted statistical learning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1. IEEE, 2000.

[40] B.D. Ziebart, N. Ratliff, G. Gallagher, C. Mertz, K. Peterson, J.A. Bagnell, M. Hebert, A.K. Dey, and S. Srinivasa. Planning-based prediction for pedestrians. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. IEEE, 2009.

List of Figures