Human Motion Prediction for Navigation of a Mobile Robot

eingereichte
MASTERARBEIT
von

cand. ing. Klaus Schmiedhofer

geb. am 12.01.1986
wohnhaft in:
Türkenstrasse 52
80799 München
Tel.: 0176 99478790

Lehrstuhl fur
STEUERUNGS- und REGELUNGSTECHNIK
Technische Universität München

Univ.-Prof. Dr.-Ing./Univ. Tokio Martin Buss
Univ.-Prof. Dr.-Ing. Sandra Hirche

Professor:	Prof. Luca Schenato
Betreuer:	Dipl.-Ing. Betreuer Daniel Althoff
	Dipl.-Ing. Betreuer Roderick de Nijs
Beginn:	18.10.2010
Ende:	18.04.2011

Abstract

Human motion prediction is an important feature to improve the path planning of mobile robots. An exact prediction of the pedestrian trajectory allows not only to avoid them safely but to induce also socially acceptable motions.
In this thesis we introduce a procedure to model social rules, which guide pedestrians through crowded human-lived environments. Human motion is always driven by a future destination and is influenced by the distance to other pedestrians and static obstacles. All this parameters are included to estimate the most probable trajectory, selected from a predefined set of human-like trajectories, the so called pedestrian ego-graph (PEG).

Zusammenfassung

Die exakte Prädiktion einer Fußgänger-Trajektorie ist für mobile Roboter von großer Bedeutung. Sie ermöglicht nicht nur Personen frühzeitig zu umgehen, sondern auch deren Bewegungsmuster zu imitieren. Der hier präsentierte Lösungsvorschlag ist in zwei Teilen realisiert. Zuerst erstellen wir offline ein Set von realistischen Trajektorien und speichern diese in einem sogenannten "Pedestrian Ego-Grap"(PEG).
Um dann die bestmögliche Trajektorie aus einem PEG auszuwählen, berechnen wir für jede eine Kosten-funktion. Diese hängt von den vorher genannten Eigenschaften und der Veranlagung eines jeden Fußgängers ein Ziel zu verfolgen, ab. Am Ende werden all diese Faktoren kombiniert und die wahrscheinlichste Trajektorie des Egographs ausgewählt.

Contents

1 Introduction **5**

2 Related Work **7**

3 Approach **11**
3.1 Markov Decision Process . . . 12
3.1.1 Inverse Reinforcement Learning . . . 14
3.1.2 IRL from sampled trajectories . . . 15
3.2 Ego-graph . . . 16
3.2.1 Probabilistic Ego-graph . . . 16
3.2.2 Trajectory Clustering . . . 16
3.2.3 Probability framework . . . 19
3.3 Description of Human Behavior . . . 21
3.3.1 Spatial Effects . . . 22
3.3.2 Human Interaction . . . 23
3.4 Goal Definition . . . 25
3.4.1 Configuration Space . . . 26
3.4.2 Visibility-graph . . . 28
3.5 Most probable future trajectories . . . 32

4 Implementation **35**
4.1 Generation of the PEG . . . 36
4.1.1 Probabilistic Ego-graph . . . 36
4.1.2 Clustering . . . 37
4.1.3 Positioning of the ego-graph . . . 41
4.2 Goal Analysis . . . 43
4.2.1 Goal-Clustering . . . 44
4.2.2 Generation of Configuration Space . . . 45
4.2.3 Localization of the Goal . . . 45
4.2.4 Visibility-check for the Goal . . . 46
4.3 Cost function . . . 49
4.3.1 Distance function . . . 49
4.3.2 Steering function . . . 49

4.3.3 Obstacle function . . . 50
4.4 Estimation of the Optimal trajectories . . . 52
4.5 Inverse Reinforcement Learning . . . 53
4.6 Update-algorithm . . . 55

5 Simulation and Results **57**
5.1 Simulation . . . 57
5.1.1 Steps of the Algorithm . . . 57
5.1.2 Invisible goal solution . . . 60
5.1.3 Special cases . . . 62
5.2 Results . . . 64
5.2.1 Confront with State of the Art . . . 64
5.2.2 Consideration Simulation Time . . . 65

6 Conclusions and Future Work **67**
6.1 Conclusions . . . 67
6.2 Future work . . . 67

A Implementation **73**
A.1 Dijkstra Algorithm . . . 73
A.2 Dataset . . . 75
A.2.1 Description . . . 75
A.2.2 Analysis and Preparation of the Dataset . . . 76

B Simulation **79**
B.1 Single pedestrian . . . 79
B.1.1 Single cost function . . . 79
B.1.2 Combined cost function . . . 82
B.2 Multiple pedestrians . . . 84

Chapter 1

Introduction

Autonomous navigation through real-world scenarios is a demanding challenge for today's mobile robots. They should be able to navigate among static and moving obstacles to guarantee safe and efficient navigation, but also to follow human-like trajectories.
An important example is the IURO (Interactive Urban Robot). This city explorer has the objective to reach a predefined target in a unknown environment without localization tools like maps and GPS. The only method to obtain information is to

Figure 1.1: IURO: Interactive Urban Robot

interact with humans. Therefore it is necessary that a robot is able to understand human behavior.
Pedestrian consider always, mostly unconscious, the surrounding environment. They keep a comfortable distance to static obstacles and start the avoidance of other humans long time before a collision. Every motion is driven from a future destination, because pedestrians never walk around randomly.
The objective of this thesis is to model these social rules and spatial effects to estimate the most probable trajectory.
The solution is separated in two parts. First we generate human-like trajectories and collect them in a pedestrian ego-graph (PEG). After we use all the informations about human behavior to select the most probable trajectory from the PEG.
To obtain realistic results we train the prediction algorithm on a real dataset.

Chapter 2

Related Work

Human motion prediction is a difficult challenge because it depends on intentions which cannot be directly measured. Many methods are suited only in indoor environments or only for one person. We define some criteria to compare the different approaches.

- Human motion prediction for multiple persons including interaction
- Application to dynamic environments
- Generating multiple probabilistic solutions
- Computational efficient

A suitable method for human motion prediction should fulfill these four criteria. In the following common approaches are evaluated with respect to these criteria.

In [28] an approach for estimating human behavior and the goal position of the pedestrian is presented. The idea is to calculate a trajectory between the goal and the pedestrians current position. They do not use a straight line, but introduce sub-goals. The position of the sub-goals is based on geographical information, the location of obstacles, the current direction and position of the agent in the scene. Evaluating all the sub-goals, they obtain the most likely trajectory.
The major disadvantages are the missing simulation of human behavior, the interaction between multiple pedestrians as well as the general behavior of pedestrians in outdoor environments.

Another problem is the assumption of a static environment. For example [27] models the structure of the environment with a Hidden Markov model (HMM). The states represent particular locations in the environment and the transition matrix, describes the possibility to pass from one state to the other. The HMM is not renewable.
When the environment changes, it is possible to update only the transition matrix

but not the states. Therefore is necessary to create a new HMM for each alteration of the environment.
This operation has a high computational effort and the information about the previous state configuration is thereby lost.

Simon Thompson's idea [29] is to predict trajectories following way-points which describes particular positions in a indoor place. Way-points are locations where persons pass very often, like doors or the cooker in the kitchen.
The algorithm generates realistic results, only if the surrounding environment is know and the model of way points was previously trained on it. This property exclude the possibility to react on changes of the environment.

The estimation method of [32] is based on a goal directed human motion model. A grid map of the environment is generated and for each cell we calculate the distance to all the possible goals.
This first information is used to determine the most probable direction to reach a target and is combined with a constant velocity model. At this point we have a set of trajectories directed to a certain goal. The procedure is repeated for all the possible goals.
To estimate the most probable trajectory we calculate the proportion of the number of trajectories directed to a goal respect to the total number of trajectories and select the best result.
The limitation for this solution is that, no human behavior and information about the environment are included. Therefore the algorithm cannot avoid collisions with other pedestrians and static obstacles.

The social force model presented by Lewin has the drawback, to determine only one possible trajectory and not a probabilistic output for more possible paths [33].
The interesting part of this implementation is the simulation of human behavior. They model social forces with potential fields, like the distance between pedestrians and obstacles. In our approach we include this two forces, but model them like in [5]. Pedestrians are modeled as Gaussian bumps and the interaction between two humans is modeled as a energy function influenced by this Gaussians.

In [1] is described a method to formulate a model for the human motion. It is called spatial behavior cognition model and simulates the internal states of persons. This method allow to generate a probabilistic output considering the interaction between different pedestrians and the method is able to react to fast environment changes. Also is introduce the concept of pedestrian ego-graph(PEG). A PEG is a set of predefined pedestrian-like trajectories used to rapidly predict human paths.

This last construction is the start base for our approach, but we change it in two significant parts. First of all we create multiple PEGs. For each different initial

velocity we generate a new one. Therefore the prediction in time will be more exact, because in each PEG are stored only trajectories with similar velocities.
Another improvement is based on [28]. The paper presents a method to estimate subgoals based on the environment. The property that pedestrian are driven by a specified goal is modeled in our approach by selecting only the trajectories of the PEG oriented to the goal. A reduction of the computational effort is a important consequence.

Summarizing we use a set of predefined trajectories depending on the initial velocity of the pedestrian and select the trajectories oriented to a known target. The most probable trajectory is estimated by the evaluation of the influence of the different social forces on the trajectories.

In the following paragraph we explain how the thesis is structured and list all the methods used to solve the problem:
In chapter 3 are presented all the theoretic fundamentals to implement the prediction algorithm. A Markov decision process is introduce in (3.1), the definition of PEG (3.2) and the human behavior (3.3) are presented. In the last part (3.4) we explain the theory necessary to reduce the PEG.
In chapter (4) we introduce the implementation part. All the functions and methods to implement the prediction algorithm are presented.
The results and simulations are shown in (5). We compare the results with other implementations and show the application of the prediction algorithm in different real scenes.
Finally in (6) we present our conclusions.

Chapter 3

Approach

The human motion prediction is a complex challenge and its main goal is the estimation of the most probable future trajectories of a pedestrian.
Human motion is influenced by different parameters and variables like the environment, spatial effects and the human behavior.
A Markov decision process (MDP) is presented to model all this different influences and to solve the optimization problem.
The states of the MDP are modeled with a pedestrian ego-graph (PEG) [1]. It is a local motion approach used in the field of mobile robots. We use it to represent possible pedestrian trajectories.
Clustering algorithms associate indirectly the transition probability to each trajectory of the PEG and therefore we obtain a PEG of policies. The generation of the PEG is done offline to reduce the computational effort of the prediction algorithm.

In the online part we will determine the set of the most probable trajectories. Two factors influence this decision.
The first, is the effect of a goal on human behavior. We select only the set of trajectories leading to the goal of the pedestrian, assuming that the goal is known for every pedestrian.
The second factor models different internal and external factors (spatial effects) by cost functions representing influences like the comfortable distance between pedestrians and obstacles and the disposition to choose the shortest way.
Both factors are summarized in the reward function of the MDP.
All the components of the Markov decision process are presented except one, the weightings of the cost functions.
To estimate realistic values we solve a "Inverse Reinforcment Learning"-problem. All the policies of the PEG are compared with a trajectory from a real dataset, and the combination of weightings, that generates the minimum distance error between them is the optimal combination.
At this point we are able to solve the optimization problem of the MDP and estimate a set of most probable trajectories.

3.1 Markov Decision Process

The basic modeling paradigm underlying the MDP [11] is that of a decision maker who interacts with an external environment by taking actions. The sequential decision problem is formulated as a set of states, each of which models a particular situation in the decision problem.
A crucial assumption made in the MDP model is that the evolution of the states is a Markov process, meaning the distribution of the future states is conditionally independent of the past, given perfect knowledge of the current state.
The desirability of a state is determined by a reward. The goal of the agent is to choose actions such that it maximizes its longterm reward.
The transition from one state to the next is governed by probability distribution that reflects the uncertainty in the outcome decision. The first definition is the discounted optimality framework:

Definition 1. *A discrete Markov decision process $M = (S, A, P^a_{ss'}, \lambda, R^a_{ss'})$ is defined by a finite set of discrete states $s \in S$, a finite set of actions $A = \{a_1, ..., a_t\}$, a transition model $P^a_{ss'}$, specifying the distribution over future states s' when an action a is performed in state s, a corresponding reward model R, specifying a scalar cost or reward and a discount factor $\lambda \in (0, 1)$.*

Abstractly, a value function is a mapping $S \rightarrow \mathbb{R}$. Given a policy $\pi : S \rightarrow A$ mapping states to actions, its corresponding value function V^π specifies the expected long-term sum of rewards received by the agent in any given state s when actions are chosen using the policy π.

Definition 2. *The long-term discounted value associated with a state under a fixed policy is defined as:*

$$V^\pi(s) = E_\pi(r_t + \lambda r_{t+1} + \lambda^2 r_{t+2} + ...|s = s_t)$$

where E_π indicates the conditional expectation that the process begins in the initial state s. Actions are chosen at each step using policy π, and r_t is the reward received at the time step t.

To any policy π we can associate a value function V_π which is the fixed point operator T^π.

Definition 3. *Given any deterministic policy π, the operator T^π is defined as*

$$T^\pi(V)(s) = \sum_{s' \in S} P^{\pi(s)}_{ss'}(R^{\pi(s)}_{ss'} + \lambda V(s')).$$

It can be shown that $V^\pi(s)$ is a fixed point of T_π, that is

$$T^\pi(V^\pi)(s) = V^\pi(s).$$

Definition 4. *The optimal value function V^* of a MDP is defined as*

$$V^*(s) \equiv V^*(s) \geq V^\pi(s) \quad \forall \pi, \quad s \in S,$$

where the optimal policy π^* is defined as

$$\pi^*(s) \in argmax_a(R_{sa} + \lambda \sum_{s'} P^a_{ss'} V^*(s')).$$

where, $R_{sa} = \sum_{s' \in S} P^a_{ss'} R^a_{ss'}$

In general the optimal policy π^* is not unique. However, any optimal policy π^* defines the same unique optimal value function. The optimal value function can be shown to be a fixed point of another operator T^*, defined as follows.

Definition 5. *The fixed point operator T^* for a MDP is defined as*

$$T^*(V)(s) = max_a(R_{sa} + \lambda \sum_{s' \in S} (P^a_{ss'} V(s'))$$

It can be shown that the optimal value is a fixed point of T^*, that is

$$T^*(V^*)(s) = V^*(s), \quad \forall s \in S.$$

Action-value functions are mappings from $S \times A \to \mathbb{R}$ and represent a convenient reformulation of the value function.

Definition 6. *The optimal action value $Q^*(x, a)$ is defined as the long-term value of the non stationary policy performing action a first, and then acting optimally, according to V^*:*

$$Q^*(s, a) \equiv E(r_{t+1} + \lambda max_{a'} Q^*(s_{t+1}, a') | s_t = s, a_t = a),$$

where $V^*(s) = max_a Q^*(s, a)$.

where r_{t+1} is the actual reward received at the next time step, and s_{t+1} is the state resulting from executing action a in state s_t. The corresponding Bellman equation for $Q^*(x, a)$ is given as

$$Q^*(s, a) = R_{sa} + \lambda \sum_{s'} P^a_{ss'} max_{a'} Q^*(s', a').$$

This formulation allows to solve the standard MDP-problem of the optimal action-value function with two types of methods: The exact and the approximation method.

An example for the first is the Value-Iteration-method, where the algorithm computes the next approximation V^{t+1} by iteratively "backing up" the current approximation:

$$V^{t+1}(s) = T^*(V^t)(s) = max_a(R_{sa} + \lambda \sum_a P^a_{ss'} V^t(s')).$$

The approximation methods retain the restriction of representing functions exactly, but require only sample transitions (s_t, a_t, r_t, s'_t) instead of true knowledge of the MDP. Such methods can be generically referred to as simulation-based methods and used in various areas including reinforcement learning.
A well note method is the Monte-Carlo method, which is based on the idea that the value of a particular state $V^\pi(s)$ associated with a particular policy π can be empirically determined by "simulating" the policy π on a given MDP, and averaging the sum of rewards received. As a statistical procedure, they have the attractive property of being unbiased estimators of the true value.

However to solve our problem we need the *inverse* reinforcement learning method. For this we introduce the formal definition of the IRL-problem and show an application on sampled trajectories.

3.1.1 Inverse Reinforcement Learning

Inverse reinforcement learning(IRL) deals with the inversion problem to that of an MDP [8]:

Given a policy π_i and the Markov decision model (X, A, P, λ), IRL seeks to determine a reward function R^* such that π_i is an optimal policy for the MDP (X, A, P, R^*, λ)

In the previous section we introduced the Markov decision process and different solution methods. We suppose to know all the parameters and variables, except the reward function. To solve this problem we use the inverse reinforcement learning algorithm [7]. This method is able to estimate the reward function,based on known different policies π. The general IRL-algorithm generates a set of all reward functions for which a given policy is optimal. The definition of the solution set is:

Definition 7. *Let a finite state space S, a set of actions $A = \{a_1, ..., a_k\}$, transition probability matrices $\boldsymbol{P}_a$, and a discount factor $\lambda \in (0, 1)$ be given. Then the policy π given by $\pi(s) \equiv a_1$ is optimal if and only if, for all $a = a_2, ..., a_k$, the reward $\boldsymbol{R}$ satisfies*

$$(\boldsymbol{P}_{a_1} - \boldsymbol{P}_a)(\boldsymbol{I} - \lambda \boldsymbol{P}_{a_1})^{-1} \boldsymbol{P} \succeq 0$$

is necessary and sufficient for $\pi = a_1$ to be unique optimal policy.

The optimal policies had two problems: First, $\mathbf{R} = constant$ is always a solution, when we use the same reward function, no matter what action we take, then any

policy, including $\pi = a_1$ is a optimal solution. Second, for most MDPs, there are many choices of $\mathbf{R}$ that meet the criteria. To reduce the number of optimal policies we use another equation for $\mathbf{R}$, a more restrict one, which favor solutions that make any single step deviation from π as costly as possible. We seek to maximize the sum of the differences between the quality of the optimal action and the quality of the next best action.

$$\sum_{s \in S} (Q^{\pi}(s, a_1) - max_{a \in A \setminus a_1} Q^{\pi}(s, a)) \tag{3.1}$$

Another improvement is to use a penalty term $-\lambda|R|$, is a adjustable coefficient that balances between having small reinforcements and of maximizing the previous function (3.1). Combining all this terms and equations we obtain the following optimization problem:

$$\begin{aligned} maximize \sum_{i=1}^{N} min_{\{a \in \{a_2, \ldots, a_k\}\}} \{(\mathbf{P}_{a_1}(i) - \mathbf{P}_a(i))(\mathbf{I} - \lambda \mathbf{P}_{a_1})^{-1}\mathbf{R}\} - \lambda \parallel \mathbf{R}_1 \parallel \\ s.t. (\mathbf{P}_{a_1} - \mathbf{P}_a)(\mathbf{I} - \lambda \mathbf{P}_{a_1})^{-1}\mathbf{R} \qquad \forall a \in A \; a_1 \\ |\mathbf{R}_i| \leq R_{max}, i = 1, ..., N \end{aligned}$$

3.1.2 IRL from sampled trajectories

We reformulate the IRL problem for a special case of sampled trajectories [7], where we have access to the policy π only through a set of actual trajectories in the state space and not through the combination of transition matrix and state matrix.
The reward is defined as $R_i(s) = w_i^1\phi_1(s) + w_i^2\phi_2(s) + w_i^3\phi^3(s) + w_i^4\phi_4(s)$ a linear combination of four weighted functions.
We first execute m Monte Carlo trajectories under the policy π and for each of them we estimate all the possible value function

$$V_i^*\pi(s_0) = R_i(s_0) + \lambda R_i(s_1) + \lambda^2 R_i(s_2) + ...$$

for the linear combinations of $w_i^l \quad l = 1, .., 4$.
We consider a set of $\{\pi_j\} \quad j = 1, ...k$ policies and suppose that the optimal policy π_p is given. After this considerations we recall the formulation of the general IRL-problem and introduce the following IRL optimization problem:

$$max \sum_{s_0 \in X_0} \sum_{j} (V^{\pi_j}(s_0) - V^{\pi_p}(s_0)) - \lambda w_i \tag{3.2}$$

$$s.t. \; \lambda \geq 1, \tag{3.3}$$

$$0 < w_i \leq w_{max}, \tag{3.4}$$

$$V^{\pi_j}(s_0) \geq (V^{\pi_p}(s_0)) \tag{3.5}$$

The optimization can be solved iteratively or with a linear programming algorithm.

3.2 Ego-graph

The approach ego-graph is a local motion planning method used in the field of mobile robots [6], especially for the robots with motion constraints. In our case we adapt this method and use it for the generation of the trajectories of a pedestrian[1].
The creation of a pedestrian ego-graph is divided in different steps.
Randomly trajectories, limited by human and dynamic constraints, are generated.
Different clustering algorithms are then presented to extract the most human-like trajectories, which define the pedestrian ego-graph (PEG) and associate indirectly the transition probability to the states.

3.2.1 Probabilistic Ego-graph

Every trajectory is composed by states s of the MDP. The PEG is modeled with the Constant Acceleration Model:

$$p_x(t) = p_x(t-1) + (v_x(t-1) + \frac{1}{2}a_x(t)\Delta t)\Delta t \tag{3.6}$$

$$p_y(t) = p_y(t-1) + (v_y(t-1) + \frac{1}{2}a_y(t)\Delta t)\Delta t \tag{3.7}$$

p_x and p_y are the position of the state $s = (p_x, p_y)$, v_x and v_y are the velocities of the states, respectively in the x-direction and y-direction, and a_x and a_y are the accelerations. Δt is the time interval between two sequential states. The different trajectories are obtained with a fixed initial velocities and uniform random accelerations.
Pedestrian have a large range of velocities. To model all of them we separate this range and determine a new ego-graph with the respective initial velocity for each of them. Human tend to maintain their velocity constant during a walk [15], but it is also important to model braking and accelerated trajectories. To combine all this factors, we use the initial velocity as a reference parameter and fix a maximal and minimal velocity for each ego-graph

3.2.2 Trajectory Clustering

Cluster analysis or clustering is the assignment of a set of observations into subsets (called clusters) so that observations in the same cluster are similar according trajectories similarities and distance metric.

The idea is to extract the most probable trajectories from the probability ego-graph. We create spatial partition-sets and cluster there all the trajectories. A linear regression mixture clustering method is utilized to estimate the parameters characterizing every partition-set [16]. The special property of this method is to estimate the general description, assuming to have trajectories which is a function

of the known variable. In our case we consider the trajectories y_j of length n_j of the ego-graph depending on the time x (regression model).
The dependency on another variable is the reason why standard algorithm like the K-means algorithm are not applicable

Mixture model

Before we define the regression model we introduce a mixture model, which allows to model the dependency of a trajectory to belong to a cluster k.

- Set of different trajectories y_j
- Every trajectory y_j is assigned to a cluster k with probability w_k, $\sum_{k=1}^{K} w_k = 1$, where K is the number of clusters.
- Given that a trajectory y_j belongs to cluster k, the respective density function is $f_k(y_j|\theta_k)$, where θ_k are the parameters of the cluster k.

and the observed density is a mixture model:

$$P(y_j|\theta) = \sum_{k}^{K} f_k(y_j|\theta_k) w_k \tag{3.8}$$

Thus, if we observe the y_j, and we assume a particular functional form of f_k components, we can try to estimate from the data the most likely values of the parameters θ_k and the weights w_k, characterizing the cluster k. To estimate the maximum likelihood is possible to use the EM-algorithm [23].

Mixture of Regression Model

We generalize this mixture model to a mixture's of regression model. Every trajectory y_j is a function of the known x. We assume the standard regression relationship between x and y,

$$y = g_k(x) + e \tag{3.9}$$

where e is a zero-mean Gaussian with standard deviation σ_k (we consider a constant noise model for all trajectories) and $g_k(x)$ is a deterministic function of x.
We assume the case of a Gaussian noise and the new conditional density result $f_k(y|x, \theta_k)$ depending now also on x. In words, given trajectory y_j it belongs to cluster k with mean $g_k(x)$ and standard deviation σ_k. Here θ_k includes both parameters. Furthermore we segment the density function of every trajectory y_j in the density function for every state $y_j(i)$:

$$f_k(y_j(i)|x_j(i), \theta_k)$$

All this assumption result in the following equation:

$$P(y_j|x_j,\theta_k) = \prod_i^{n_j} f_k(y_j(i)|x_j(i),\theta_k) = \sum_k^K f_k(y_j|x_j,\theta_k)w_k \tag{3.10}$$

The first notation is justified by the standard regression assumption that the noise is independent at different states $y_j(i)$ along the trajectory. The second step is motivated, considering that we don't know which component generated which trajectory. Solving other steps to generalize the result we obtain the log-likelihood, necessary to implement the EM-algorithm to estimated the cluster parameter θ_k:

$$L(\theta|S) = \sum_j^M log \sum_k^K w_k \prod_i^{n_j} f_k(y_j(i)|x_j(i),\theta_k) \tag{3.11}$$

EM Algorithm for Mixture of Regression Models

A EM algorithm is the simplest way to estimate the group behavior of a statistical model, but only if is known to which group each trajectories belongs.
This information is not available and we must implement a more complex procedure to solve the EM-problem. To be exact we use the common approach for dealing with hidden data the extended version of the EM-algorithm[25].
The idea is to assume to know the hidden data. Then we work out the simpler problems like the expectation and maximization step and re-estimate the hidden data again using the current answers that we just computed. This procedure is repeated until the algorithm converge to a stable value. The convergence is guaranteed by the convergence of the EM algorithm.
Solving different equation and introducing the new variable z_{jk}, the probability that the trajectory j belongs to cluster k.
The result is the division of the membership probability w_k from the rest of the functions.

$$E[L(\theta|S,Z)] = \sum_j^M \sum_k^K h_{jk} log w_k + \sum_j^M \sum_k^K \sum_i^{n_j} h_{jk} log f_k(y_j(i)|x_j(i),\theta_k) \tag{3.12}$$

where $h_{jk} = E[z_{jk}]$ and corresponds to the posterior probability that the trajectory was generated by component k. It is important to note that for every trajectory is valid $\sum_k^K w_k = 1$. We adapt our notation of Eq.(3.12) to fit the multidimensional data in this framework. The equations changes a follows:

$$Y_j = X_j\beta_k + e_k, \tag{3.13}$$

with $Y_j = [1 \quad y_j(1) \quad ... \quad y_j(n_j)]'$,

$$X_j = \begin{bmatrix} 1 & x_j(1) & x_j(1)^2 & ... & x_j(1)^p \\ 1 & x_j(2) & x_j(2)^2 & ... & x_j(2)^p \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_j(n_j) & x_j(n_j)^2 & ... & x_j(n_j)^p \end{bmatrix}$$

Y_j is a vector with all the trajectory observations and X_j is a n_j by $p+1$ matrix, with the corresponding time instant for every observation. p is the order of the regression model. e_k is a vector consisting of zero-mean Gaussian with variance σ_k^2, and β_k is a vector of regression coefficients. To collect this result with the mixture model, we equal the conditional density to a Gaussian with mean $X_j\beta_k$ and covariance matrix $diag(\sigma_k^2) = e_k$. Considering now that we start with a randomly initialization for the membership probability we solve the maximization step of the EM algorithm (Eq.(3.14)). To obtain the parameters $\theta_k = \{w_k, \beta_k, \sigma_k^2\}$ we solve the weighted least squares problem [24]. The regression coefficients are:

$$\hat{\beta}_k = (X'H_kX)^{-1}X'H_kY \tag{3.14}$$

$$\hat{\sigma_k^2} = \frac{(Y - X\hat{\beta}_k)'H_k(Y - X\hat{\beta}_k)}{\sum_j^M h_{jk}} \tag{3.15}$$

$$\hat{w_k} = \frac{1}{M}\sum_j^M h_{jk} \tag{3.16}$$

All this steps lead to the following EM algorithm for mixtures of linear regression models:

1. Randomly initialize the membership probabilities h_{jk}
2. Calculate new estimates for $\hat{\beta}_k, \hat{\sigma_k^2}$, and $\hat{w_k}$ from the weighted least squares solutions, using the current membership probabilities as weights.
3. Compute the new membership probabilities using Eq.(3.7) and the newly computed parameter estimates from the previous step.
4. Loop to step 2 until the log-likelihood stabilizes.

3.2.3 Probability framework

The number of clusters k for every partition-set is fixed. So far we considered $k = 1$, but thereby a partition-set containing one or 1000 trajectories has always the same number of cluster in the pedestrian ego-graph. Hence we introduce a hierarchal method to augment the number of cluster k for every partition-set till some conditions are not accomplished.
The main condition is the mean squared error between all the trajectories and the

cluster representation for a partition-set. Every cluster is divided in two new cluster till the condition is not satisfied.
Another condition is the number of trajectories, should those drop under a fixed limit, then the "segmentation" is interrupted.
All the obtained trajectories are then stored in the final pedestrian ego-graph and an example Ego-graph is shown in Figure (3.1).

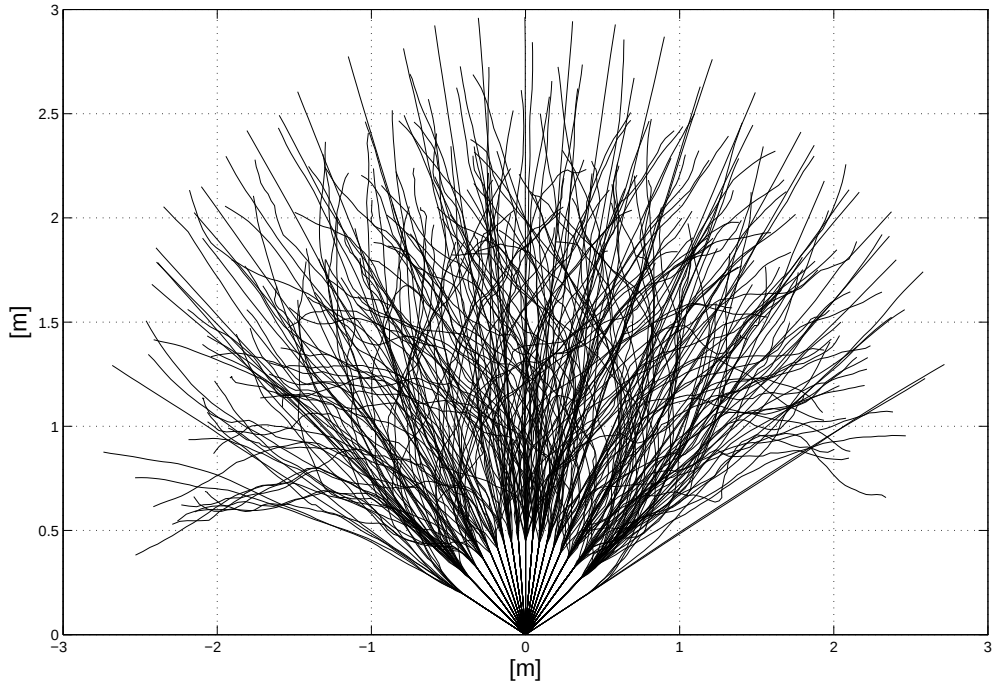


Figure 3.1: Pedestrian Ego-Graph

From now on, when we talk about pedestrians trajectories, we mean a policy. This is not completely correct, because as yet we have not introduced any definition for an action a. But indirectly we have modeled the transition probability $P^a_{ss'}$, to pass from state s to state s' with the action a with the considerations about the number of clusters and the introduction of the mixture regression model.

3.3 Description of Human Behavior

The idea of the cost functions is not only to avoid all possible collisions but also to help the robot to move and drive following human behviors. The pedestrian behaviors are based on internal states of the persons. In [33] the idea that behavioral changes are guided by social fields is described. The following figure explains the theory.

A sensory stimulus is noticed by the pedestrian. Combining this perception with

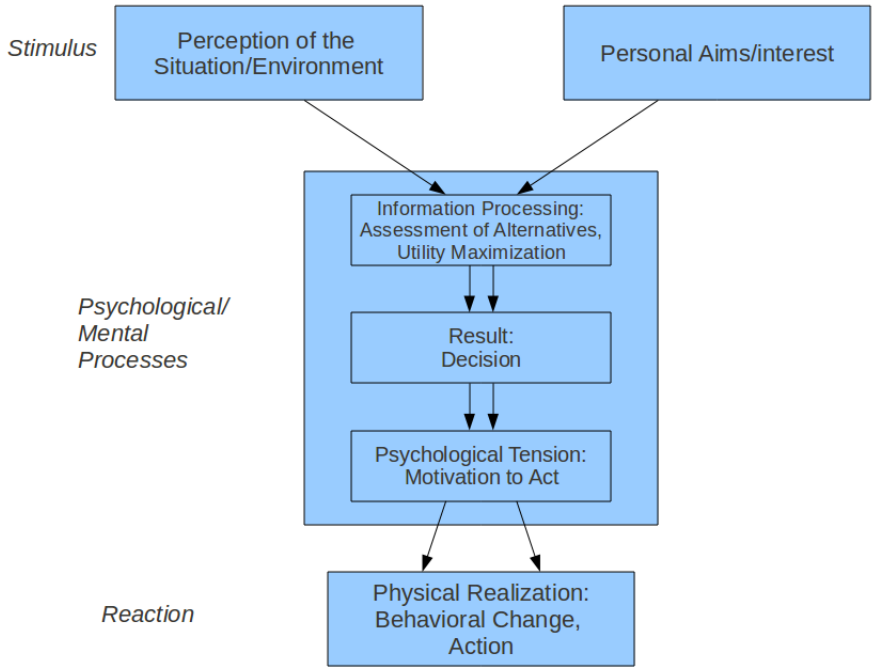


Figure 3.2: Schematic representation of processes leading to behavioral changes [33]

the personal aim, a set of different probable decisions can be generated. The optimal decision induce then a certain behavioral reaction. Evaluating this concept, we define different social forces, following [3]:

- Reaching a certain goal location as comfortable as possible:
  - Shortest way: Distance cost function : C_{dist}
  - Straightest way: Steering cost function : C_{str}
- Keeping a certain distance to borders : Static obstacle cost function : C_{obs}

To describe the interaction between different pedestrians we use the definition of personal space explained in [5]. The idea is to model the area around a person with two Gaussian functions and retrieve from them the cost function C_{pers}.

The next step is to combine all this cost functions for every state of the MDP. The total cost function is written as a linear combination of these four components with different weights.

$$C_{total}(s_i) = w_{dist}C_{dist}(s_i) + w_{str}C_{str}(s_i) + w_{obs}C_{obs}(s_i) + w_{pers}C_{pers}(s_i) \tag{3.17}$$

s_i is the pedestrian state in time step i. Every state is composed by a x-coordinate p_x and y-coordinate p_y.
$C_{total}(s_i)$ is the reward function for the MDP when we consider the transition from state s_{i-1} to s_i following the policy π.
In the following paragraphs the singular cost functions are explained.

3.3.1 Spatial Effects

Distance Function

This cost functions represents the social rule of humans to use always the shortest way to reach a goal. To simulate this behavior we calculate the euclidean distance between two states in the MDP.

$$C_{dist}(s_i) = dist(s_i - s_{i-1})$$

Steering Function

The objective of every person is to reach a goal as comfortable as possible. Normally the persons try to reach the goal on the straightest way, but if there are obstacles in the space between the person and the goal, the pedestrian must avoid them. In this case the person try to steer as minimum as possible. In mathematical terms it means that we penalize the following steering variation:

$$C_{str}(s_i) = (steering(s_i) - steering(s_{i-1}))^2 \tag{3.18}$$

Static obstacle

The static obstacle function modeled the relation between the pedestrian and a static obstacles with the repulsive forces[3].
The objective of this cost function is to model the influence of static obstacles to human motion behavior. To create this function we calculate the distance map for the person respect to the closest obstacle. The following formula converts the map to an energy function. This means that a cell close to the obstacle has a value close to 1, and a energy function of a cell far away tend to 0.

$$C_{obs}(s_i) = \exp(-0.5\frac{dist(s_i)^2}{\sigma_d^2})\exp(-0.5\frac{dist(s_i)^2}{\sigma_w^2}) \tag{3.19}$$

The first term has the function to penalize states near to a obstacle. In contrast the second models the distance of influence of an object. Obstacles farther away then σ_w are not considered. This second term is not originally used in [1] but found in the article [2] and considered as a useful constrain for the energy function.

3.3.2 Human Interaction

The human interaction cost function is based on the personal space theroy [5] and explains the behavior between pedestrians. The goal is like in the previous sections to find a mathematical description of this spatial human behavior. The description of the pedestrian cost function is based on the definition of the personal space. Edward Hall [4] introduced this notation to describe the different space areas around a person. The theory is based on the characteristic of persons, that they hold unconsciously a shape surrounding him like bubbles. The shape of the personal

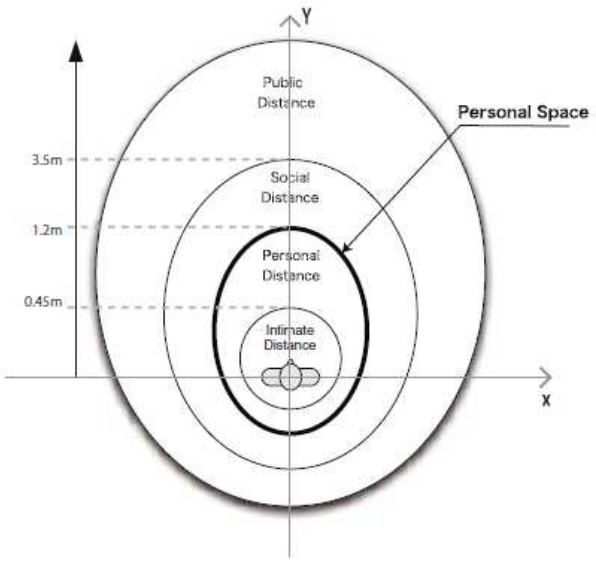


Figure 3.3: Definition of the Personal Space [5]

.

space is affected by four parameters: face orientation, distance, gender and age. The last two parameters are neglected, because at the current state of the art, it is impossible to determine the gender and age of pedestrians in a certain distance. Hence the personal space depends only from two variables: Face orientation and distance between two pedestrians.
To describe the personal space we follow the steps of [4]. We create a new local coordinate system in which the person is the origin p. The x-axis is oriented along the face and the y-axis along the sight direction. The personal space around the person can then be defined as a function Φ_p which has the maximum at p and

decreases with the increase of the distance to p. This can be represented by a two-dimensional Gaussian function Φ_p with covariance matrix Σ and centered at p.

$$\Phi_p(q) = \exp(-0.5(q-p)^t\Sigma^{-1}(q-p))$$

where $q = (x_q, y_q)$ is a second person and Σ is a diagonal matrix, which changes if a person is in front (Σ_f) or on the side (Σ_s) of the principal pedestrian:

$$\Sigma_f = \begin{pmatrix} \sigma_{xx}^2 & 0 \\ 0 & \sigma_{yy}^2 \end{pmatrix} \qquad \Sigma_s = \begin{pmatrix} \sigma_{xx}^2 & 0 \\ 0 & \sigma_{xx}^2 \end{pmatrix}$$

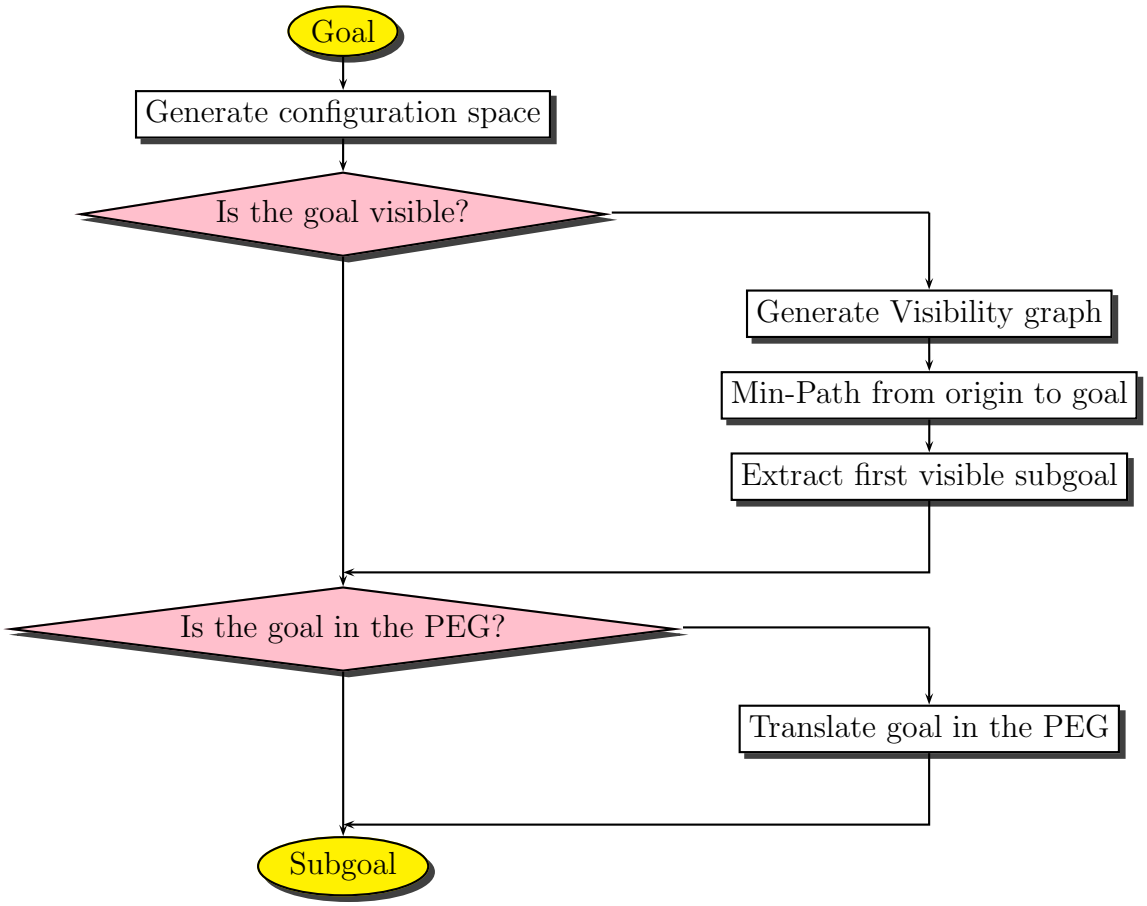
The value σ depends from the dimension of the personal space of the pedestrian. These theoretical considerations allow us now to define the cost function between the person k, whose trajectory we try to predict, and another pedestrians j in the scene:

$$C_{pers}(s_i^k) = \sum_j C_{jk} = \sum_j exp(-0.5(s_i^j - s_i^k)^t\Sigma^{-1}(s_i^j - s_i^k)) \tag{3.20}$$

and we use Σ_f if j is in front of k and Σ_s if j is on the side of k.

3.4 Goal Definition

An important feature of the human motion prediction is the definition of a target or goal for a pedestrian. The objective is to reduce the number of trajectories in the ego-graph, to decrease the computational effort.
We suppose that every human try to reach a particular geographical goal. Recent psychological studies shows that pedestrian behaviour is always influenced by a goal. Also if a human is not following intentionally a goal [26].
This information we will include in our model. Obviously this implicates that every pedestrian must have associated a goal, but we suppose this as a external input for our algorithm.
The idea is to define a subgoal inside the maximum range of the PEG and cluster around the trajectories.
To define a subgoal we have to check different characteristics:



In the following we introduce the theoretical basis to realize this procedure.
To solve all this problems we need to introduce the configuration space , a visibility graph and the Dijkstra-algorithm to determine the shortest path.

3.4.1 Configuration Space

The notion of configuration space derives from the physics. A configuration space is the space of possible states that a physical system may attain, possibly subject to external constraints.
In robotics this notation is useful to avoid collisions between robots and obstacles. The idea is to sum the shape of the obstacle and the robot in every point and create so a collision free space, called configuration space.
Obviously the idea is also applicable between pedestrian and obstacles and we consider the description between the shape of a human and the obstacles of the environment.
Following the explanation of [17] we define a obstacle region in a plane, model a pedestrian as a rigid body with circular shape and present the formulation of configuration space.

Suppose that the world $\boldsymbol{W} \subset R^2$ contains an obstacle region, $\boldsymbol{O} \subseteq \boldsymbol{W}$. Assuming that a pedestrian shape is defined as a rigid body and more in detail as a circle $\boldsymbol{A} \subset \boldsymbol{W}$.
Let $q \in \boldsymbol{C}$ denote the configuration of $\boldsymbol{A}$, in which $q = (x_t, y_t)$ for $\boldsymbol{W} \in R^2$.
The obstacle region, $\boldsymbol{C}_{obs} \subseteq \boldsymbol{C}$, is defined as

$$\boldsymbol{C}_{obs} = \{q \in \boldsymbol{C} | \boldsymbol{A}(q) \cap \boldsymbol{O} \neq \emptyset\},$$

which is the set of all configurations, q, at which $\boldsymbol{A}(q)$, intersects the obstacle region $\boldsymbol{O}$. Since $\boldsymbol{O}$ and $\boldsymbol{A}(q)$ are closed sets in $\boldsymbol{W}$, the obstacle region is a closed set in $\boldsymbol{C}$.
The leftover configurations are called the free space, which is defined and denoted as $\boldsymbol{C}_{free} = \boldsymbol{C} \setminus \boldsymbol{C}_{obs}$. Since $\boldsymbol{C}$ is a topological space and $\boldsymbol{C}_{obs}$ is closed, $\boldsymbol{C}_{free}$ must be an open set. This implies that the pedestrian can come arbitrarily close to the obstacles while remaining in $\boldsymbol{C}_{free}$. If $\boldsymbol{A}$ "touches" $\boldsymbol{O}$,

$$int(\boldsymbol{O}) \cap int(\boldsymbol{A}(q)) = \emptyset \quad \text{and} \quad \boldsymbol{O} \cap \boldsymbol{A}(q) \neq \emptyset$$

(int is the interior operation) then $q \in \boldsymbol{C}_{obs}$. The condition above indicates that only their boundaries intersect.

The next step is to define how it is possible to generate the obstacle space. The simplest case for $\boldsymbol{C}_{obs}$ is when $\boldsymbol{C} \subset R^2$ and the pedestrian is again modeled as a circle shape and the motion is restricted to rotation and translation. Under these

conditions, $\boldsymbol{C}_{obs}$ can be expressed as a type of convolutions. For any two sets $X, Y \subset R^2$ let their Minowski sum be defined us:

$$X \oplus Y = \{x + y \in R^2 | x \in X \quad and \quad y \subset Y\}$$

in which $x + y$ is just vector addition on R^2. In terms of the Minowski sum we obtain, $\boldsymbol{C}_{obs} = \boldsymbol{O} \oplus \boldsymbol{A}$ and recalling $\boldsymbol{C}_{free} = \boldsymbol{C} \setminus \boldsymbol{C}_{obs}$ the configuration space is defined.

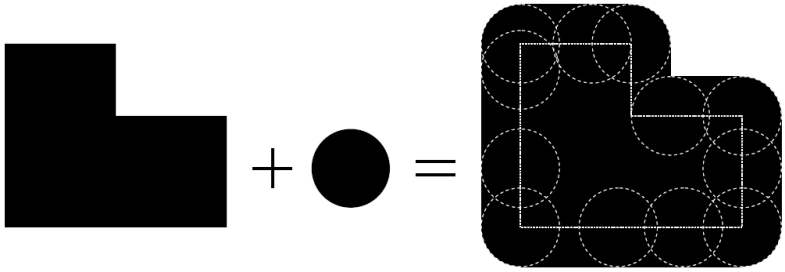


Figure 3.4: Configuration space (white) of a obstacle map

3.4.2 Visibility-graph

The visibility graph is a graph of inter visible locations, typically for a set of points and obstacles in the euclidean plane. Each node in the graph represents a point location and each edge a visible connection between them. A connection is visible if the line of sight connecting two locations does not cross any obstacle.
The start vertex of the visibility graph is the actual position of the pedestrian and the end vertex is his goal. The other vertexes are the corner of the configuration space.
In the section, Delaunay Triangulation is used to estimate all the visible edges between the vertex of the visibility graph.
In the last section, the Dijkstra-algorithm permits to estimate the shortest path through the visibility graph.

Delaunay Triangulation

The Delaunay triangulation DT(P) [18] is used to find all the visible edges between a set of independent points P. More in detail generates triangle between the points such that no point is inside the circumcircle of any triangle in DT(P).

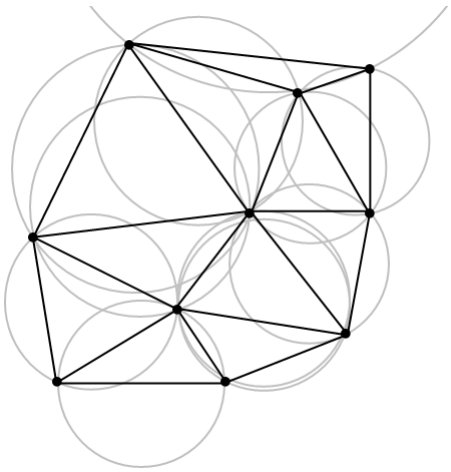


Figure 3.5: Delaunay triangulation and the circumcircles.

At the begin we want introduce the definition for a general triangulation in a Euclidean plane:

Definition 8. *Let* $\boldsymbol{A}$ *be a point configuration in two dimensional space, with set of labels J. A collection* $\mathcal{J}$ *of affinely independent subsets of J is a triangulation of* $\boldsymbol{A}$ *if it satisfies the following conditions:*

- *If* $B \in \mathcal{J}$ *and* $F \subset B$*, then* $F \in \mathcal{J}$ *as well. (Closure Property)*

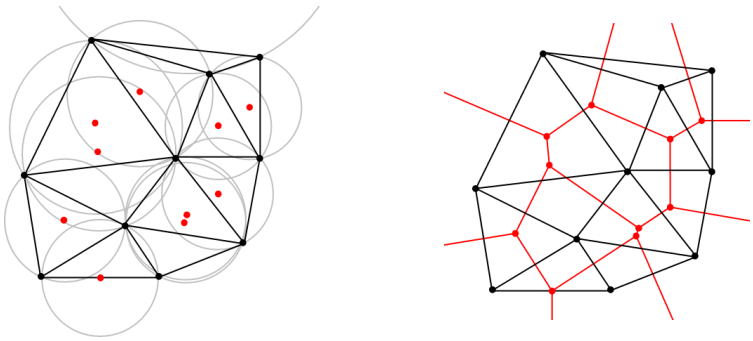


Figure 3.6: A Delaunay triangulation and the dual Voronoi diagrams.

- $\bigcup_{B \in \mathcal{J}} conv(B) = conv(\boldsymbol{A})$. *(Union Property)*

- *If* $B \neq B'$ *are two cells in* $\mathcal{J}$*, then* $relint(B) \cap relint(B') = \emptyset$ *(Intersection Property)*

A particular type of a regular triangulation in 2D is the Delaunay Triangulation:

Definition 9. *The Delaunay subdivision of a point configuration* $\boldsymbol{A}$ *is the regular subdivision obtained by the choice of lifting heights given by* $\omega(\boldsymbol{p}) = \|\boldsymbol{p}\|^2$*. A Delaunay triangulation is any triangulation that refines it. We want to annotate that* $\omega(\boldsymbol{p}) = \|\boldsymbol{p}\|^2$ *is the associated height function to the point set* $\boldsymbol{A}$ *and* $\omega : \boldsymbol{A} \to R^2$*.*

Planar subdivision is the subdivision of a plane into a set of non-overlapped regions that cover the whole plane. A important construction property is the "empty-sphere" characterization:

Definition 10. *Let* $\boldsymbol{A} \subset R^2$ *be a finite point set, with the label set J. let* $B \subset J$ *be a subset of its elements. Then B labels a cell in the Delaunay subdivision of* $\boldsymbol{A}$ *if and only if there is a circle with no points of* $\boldsymbol{A}$ *in its interior and with exactly the points of* $\boldsymbol{A}$ *labeled by B on its boundary.*

A interesting note is the dual part of the Delaunay triangulation is the Voronoi Diagram. Every Voronoi cell is the origin for the circumcircle for every Delaunay subdivision. A theoretic definition describes this fact as following:

Definition 11. *The dual graph of the Voronoi diagram of a point configuration* $\boldsymbol{A}$ *consists precisely of the edges of the Delaunay subdivision.*

The implementations to construct a complete Delaunay triangulation are various. The most intuitive approach is the flipping technique. In the image (3.7) is shown an example.

We consider two triangles ABD and BCD with the common edge BD. We check if the two triangles meet the Delaunay condition presented in Def.9. In the middle

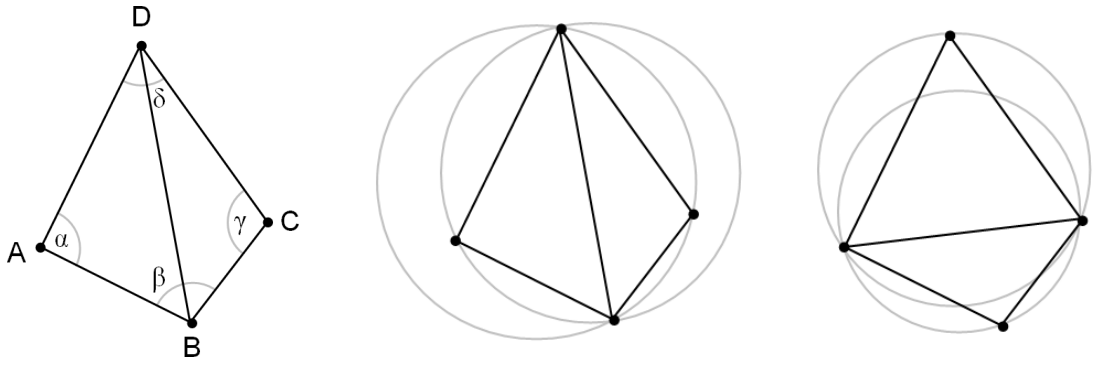


Figure 3.7: Flipping method for Delaunay Triangulation.

image it is obvious that th condition is not met, because both circumcircle contain another vertex. Flipping the common edge we obtain the last configuration and two Delaunay subdivision which respect the condition.
Considering a general set of points **A**, the Flip-algorithm generates all the possible triangulations, checks which triangle does not meet the condition (Def.9) and flips the edge. These procedure is repeated till all triangles meets the "empty sphere"condition.
The approach has a computational effort of $O(n^2)$ where n is the number of points in the set **A**.
Another approach is the Incremental-algorithm, which permits to reduce the computational effort to $O(n \log(n))$ [19].
One vertex after the other is added and after every insertion is checked if all edges meet the Delaunay property. But it is important to choose the new vertex always randomly otherwise the computational complexity will be still $O(n^2)$.

Dijkstra-Algorithm

The Dijkstra-algorithm is a method to estimate the shortest path for a graph with non-negative edge path costs [22]. Given a start vertex the algorithm determine the minimum cost-paths to every other vertex. The computational complexity in the worst case, using a Fibonacci heap, is $O(|E| + |V|log|V|)$, where $|E|$ is the number of edges an $|V|$ the number of vertexes. In the following we show a sequence with the steps to determine a minimum path from vertex a to vertex b.

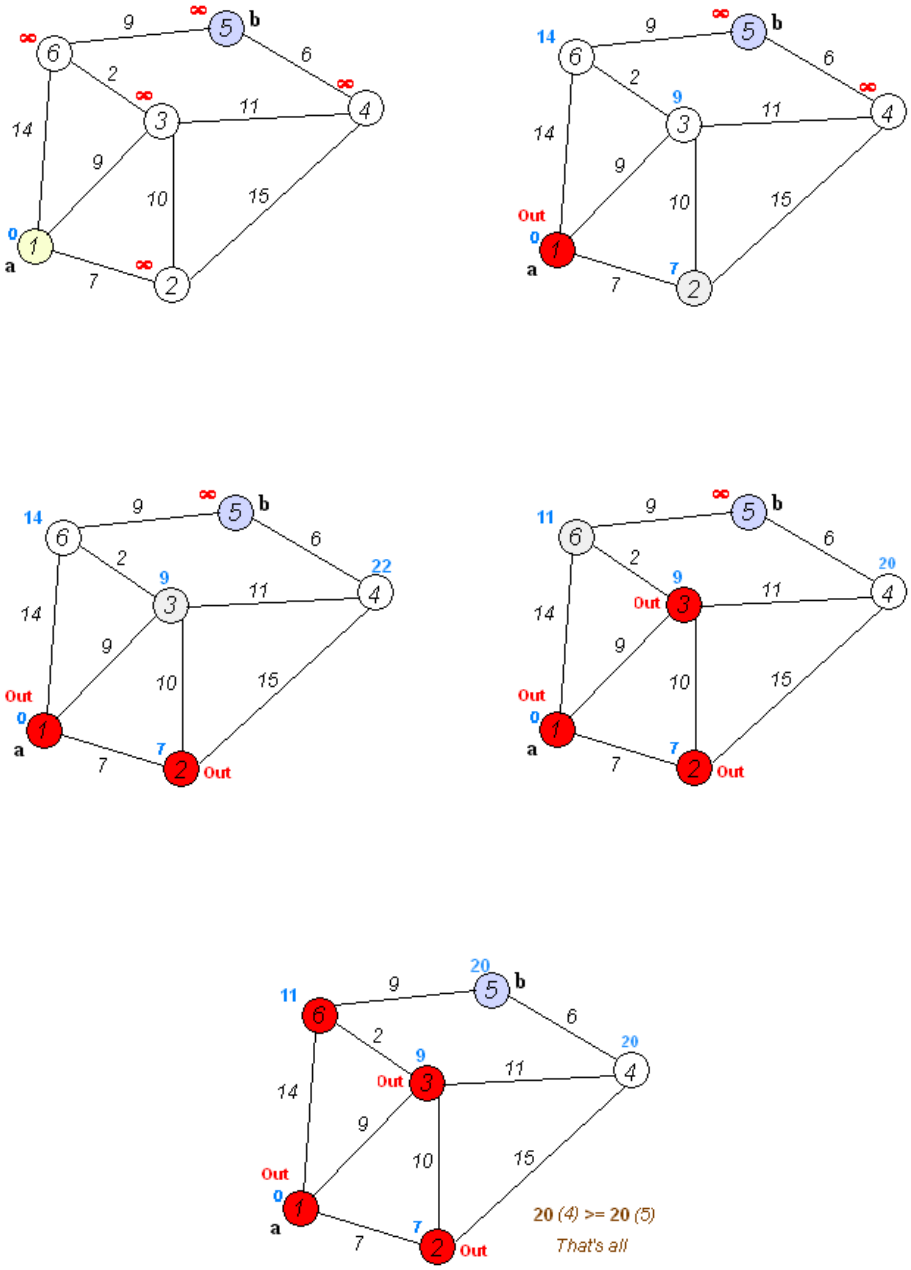


Figure 3.8: Example of a Dijkstra-Algorithm

3.5 Most probable future trajectories

In the previous section all the necessary elements of the Markov decision process were introduced.

- Ego-graph → Policies
- Cost functions → Reward function

However the reward function is introduced with one variable, the weightings of the cost function $w_{dist}, w_{dstr}, w_{obs}, w_{pers}$:

$$C_{total}(s_i) = w_{dist}C_{dist}(s_i) + w_{str}C_{str}(s_i) + w_{obs}C_{obs}(s_i) + w_{pers}C_{pers}(s_i) \tag{3.21}$$

to estimate them we have to solve the IRL-problem. Following the procedure introduced in the section(3.1.1) and the definition of reward function of section(3.3) we consider the value function V^{π_j} with the discount factor $\lambda \in (0, 1)$

$$V^{\pi_j}(s_0) = C_{total}(s_0) + \lambda C_{total}(s_1) + \lambda^2 C_{total}(s_2) + ...$$

by the fixed policies π_j extracted from the PEG and maximizes the difference of quality between the observed optimal policy π_p and other policies π_j:

$$\begin{aligned} max & \sum_{s_0 \in X_0} \sum_{j} (V^{\pi_j}(s_0) - V^{\pi_p}(s_0)) - \lambda w_i \\ s.t.\ & \lambda \geq 1, \\ & 0 < w_i \leq w_{max}, \\ & V^{\pi_j}(s_0) \geq (V^{\pi_p}(s_0) \end{aligned}$$

where $w_i, \quad i = 1, 2, 3, 4$ are the weightings of the cost function.

Therefore we can start with the estimation of a the most probable policies. Later on we will extend the method to a set of optimal policies.
To find the optimal policy we evaluate the value function for the policies of the PEG π_j with the optimal values for the weightings and search the minimum:

$$V^{\pi^*}(s_0) = \min_j \quad V^{\pi_j}(s_0)$$

where s_0 is the start state of the policy π_j and $V^{\pi^*}(s_0)$ is value function associated to the optimal policy π^*.

The estimation of a set of the most probable policies is simple. We must only eliminate the optimal policy from the set of value functions $\{V^{\pi_j}\}_{j=1...n_{opt}}$ after every calculation.

A set of n_{opt} optimal policies is presented in the following formula, where $V_{set}^{\pi^*}(s_0)$ is a set of n_{opt} optimal value functions:

$$V_{set}^{\pi^*}(s_0) = \{V_1^{\pi^*}(s_0) \qquad V_2^{\pi^*}(s_0) \qquad ... \qquad V_{n_{opt}}^{\pi^*}(s_0)\}$$

and the single elements of the set are defined as:

$$V_l^{\pi^*}(s_0) = \min_j V^{\pi_j}(s_0) \setminus \{V_m^{\pi^*}(s_0)\}_{m=1...l-1}, \qquad l = 1, ..., n_{opt}$$

The set of the most probable policies is:

$$\pi_{set}^* = \{\pi_1^* \qquad \pi_2^* \qquad ... \qquad \pi_{n_{opt}}^*\}$$

Chapter 4

Implementation

The aim is to create a human motion prediction algorithm with the following characteristics. Input values are the initial positions and velocities and target of all the pedestrians in the same frame. Furthermore, we need an image of the environment to extract the informations of the obstacles.
All this factors are used to generate the desired output value: a set of most probable pedestrian trajectories

The algorithm is separated in two significant parts. The first part is the offline part with the generation of a pedestrian ego-graph (PEG). In this section we describe the method of [1] and our improvements . Uniform randomly trajectories are generated respecting motion and human constraints.
A spatial clustering method separates the most probable trajectories in different partition-sets and a curve-cluster-algorithm is used to estimate a general description of each of them.
After the determination of the optimal number of clusters we store the obtained trajectories in the pedestrian ego-graph. An important note is ,that we create a new ego-graph for each different initial velocities of the pedestrians. This allows a more exact determination of the future trajectory and has no influence on the computational effort, because the generation is done offline.
In the online part, we show methods to reduce the ego-graph and define cost functions representing the spatial behavior of human in common environments.
These two contributions are combined by calculating a cost function for every trajectory of the ego-graph, to estimate the optimal trajectory.
We introduce also a method to characterize the targets of the pedestrians necessary to select only the trajectories oriented to a goal.

Every single cost function is weighted and to obtain human-like values we train the model with a real dataset. The Inverse reinforcement learning algorithm is implemented to solve the estimation of the optimal combinations of the weights.

4.1 Generation of the PEG

We use the ego-graph to store all the possible trajectories of a pedestrian. The creation is separated in the following steps:

- Probabilistic ego-graph
- Clustering method
- Positioning of the final PEG

In the first part we create a set of probabilistic trajectories. With the clustering algorithms explained after we accumulate trajectories in predefined partition-sets and estimate then the regression model for each of them.
The number of clusters for this method is not fixed and we introduce a method to estimate the optimal number. At this point we have a complete pedestrian ego-graph. The last step is to shift and orient the PEG in the desired initial position.

4.1.1 Probabilistic Ego-graph

The objective of the probabilistic ego-graph is to cover the area in front of a human. An origin is defined where all the trajectories starts. The generation is done with uniform randomized accelerations, but limited by different human and motion constraints, listed below:

- acceleration boundaries
- time constraint
- initial velocity

We create the trajectories described by a Constant Acceleration model:

$$p_x(t) = p_x(t-1) + (v_x(t-1) + \frac{1}{2}a_x(t)\Delta t)\Delta t \tag{4.1}$$

$$p_y(t) = p_y(t-1) + (v_y(t-1) + \frac{1}{2}a_y(t)\Delta t)\Delta t \tag{4.2}$$

These equations are used to generate all the trajectories of the ego-graph. The acceleration changes every 0.5 [s]. To create all the possible trajectories we use two uniform random values for the accelerations in the respective x and y direction. So at the end we obtain a Ego-graph covering all states in front of a pedestrian.

Pedestrian normally tend to maintain there velocity in magnitude constant. A maximum-velocity-limit is implemented to guarantee this characteristic, but with a

variance. We will include in our model also changes of the velocities over a trajectory. So we fix the limit at

$$v_{limit} = co + v_{initial} = co + (\sqrt{v_x(0)^2 + v_y(0)^2})$$

where $co = 0.6$ is a parameter determined from the real dataset described in section (A.2). The exact values are reported in the table(4.1). We evaluated the velocities of the different pedestrian trajectories and estimated a mean-variation of $0.6[m/s]$. We include no minimum constraint for the velocity, because it is a common behavior that pedestrian reduce their velocity to avoid obstacles.
A constant acceleration over the sampling period is assumed and permits to use the equations for constant accelerated motion.
The sample time is assumed to be 50 ms over a interval of 3 [s] (t_{simu}). Both values are chosen heuristically. The sample time has to guarantee a good balance, between a to fast sampling and high computational effort. A high sample rate induce very accurate values but a large data set which causes a slow algorithm. In contrast a to slow sample rate generates few information and not a high precision in the prediction but a faster computation.
The simulation time is set to 3 [s]. Longer simulations are not necessary, because the whole motion prediction is continuously updated.

Pedestrians velocities cover a broad range. In [14] is presented a listing of the range of human-walk-velocities: Elderly people walks normally with a velocity of $0.5[m/s]$ in contrast to younger persons having a maximum velocity of $2.4[m/s]$. Include all this velocities in only one ego-graph is difficult. For this and to attain a more precise prediction we generate a new Ego-graph for every different initial velocity. To cover all those velocities we decide to sample the interval with 0.2 [m/s] steps. The result are eleven different Ego-graphs.
It is not a problem to generated so much PEGs because they are calculated offline and so we have no increase of the computational effort.

Summarizing we have created eleven probabilistic Ego-graphs with different initial velocities. The number of trajectories is at the moment around 50000 for each Ego-graph. Obviously this are to much for a online algorithm, but we need them to implement our clustering algorithm described in the next section.
In the following table (4.1) are shown all the parameters for the different probabilistic ego-graphs.

4.1.2 Clustering

In this section we describe a method to extract the most realistic trajectories from the probabilistic ego-graph. At the end we want to have a reduced one with around 200 trajectories to allow a fast online scoring and estimation of the optimal trajec-

	PEG		**Cluster**		
$v_{initial}$ *[m/s]*	v_{limit} *[m/s]*	l_{max} *[m]*	*1.Layer [m]*	*2.Layer [m]*	*3.layer [m]*
0.4	1	2.48	0.3	0.54	0.96
0.6	1.2	3.23	0.43	0.81	1.53
0.8	1.4	3.91	0.58	1.08	2.04
1.0	1.6	4.47	0.75	1.41	2.61
1.2	1.8	5.03	0.94	1.66	3.17
1.4	2.0	5.69	1.09	1.93	3.69
1.6	2.2	6.27	1.25	2.21	4.27
1.8	2.4	6.93	1.41	2.48	4.81
2.0	2.6	7.55	1.56	2.76	5.40
2.2	2.8	8.12	1.72	3.04	5.95
2.4	3.0	8.68	1.73	3.31	6.55

Table 4.1: Probabilistic ego-graphs

tory. The theoretical fundamentals for the procedure are described in the section (3.2). To implement this procedure we follow the successive steps:

- Generate a set of partitions over three layers
- Create all the possible partition combinations
- Use mixture of regression models to estimate a general description for every partition-set (k variable)
- Determination of the number of clusters

Spatial-Clustering

This method is useful to select sets of realistic trajectories from the probabilistic-ego-graph. The range of the ego-graph is segmented in three layers with a constant radius. Each of them is then split another time in different partitions. The dimension and positions for those are token from [1]. In this paper they captured real trajectories from different dataset. Segmented them in pieces of 7-9[m] and shifted all of them in a common origin. A partition-constellation like described as before is overlapped. A set of three partitions from three different layers are called partition-set the trajectories are clustered in this different sets.. Only those one with a certain number of trajectories are considered the others are neglected.
Our ego-graphs has different dimensions. Therefore we use only the information about the proportion of the positions and the radius of the layers respect to the length of the trajectories. Analyzing the data, we estimated that the layers lie at 78% 44% 20% of the maximum length of the trajectories. Defined the spatial delimiter, we have partitions separated over three layers. The next step is to calculate

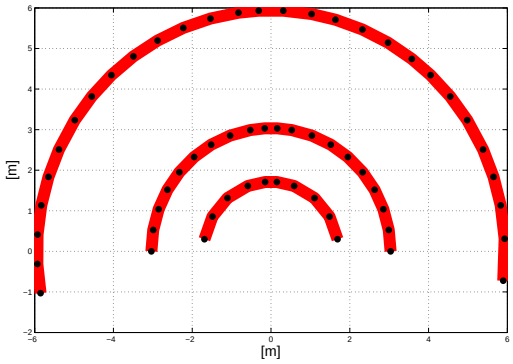


Figure 4.1: Intervals of the spatial-cluster

all the possible combination of partition-sets,a combination of three partitions each in a different layer. All the trajectories are associated to one of the partition-sets. In the next section we present a method to estimate a general representation for each partition-set.

Estimation of Regression Model

Reassuming we have the trajectories of the probability ego-graph separated in different partition-sets and will estimate now a general representation for every set. To perform this step we use the method introduced in section (3.2.2) and we obtain two coefficients for every partition-set (β_k, e_k). The first β_k is the regression coefficient and e_k is the covariance matrix. Combining this values over the simulation time, we estimate the x-and y-position for a trajectory at every time instant. For our implementation we have to extend the Eq.(3.13) because we have two dimensions, the x and y direction, and so the equation becomes:

$$\begin{bmatrix} y_j^{(1)}(i) & y_j^{(2)}(i) \end{bmatrix} = \begin{bmatrix} 1 & x_j(i) & x_j(i)^2 \end{bmatrix} * \begin{bmatrix} \beta_{k0}^1 & \beta_{k0}^2 \\ \beta_{k1}^1 & \beta_{k1}^2 \end{bmatrix} + \begin{bmatrix} e_k^1 & e_k^1 \end{bmatrix}$$

where $x_j(i)$ are the different discretization step of the simulation time $i = 1, ..., t_{simu}$ and $y_j^{(1)}$ is the x -value of the j-trajectory of the PEG and $y_j^{(2)}$ the respective y-direction.
It is important to annotated that we solve the problem for a fixed number of clusters $k = 1$ for every partition-set.
How it is possible to see in the figure(4.3) the trajectory obtained with the regression model is not a element of the PEG. To avoid this problem we have to estimate the median. Medians are representative objects of a data set whose average dissimilarity to all the objects in the cluster is minimal. The difference between the mean and the median is, that a median is always a member of the data set. In the figures below are presented the regression models and a PEG containing the respective medians.

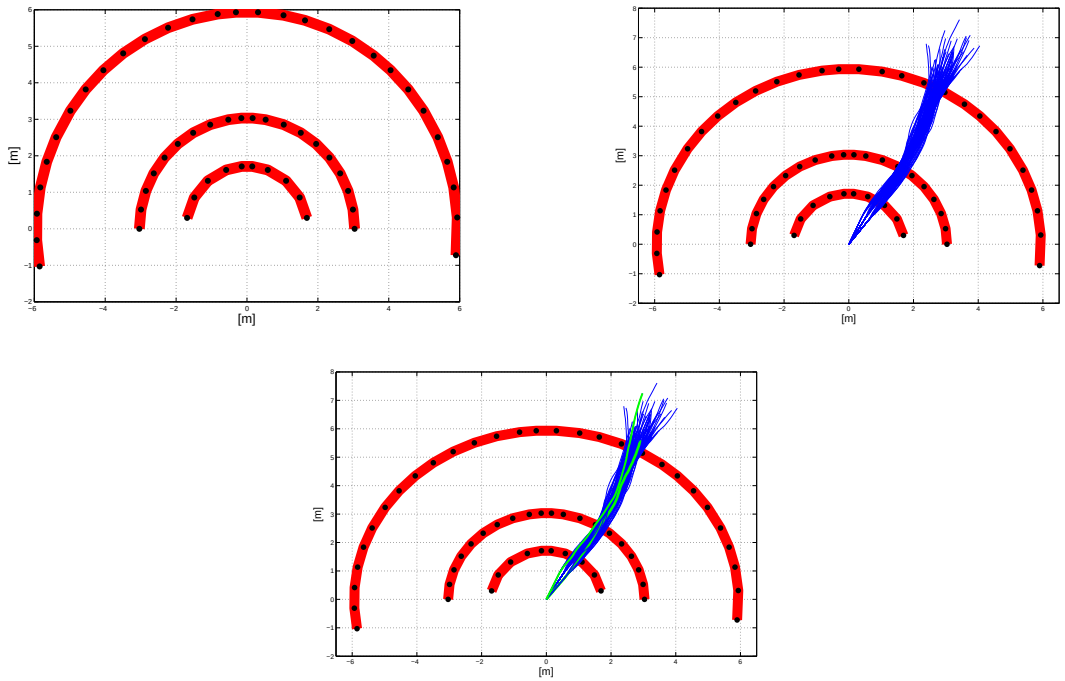


Figure 4.2: Spatial clusters (top left), collection of trajectories in a partition-set (top-right), regression model for a partition-set (bottom)

Determination Number of Cluster

As yet we have a ego-graph composed by the medoids estimated from the partition-sets with one cluster. Henceforth we will include in the ego-graph also the factor of how many trajectories belongs to one partition-set and the mean-squared error of them to the regression model.
The ideas is to augment the number of cluster for every set till two constraints are not satisfied.

- Mean-squared error between the trajectories and the regression model < 0.1 [m]
- Number of trajectories per set > 30

We use a kind of hierarchal clustering. Starting from a partition-set described by one cluster, we separate those in two parts and reestimate which trajectories belongs to which cluster. Then we check another time the constraints. If for example only the first subset exceeds the constraints then we stop the segmentation for this set, pass to the second subset and repeat the procedure.

The limits are determined heuristically. During the simulations we note that if the number of trajectories is lower then around 30, then single outliers influence a lot

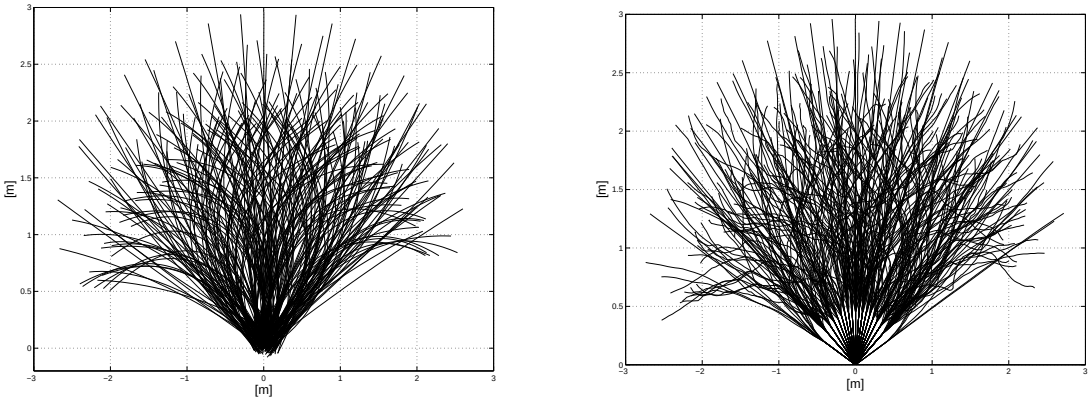


Figure 4.3: Regression-trajectories (left) and the PEG with median (right)

the regression model.
The mean-squared error is fixed at 0.1[m]. We calculated them as the mean over the squared error at each sample-time. The value selected guarantees a covering of the space in front of the pedestrian with a new trajectory every 0.1[m]. As we see in the following the PEG is enough exact to produce good results.

At the end when all the partition-sets are separated and all the limits exceeded, we estimate the medians and store them in the final pedestrian ego graph. Obviously this procedure is repeated for all the different ego-graphs.

4.1.3 Positioning of the ego-graph

Before we can weight the trajectories of the PEG $s = (p_x, p_y)$ and estimate the optimal one, we must collocate the PEG in the environment. The input values position $p_{intitial} = (p_{x_{initial}}, p_{y_i nitial})$ and velocity $v_{initial} = (v_x, v_y)$ allow to fine the position and orientation in the environment.
The translation is the simple part, we sum only the value of the initial position to all the states of the PEG. However the orientation is extracted form the initial velocity. From the x-and y-value we obtain:

$$\theta = \arctan(\frac{v_y}{v_x})$$

The inverse tangent function is defined $\arctan : \mathbb{R}^2 \to (-\frac{\pi}{2}, \frac{\pi}{2})$ but we need a angle over $[0, 2\pi]$. We divide the definition space in to parts. If the velocity in the x-direction is negative we sum π to the angle:

$$\theta = \begin{cases} \theta, & \text{if } v_x \geq 0 \\ \theta + \pi, & \text{if } v_x < 0 \end{cases}$$

we must sum π to the angle if the x-velocity < 0.
To rotated the pedestrian ego-graph around this angle θ we introduce the notation of rotation matrix for the euclidean space:

$$R = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix}$$

where θ indicates the angle of orientation of the pedestrian.
Combining the information of rotation and translation we obtain the final positions for the PEG:

$$\begin{bmatrix} p_{x_{rot}} \\ p_{y_{rot}} \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} p_x \\ p_y \end{bmatrix} + \begin{bmatrix} p_{x_{initial}} \\ p_{y_{initial}} \end{bmatrix}$$

and define the new state $s = (p_{x_{rot}}, p_{y_{rot}})$ utilized in the resting sections.

A special case is $v_{initial} = 0$, a stationary pedestrian. The equations are still valid and we obtain:

$$\begin{bmatrix} p_{x_{rot}} \\ p_{y_{rot}} \end{bmatrix} = \begin{bmatrix} p_{x_{initial}} \\ p_{y_{initial}} \end{bmatrix}$$

but lost the information about the direction of the pedestrian. To avoid this problem, we treat the pedestrian like an obstacle. A pedestrian is modeled as a circular shape, like described in the section(configuration space). The pedestrian is then included in the obstacle map as a static obstacle.

4.2 Goal Analysis

Every pedestrian is always directed to a goal.We want to use this characteristic of the human motion to reduce the ego-graph, to decrease the computational effort.
In this thesis we assume always that a goal is available. This input data is extracted from the environment. For example a pedestrian on a sidewalk normally wants to go straight or he enters in some door. Different techniques of computer vision evaluates this information and determine a target.
The idea is to cluster the trajectories around the goal inside the ego-graph.
The problem is that very often a goal is behind an obstacle or farther away then the maximum range of the pedestrian ego-graph. All the possible sceneries and the necessary solution steps are listed in the understanding schedule:

- Goal inside the Ego-graph: Clustering around this goal
- Goal is farther away:
  - (1) Between goal and pedestrian is no obstacle:
    - We calculate a line between the target and the origin of the pedestrian
    - Draw a circle with a radius close to the maximum range of the ego-graph
    - Calculate the intersection and use the solution closer to the target
    - Cluster the trajectories around this subgoal
  - (2) Between goal and pedestrian is an obstacle:
    - Create a visibility graph between target and origin
    - Calculate the minimum path between origin and target with Dijkstra
    - Then we check if the first subgoal of the set is inside or outside of the ego-graph. If it's inside we cluster directly around the subgoal, otherwise we invoke the previous step(1).

After the solution of all this functions and issues we obtain first a subgoal inside the ego-graph and then a reduced ego-graph utilized in the following sections to solve the online algorithm. Below we explain the implemented techniques to solve this problem and all the special cases which we found on the way.

4.2.1 Goal-Clustering

The first step is to present a clustering algorithm to select the trajectories directed to a goal. We assume to be in the situation with the goal (or subgoal) inside the ego-graph.
The algorithm is based on a cell around the subgoal. The dimension depends on the maximum length and so from the initial velocity of the ego-graph. It is defined at 0.2[m] for a initial velocity of 2 [m/s]. In general we fix it at:

$$r_{cell} = v_{PEG} * 0.1 \quad [m]$$

Then we start a research of all the trajectories of the PEG which belongs to the cell. Those were saved in a new reduced PEG and are the base for all the considerations in the following sections.

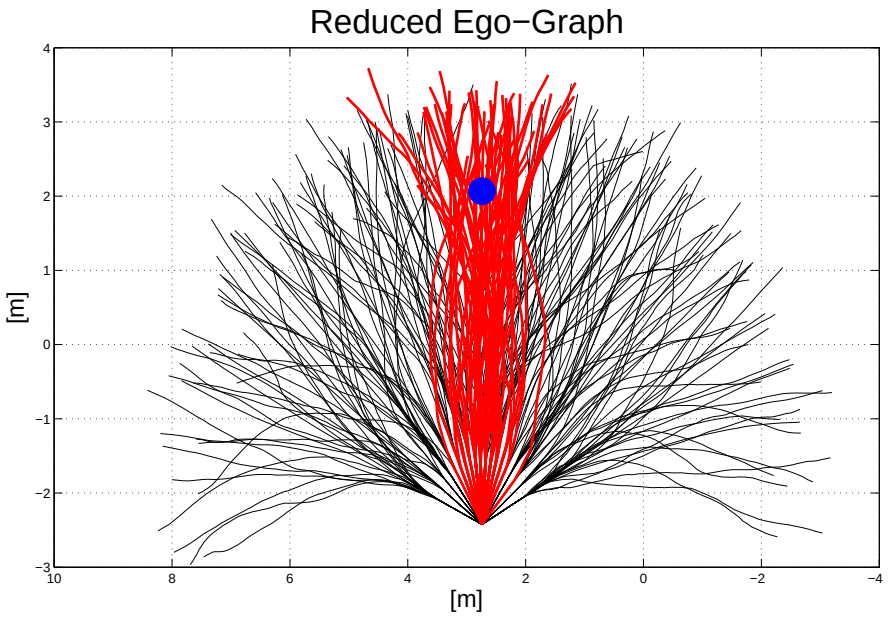


Figure 4.4: Reduced Ego-Graph

4.2.2 Generation of Configuration Space

In the theory part we presented the definition of configuration space. We recall that it determine the area where a pedestrian can pass without a collision with static obstacles. In other words is the sum of the pedestrians area and the obstacle area. In robotics a pedestrian is modeled with a circular space with radius around $0.36[m]$, but after different simulations we noted that pedestrian in the used dataset pass much closer obstacles. For this we decide to fix the value at $0.15[m]$ A solution for this issue is the implementation of a padding algorithm. It is the realization of the Minowski sum between the pedestrian shape and the obstacle space. The obstacle space is obtained with a "border search"-algorithm and a corner-detection.

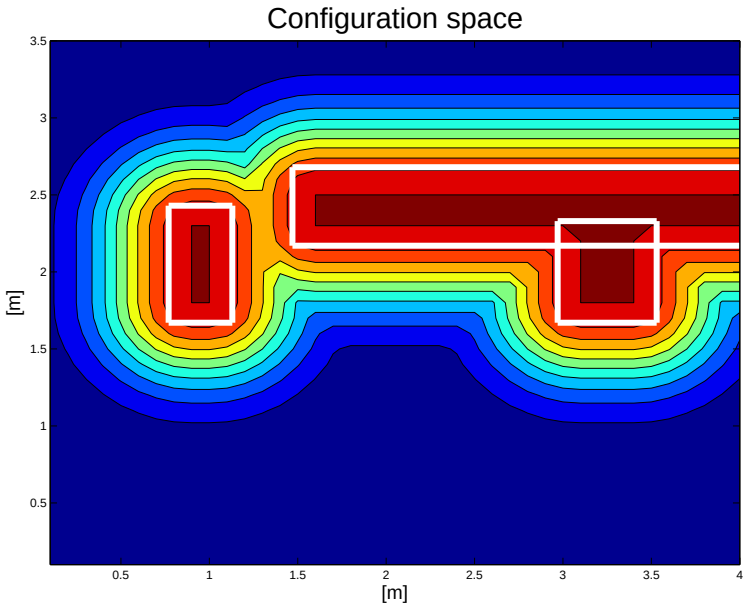


Figure 4.5: Configuration Space

4.2.3 Localization of the Goal

The objective of this section is to determine if a goal lies in or outside the ego-graph. The ego-graph has a specific form and if we generate a convex hull of only straight lines around the ego-graph, the resulting geometric figure is a rhombus and represents our constraint.
To determine if a goal is inside or outside this constraint, we generate a Delaunay triangulation between all the corners of the constraint and the goal. In the case that the goal is inside we have 4 edges laying in the inner part of the ego-graph. Otherwise some of this are outside. For every goal we check if this condition is satisfied. The following two images (Figure(4.6)) shows the two cases:

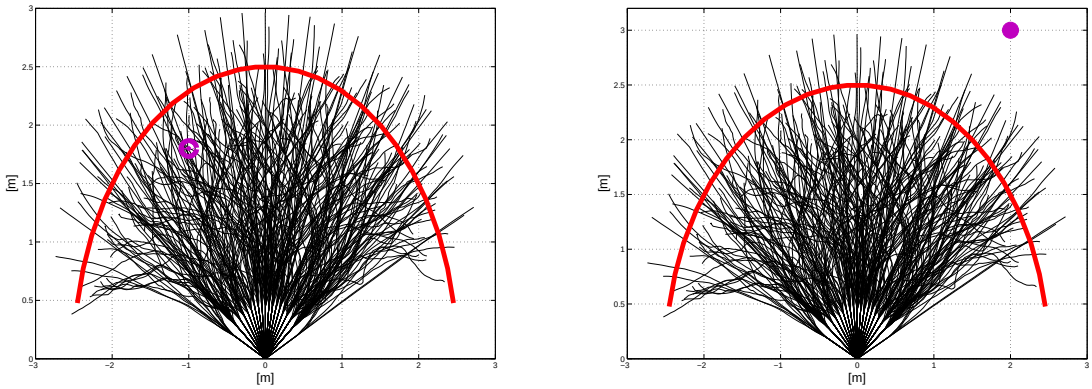


Figure 4.6: Location of the goal: In the PEG (left) or outside of the PEG (right)

4.2.4 Visibility-check for the Goal

Checked if a pedestrians target lies in or outside we pass to the second case. Is the goal visible from pedestrians origin?
An object, a position or a person is visible if on the line of sight between the origin and the other element are no obstacles.
To verify the visibility we use the Delaunay triangulation. We implement this function over the corners of the configuration space, the origin of the PEG and the target. Then we calculate the minimum-cost-path from the origin to the target with the Dijkstra-Algorithm. If the distance is equal to the length of the line of sight, then we know that the goal is visible. In other words there is no corner which forces the minimum path to avoid a corner and so an obstacle in the middle.

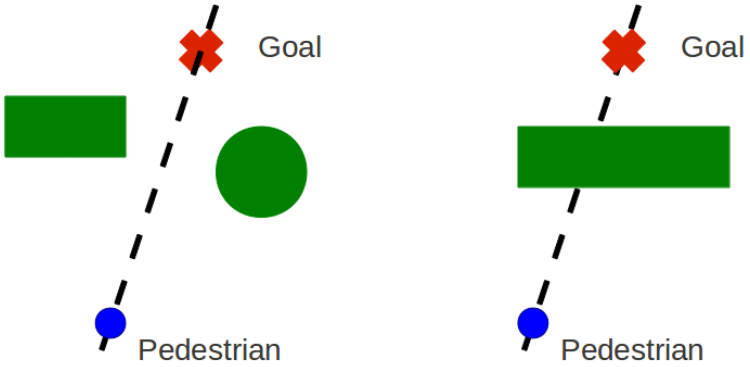


Figure 4.7: Visible and Invisible goal

Visible Goal

In the case of a visible goal we have only to shift it over the line of sight in the ego-graph.
The position of the subgoal is the intersection point between the line of sight

$$y = mx + q \tag{4.3}$$

$$m = \frac{y_{target} - y_{origin}}{x_{target} - x_{origin}} \tag{4.4}$$

$$q = y_{origin} - \frac{y_{target} - y_{origin}}{x_{target} - x_{origin}} x_{origin} \tag{4.5}$$

and a circle drawn around the origin.

$$(x - x_{origin})^2 + (y - y_{origin})^2 = r^2$$

The value for the radius is fixed at 77% of the maximum range of the ego-graph.
The solution for the intersection is a quadratic equation. The start equations are:

$$(x - x_{origin})^2 + (y - y_{origin})^2 = r^2 \tag{4.6}$$

$$y = mx + q \tag{4.7}$$

and at the end we obtain two solutions. Of interest is only the solution between the target and origin the other one is neglected. The solution is our subgoal and used to reduce the ego-graph with the clustering algorithm described in section(4.2.1).

Invisible Goal

A invisible goal, is one behind an obstacle. The theory presented in the section(3.4.2) allows to solve this problem.
The first step is to generate the triangulation between all the corners of the obstacle in the configuration space, the origin of the pedestrian and the target. Also the triangulation inside of a obstacle are created.
We introduce the information of the form of the obstacles and neglect all the edges inside or crossing those and obtain all the visible edges.
The last step is the implementation of the Dijkstra algorithm. The shortest path is found and we have a ordered set of all the visible corners necessary to pass to reach the goal.
We extract the second node of the estimated path. For the structure of the implementation we are sure that exists a visible edge between this point and the origin.
At this point we are again in the situation for visible goal. Repeating the check of localization of the subgoal explained in the previous paragraph we estimate a subgoal inside the ego-graph.
In the following image is shown a invisible goal and applied solution procedure.
An image with all the different steps of this implementation is shown in section (5.1.2).

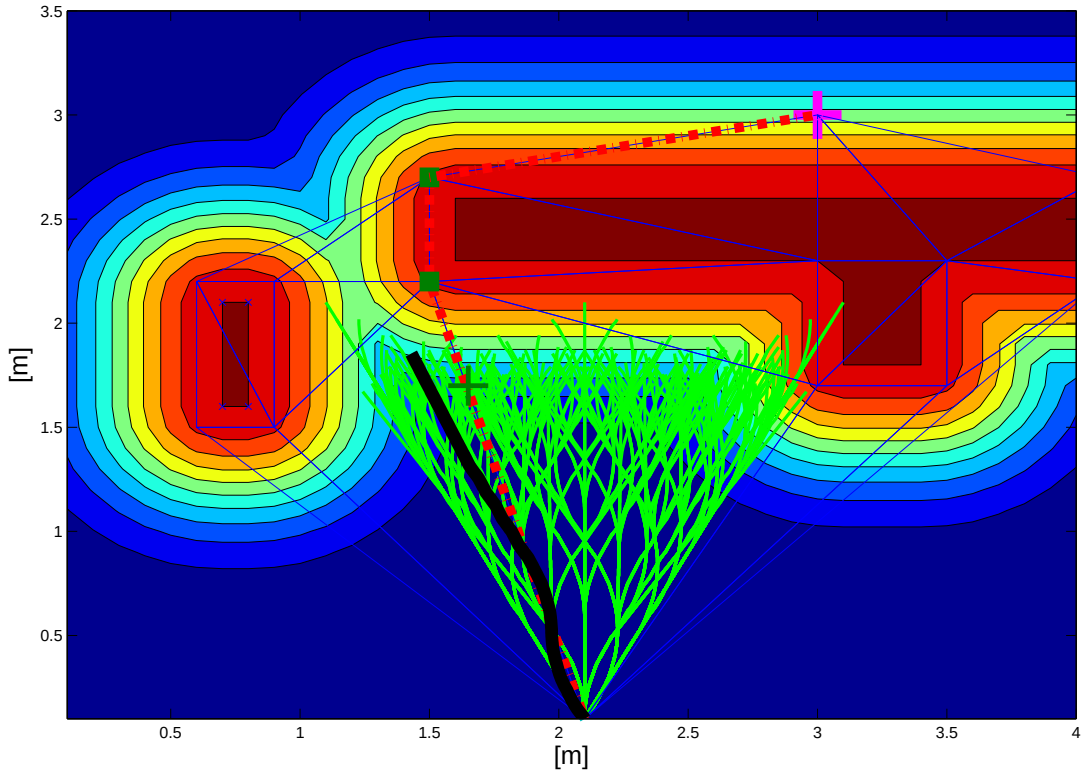


Figure 4.8: Invisible goal and the proposed solution (green points are the subgoals)

4.3 Cost function

The next step for our procedure is to define cost functions to describe the human behavior. We implement the four functions defined in the section (3.3), and calculate the value for every state of the MDP.
We prevent some assumption before the implementation is introduced. The costs are calculated separately for every observation. The total cost for each is then estimated as a weighted sum of the four elements.

$$C_{total}(s_i) = w_{dist}C_{dist}(s_i) + w_{str}C_{str}(s_i) + w_{obs}C_{obs}(s_i) + w_{pers}C_{pers}(s_i) \tag{4.8}$$

The functions are described in detail in the following subsections.

4.3.1 Distance function

The objective is to calculate the distance between two sequential states of the ego-graph. We use the definition of the euclidean distance.

$$C_{dist}(s_i) = \sqrt{(p_x(i) - p_x(i-1))^2 + (p_y(i) - p_y(i-1))^2}$$

We can calculate this cost function offline, during the generation of the ego-graph. This characteristic implicates a reduced computational effort and allows to calculate the next cost function.

4.3.2 Steering function

We recall the definition of section 3.3.1:

$$C_{str}(s_i) = (str(s_i) - str(s_{i-1}))^2 \tag{4.9}$$

$str(s_i)$ is the angle between the states s_{i+1} and s_i, to estimate this value, we use the definition of the cosine-function.
The triangle, in figure 4.9, is defined by two observations and the resulting point to generate a right-triangle. The hypotenuse is the distance between the two states s_{i+1} and s_i. The adjacent leg is represented by the difference of the x-values of the states $p_x(i+1)$ and $p_x(i)$.
Applying the inverse cosines-function the steering-value $str(i)$ is obtained.
The same operation we repeat for the states s_i and s_{i-1} to estimate $str(i-1)$. The steering difference is computed subtracting $str(i)$ from $str(i-1)$ and square it, because it is not important to know the direction but only the amplitude of the angle.

$$str(i-1) = \arccos\left(\frac{p_x(i) - p_x(i-1)}{dist(i)}\right) \tag{4.10}$$

$$str(i) = \arccos\left(\frac{p_x(i+1) - p_x(i)}{dist(i+1)}\right) \tag{4.11}$$

$$\Longrightarrow C_{str}(s_i) = (str(i) - str(i-1))^2 \tag{4.12}$$

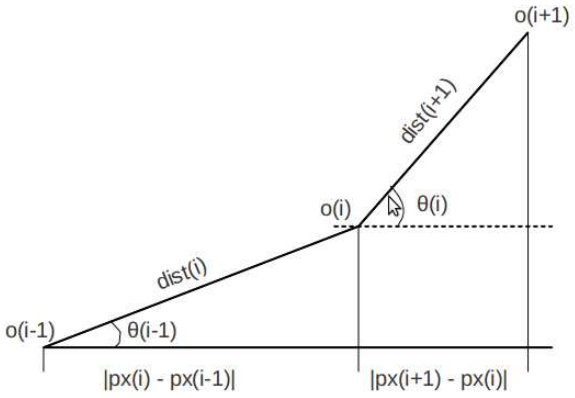


Figure 4.9: Steering calculation

4.3.3 Obstacle function

This is the most important cost function. Is composed by two parts. One is the function obtained by static obstacles and the other by the dynamic, which are the other pedestrians in the scene.

Static obstacle

The implementation is based on a distance map of the current environment. From an image we extract the dimension of the map and the position of the obstacles the exact procedure is described in Appendix A.2.
Then we use an algorithm to calculate iteratively the distance for every cell to the closest obstacle. We evaluate the cells, beginning by the border of the obstacles and insert a new distance value, only if the it is smaller that the previous one [9]. The algorithm works only for convex obstacles.
Estimated this grid map we calculate the obstacle function with the following equation:

$$C_{obs}(s_i) = \exp(-0.5\frac{dist(s_i)^2}{\sigma_d^2}) \exp(-0.5\frac{dist(s_i)^2}{\sigma_w^2})$$

The first term is the mathematical description of the comfortable distance of a person. Pedestrians keep always a certain distance to objects. This behavior is specified in this exponential function with the parameter $\sigma_d = 0.361[m]$. The value is found empirically from [2]. They trained this amount over more then 600 tracks of pedestrians in different scenes. The $dist(s_i)$ is the distance of the person in state s_i to the closest obstacle and extracted from the distance grid map.
The second term simulates the radius of influence of an obstacle. Farther then $\sigma_w = 2.088[m]$ the human behavior is not influenced by an obstacle.

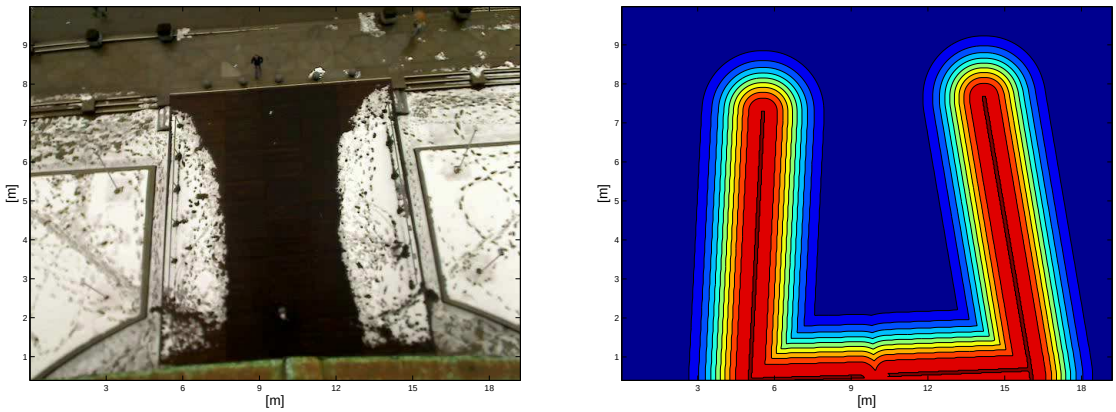


Figure 4.10: Obstacle map and the real environment

Person obstacle

The theory for this cost function is defined in section (3.3.2). In [5] they analyzed the distance between walking persons in indoor and outdoor places, between different gender and different ages. The results are reported in the following table:

	Indoor			**Outdoor**		
Sex Combination	*Adult*	*Teenage*	*Child*	*Adult*	*Teenage*	*Child*
M-M	83	83	63	83	75	62
M-F	71	63	58	79	68	58
F-F	75	62	59	75	73	67

We are interested only in outdoor places. The problem is that the determination of the gender and the age of pedestrians in open space is difficult. So we decide to calculate the mean over all this values.

$$dist = 0,71[m] \Longrightarrow \sigma_{xx} = 0.356[m] \quad and \quad \sigma_{yy} = 2\sigma_{xx}$$

4.4 Estimation of the Optimal trajectories

Finished all the previous operations, we have selected a set of trajectories from the ego-graph directed to a goal. All the trajectories are described with the exact coordinates and the associated cost functions. To estimate the most probable trajectories we have to estimate the trajectories with the lowest cost function.
The first step is to sum the four cost values for every state s_i

$$C_{total}(s_i) = w_{dist}C_{dist}(s_i) + w_{str}C_{str}(s_i) + w_{obs}C_{obs}(s_i) + w_{pers}C_{pers}(s_i)$$

To estimate the cost function over a whole trajectory we calculate the value function:

$$V^{\pi}(s_0) = C_{total}(s_0) + \lambda C_{total}(s_1) + \lambda^2 C_{total}(s_2) + ... \tag{4.13}$$

$\lambda \in (0, 1)$ is a discount factor and has the mission to weight the costs over time. It means that a cost function far away, and not only in distance sense but also in time sense, is less important then the value in the surrounding of the person.
The best trajectory is defined as the minimum value function of all trajectories:

$$V^{\pi^*}(s_0) = \min_j \quad V^{\pi_j}(s_0)$$

and $V^{\pi^*}(s_0)$ is the optimal cost function of the most probable trajectory π^* of the pedestrian.

For a set of most probable trajectories the procedure described in the section 3.5 is implemented. An algorithm estimates the most probable trajectory, eliminates those from the list and the procedure is repeated. At the end we obtain a set of the most probable trajectories.

4.5 Inverse Reinforcement Learning

In this section we describe the method used to optimize the weights of the different cost functions. The dataset utilized for the realization is the University-video described in the Appendix A.2.

The theory on which is based the algorithm is described in section 3.1.
For the solution of this linear program problem we adapt a iterative approach.
We fix a range for the five unknown parameters. The four weightings of the cost function and the discount factor λ:

w_i	w_{max}	w_{min}	$w_{optimal}$
w_{dist}	10	0.01	0.6
w_{str}	10	0.01	1.2
w_{stat}	10	0.01	7.1
w_{move}	10	0.01	3.4
λ	1	0.85	0.98

The idea is to estimate the optimal combination of these five factors. The presented solution below is based on the IRL-method and the objective is to solve

$$max \sum_{s_0 \in X_0} (V^{\pi^*}(s_0) - V^{\pi^{real}}(s_0)) - \lambda w_i \tag{4.14}$$

$$s.t.\ \lambda \geq 1, \tag{4.15}$$

$$0 < w_i \leq w_{max}, \tag{4.16}$$

$$V^{\pi_j}(s_0) \geq (V^{\pi_p}(s_0) \tag{4.17}$$

We separate this procedure in different steps. First we estimate the most probable trajectory $\pi^*(s_0)$ with the prediction algorithm presented in the previous sections. The optimal value function and so the reference function of the IRL-procedure is the value function calculated over the real trajectory from the dataset $\pi^{real} = \{st_0, st_1, ...st_t\}$:

$$V^{\pi^{real}}(s_0) = C_{traj}(s_o) + \lambda C_{traj}(s_1) + \lambda^2 C_{traj}(s_2) + ... + \lambda^t C_{traj}(s_t) + ...$$

where

$$C_{traj} = w_{dist}C_{dist} + w_{str}C_{str} + w_{obs}C_{obs} + w_{pers}C_{pers}$$

Then we start with the estimation of all the value function V^{π_j} of the trajectories π_j stored in the PEG

$$V^{\pi_j}(s_0) = C_{traj}(s_o) + \lambda C_{traj}(s_1) + \lambda^2 C_{traj}(s_2) + ... + \lambda^t C_{traj}(s_t) + ...$$

and calculate the most probable trajectory of the PEG.

$$V^{\pi^*}(s_0) = \min_j \quad V^{\pi_j}(s_0)$$

To compare the two value functions

$$(V^{\pi_j}(s_0) - V^{\pi^{real}}(s_0))$$

we decided to use a indirect calculation. We do not calculate explicit the value function for the real trajectory, but we estimate the euclidean distance at the same time instant between the trajectory π^{real} and the most probable trajectory π^* and compare the mean-squared error for each state s of the trajectory at the same time instant.
This procedure is repeated for each trajectory of the dataset and solve

$$\max \sum_{s_0 \in X_0} \sum_{j} (V^{\pi_j}(s_0) - V^{\pi^{real}}(s_0))$$

Summarizing we have now estimated an error representing the whole dataset for one combination of the weightings. To determine the optimal combination we have to repeat this method for all the possible combinations. The optimal combination is then those with the minimum error over the whole dataset.

When we calculate the error between the real and the PEG trajectory we consider the relative error at every time instant and not the absolute one. Otherwise one single factor is amplified to much.

4.6 Update-algorithm

In this section we describe a first introduction about how it is possible to re invoke the prediction-algorithm at constant updates intervals.
For the experiments we fixed the call time at 0.4[s], because the states of the dataset are given at this time steps. We invoke the algorithm, start the normal prediction procedure. At the next call we recalculate with the new initial position and velocity, the most probable trajectories. Obviously all the considerations about the environment and the pedestrian in the same scene are updated.

In the section (5.2) is shown how the error, diminish rapidly. But this is not the main objective of this implementation. Because it is important to deliver a prediction over a longer range of time. This information is then used to improve the performance of the robot navigation. If we are able to predict exactly the trajectory of the pedestrian over some meters, the robot can start the avoidance of the pedestrian much earlier, as if the pedestrian is considered as a static obstacle or as a constant velocity model without any considerations about the environment and other pedestrian in the scene.

Some considerations for the implementation are listed in the following :

- Update only if the error exceeds a certain error limit
- Update only the changes of the environment map [10]
- Set the update time in base of the environment (crossings shorter times then on sidewalks)

Chapter 5

Simulation and Results

In the following two sections we present the evaluation of our human motion prediction. In the simulation part we show visually the predicted trajectories in different real scenes.
In the result part is introduced the procedure followed to compare our implementation with other approaches.

5.1 Simulation

In this section we present the simulations realized in Matlab. All the functions utilized to solve the problem are presented before. The objective is to show the improvements and differences of our implementation visually.
In the first part we show all the necessary steps of a standard prediction in a real image.
Later on we present special scenes like collision avoidance or a situation in which our algorithm decided a wrong direction.

In the appendix are presented simulations where we show how the different cost functions work and extend the case to a multiple set of goals.

5.1.1 Steps of the Algorithm

In this first simulations we show the different steps of our prediction algorithm.
The algorithm starts with the initialization of the pedestrian ego-graph. From the dataset we know exactly the position and orientation of the pedestrian, those trajectory we want predict.
Another decision token at the beginning is the dimension of the PEG and is based on the initial velocity of the pedestrian.

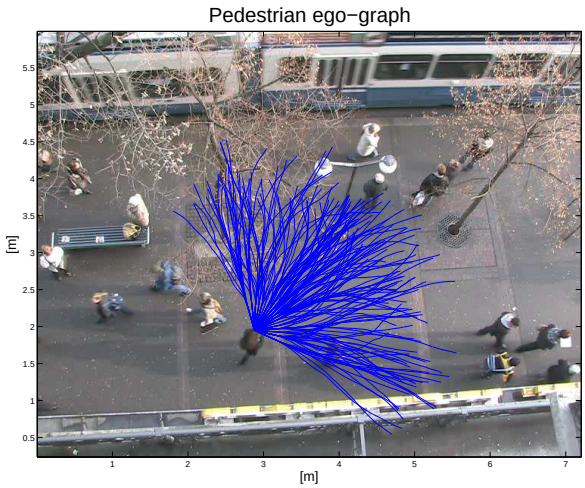


Figure 5.1: Initialization of the Pedestrian Ego-Graph

After the initialization we reduce the ego-graph. The goal for the pedestrian is known and we have only to shift it into the ego-graph. The special case, that the goal is invisible is shown in section (5.1.2).

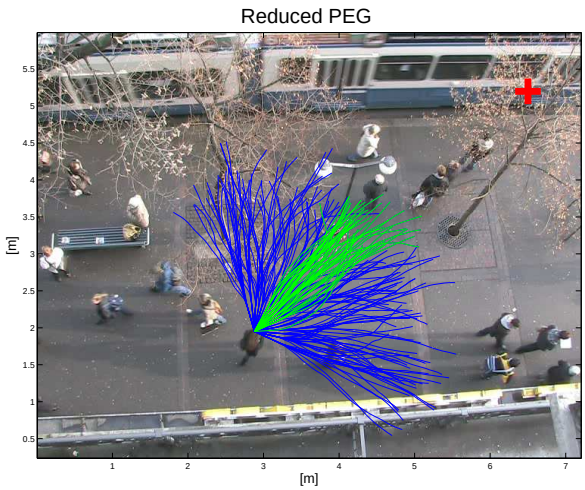


Figure 5.2: Selection of the trajectories oriented to the target

At this point we invoke the online algorithm to select the most probable trajectories of the reduced ego-graph in base of the different human behaviors.

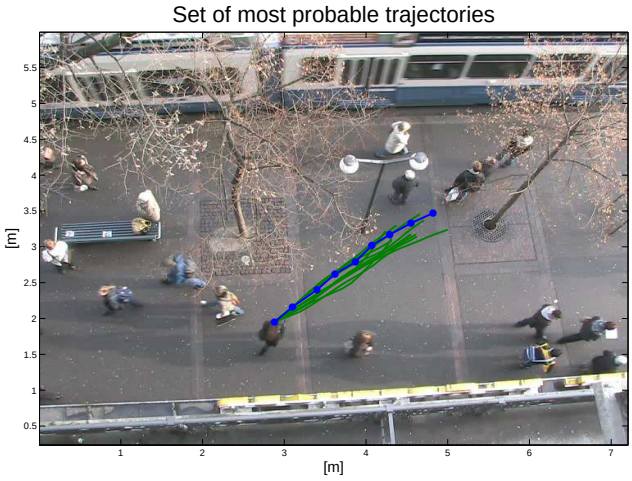


Figure 5.3: Set of most probable trajectories

The last step is the estimation of the most probable trajectory and compare it with the real trajectory from the dataset.

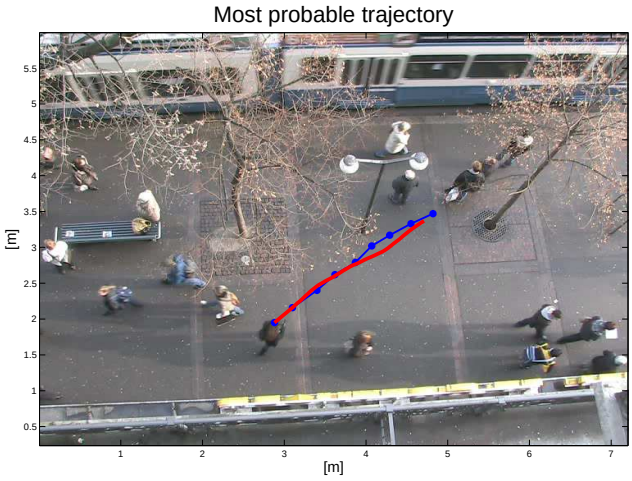


Figure 5.4: Most probable trajectory

5.1.2 Invisible goal solution

In our implementation we pretend always to know the goal for a trajectory. Sometimes it is possible that the goal is invisible. We recall that a goal is invisible if between the origin of the pedestrian and the goal is an obstacle and the obstacle is out of range of the ego-graph.
If we are in this case we follow the procedure described in section (4.2).
The obstacle is out of the maximum range of the PEG and so it has no influence of the prediction in our algorithm. A wrong prediction is the result:

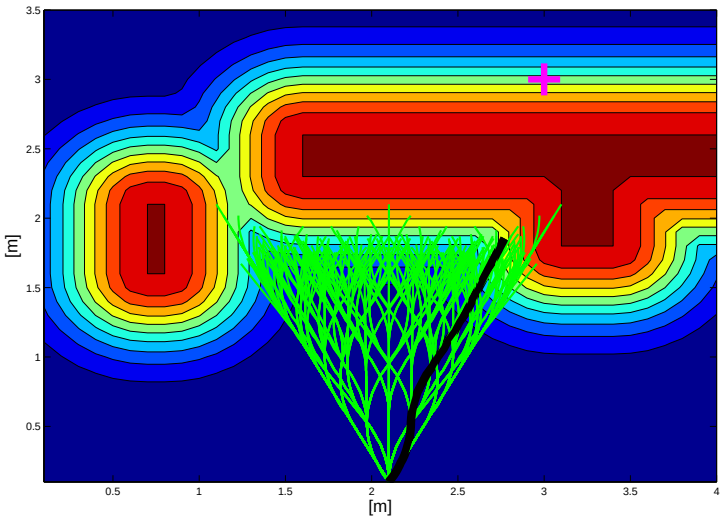


Figure 5.5: Erroneous prediction

To solve the issue, we define first the configuration space between pedestrian and the obstacles of the environment. In the following we invoke the Delaunay triangulation

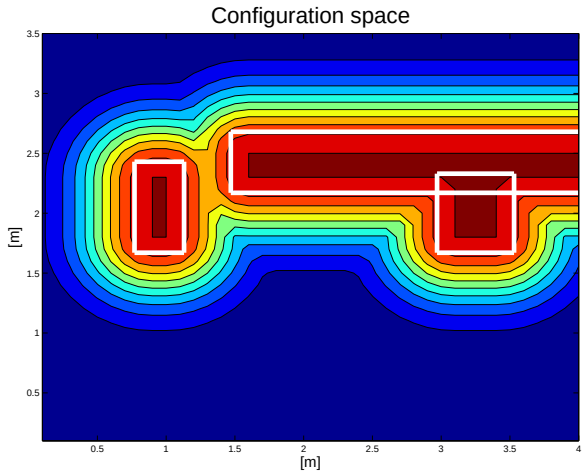


Figure 5.6: Location of the goal: In the PEG (left) or outside of the PEG (right)

and obtain all the visible edges between the origin of the pedestrian and the target. The Dijkstra algorithm estimates the minimum path and so the set of subgoals to follow. We extract the first and shift it in to the PEG. At this point the normal prediction algorithm is called and the most probable trajectory is estimated.

It is important to note, that our algorithm is the unique to take in consideration

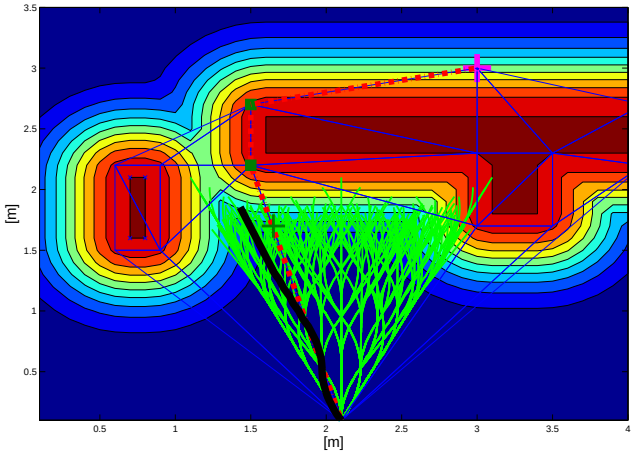


Figure 5.7: Location of the goal: In the PEG (left) or outside of the PEG (right)

this case. All the other implementations, not only the simple like constant velocity or goal directed, but also [1] and [2] fails and produces a wrong estimation.

5.1.3 Special cases

In this section we show different cases, where our approach generates better or in some case also worse results.
Our algorithm is able to avoid early pedestrians (red) and simulate well human behaviors. The constant velocity model (white) has the drawback to ignore all the human behavior and steers straightly in a collision. Sometimes the implemented

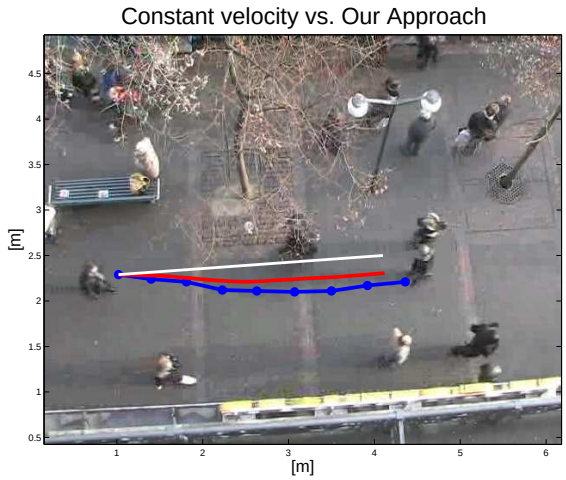


Figure 5.8: Constant velocity vs. Our approach

prediction produces wrong estimations. For example if we consider pedestrians walking in groups. The cost function of comfortable distances generates always elevated values and force the pedestrians to disperse.
By way of comparison, the constant velocity model predict the trajectory exactly.

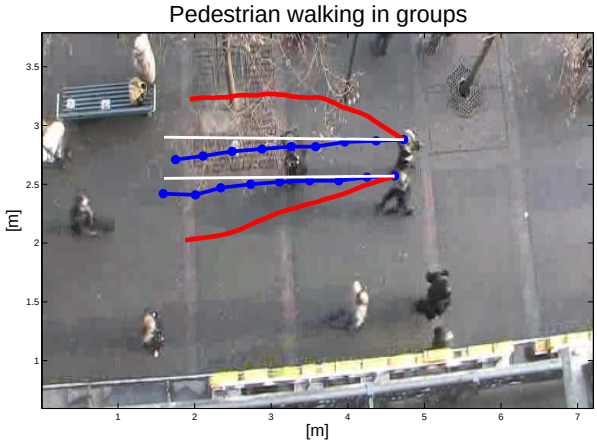


Figure 5.9: Pedestrian walking in groups

Safely collision avoidance is one of the most important features of our prediction. In this figure we show, how not only the constant velocity model (white) produces a large error, but also the goal oriented algorithm (magenta). The two last mentioned algorithms do not consider any human behavior, neither they consider the surrounding environment and finish so to collide with and obstacle or to estimate a completely wrong trajectory.

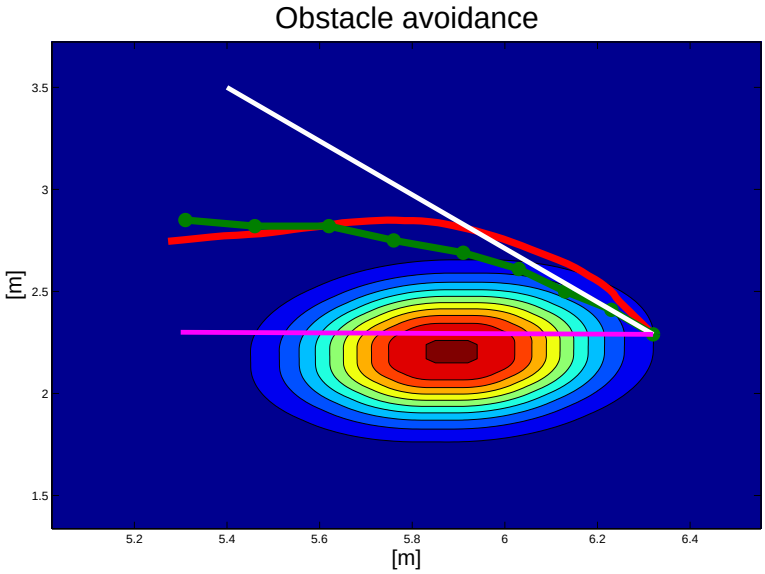


Figure 5.10: Static obstacle avoidance

5.2 Results

5.2.1 Confront with State of the Art

The analysis of the performance of our algorithm is based on the estimation of the prediction error between real trajectories and our estimation. The results are compared with other methods like the Constant Velocity model (CV), the Linear Trajectory Avoidance [2] and the Pedestrian Ego-graph model (PEG) [1]. The values of all the other implementations are taken from [1]. In the following we explain how we evaluate the predicted trajectories.
First it is important to notice that the evaluation is realized on a second dataset, the video showing the scene in front of the Hotel, and not on the same as used for the trainings procedure.

The idea is to calculate the absolute error between the real trajectory of the dataset and the respective predicted pedestrian trajectory with the optimal weightings combination. Therefore we synchronize the initial position and velocity of the different trajectories with the prediction algorithm. The algorithm is started and we repeat this procedure for each trajectory of the dataset. It is important to calculate always the distance between positions at the same time instant. The time information of the dataset was extracted like described in the section (Appendix A.2).

To have a comparison with [1] we must estimate the mean squared error of the prediction respect to the real trajectory at a fixed metric interval.
We compare the results at $1-2-3-4[m]$. The error is calculated over 80 randomly choosen trajectories from the dataset. It is important to note, that our approach and the PEG [1] consider not only the optimal trajectory, but the trajectory, from a set of (10) most probable trajectories, with the smallest error.

Method	*1 [m]*	*2 [m]*	*3 [m]*	*4 [m]*
CV	0.037	0.122	0.221	0.363
LTA [2]	0.054	0.128	0.212	0.285
PEG [1]	**0.030**	0.074	0.100	0.105
OURs	0.031	**0.072**	**0.097**	**0.099**

Table 5.1: Statistical results: Average error

Method	*1 [m]*	*2 [m]*	*3 [m]*	*4 [m]*
CV	0.042	0.155	0.259	0.388
LTA [2]	0.043	0.092	0.145	0.204
PEG [1]	**0.026**	0.057	0.067	0.083
OURs	0.028	**0.055**	**0.063**	**0.079**

Table 5.2: Statistical results: Standard deviation

The improvement of our approach respect to the [1] is based on the introduction of different pedestrian ego-graphs for different initial velocities. This implementation method allows to generate more specific trajectories for each situation. For example the ego-graph in [1] is composed by 248 trajectories. We generate a PEG of 180 trajectories for each initial velocity.

The introduction of the reduced ego-graph has no influence on the results. The dimension of the subgoal-cluster was chosen so that we never exclude the optimal trajectory from the prediction. But the most important consequence of this improvement is the reduction of the computational effort which is explained in the next section.

The problem of invisible goal and the proposed solution has less influence on this results. But only because the goals for this dataset were defined accurately and so it was never necessary to invoke this method.

We want to introduce a short note about the updated algorithm. The error decreases rapidly. In a simulation we calculated the mean of the error for all the predictions. We estimated a value of $0.025[m]$. This improvement is obvious because, after each $0.4[s]$ we recalculate the trajectory. Considering that the maximum velocity is $2.4[m/s]$, the maximum possible distance is $0.96[m]$ and lies so in the worst case at the same value as estimated in the first column of table (5.1).

5.2.2 Consideration Simulation Time

The computational effort of the algorithm is variable. The main parameters which influence directly the simulation time are:

- Dimension of the cluster for the selection of the goal oriented trajectories
- Number of pedestrians in the scene
- Sample time of the trajectory

An exact analysis is difficult to formulate. In the following we describe the different influences:

The first influence is based on the dimension of the cluster for the goal orientation. How we mentioned in section (4.2.1) we fix it at 10% of the value of the initial velocity for the respective ego-graph. We compared the results also with a larger value for the cluster dimension, but the improvement was not relevant.
However another reduction of this parameter, to improve the computational effort, is not possible without a increase of the prediction error.
The computational effort is highly depending from the density of pedestrian in the scene, but it not depends directly from the number of pedestrian in a scene The most calculation time is necessary when two trajectories cross.
For example, we predict a trajectory for a pedestrian with other 5 humans and the algorithm requires only 3.7[s]. On the other hand, the prediction for a pedestrian with 3 "neighbors" requires 5.2[s]. With this examples we want to show how the computational effort is heavily dependent from the surrounding environment and the pedestrian in the scene.

The last factor is the sample time of the trajectory. Our value was selected empirically and fixed at 0.05[s]. The experimental results show that this decision was a realistic proportion between the computational effort of the algorithm and a low prediction error.
Obviously an increase of the sample time reduce the computational effort. The two values are direct proportional. In future if we want to predict longer trajectories or have a faster algorithm and need not a so high accuracy then we can augment the sample time.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

We have proposed an approach able to predict a human trajectory. The different cost functions allow to react to changes of the environment and to avoid collisions with other pedestrians. The introduction of the Markov decision process makes it possible to have a probabilistic result. In other words we predict not only the most probable trajectory but a set of trajectories.
The experimental results show that our algorithm allows to predict an accurate trajectory. The prediction estimates the exact 2D-position in the Cartesian space, but provides also the exact prediction in time.
The last factor is improved respect to other methods with the introduction of different PEGs for different initial velocities.

To solve different special cases we implemented features like the detection of invisible goals. This improvement, had no influence on the results in this thesis for the special property of the dataset.
But for a city explorer like the IURO it will be important. We have always given a set of goals and often, they are defined generally. For example if a pedestrian walks on a sidewalk, the goal is indicated somewhere in front. But obstacles like flowerpots and billboards stand around. To avoid them early and safely it is necessary to invoke our algorithm.

6.2 Future work

There are still some open problems. For example the prediction algorithm produces wrong estimation when pedestrians are walking in groups. This case is shown in section 5.1.3..
A future work could be the introduction of a group-detection algorithm and the

consideration of a group as a single object with larger dimensions. This eliminates not only the problem of a wrong estimation, but decreases also the computational effort, because we have to invoke our algorithm for less times.

Another possible improvement is the segmentation of the outdoor environment in different classes.
During the trainings-phase it was difficult to estimate the optimal combination for the cost-function-weights able to produce realistic results for different scenes.
The solution will be the introduction of classes like, sidewalks, crossings, and open spaces. This idea is based on the property of pedestrians to change their behavior with different environments.
Created finally an accurate prediction algorithm we need a dynamic path planner. The difference to standard planner is, that it must be able to consider the pedestrian as a trajectory and not only as a static object. Without this improvement a planner is not able to exploit all the informations provided by our prediction algorithm.

Bibliography

[1] Shu-Yun Chung and Han-Pang Huang *"A Mobile Robot that Understands Pedestrian Spatial Behaviors"*. IEEE/RSJ International Conference on Intelligent Robots and Systems,2010.

[2] S.Pellegrini,A.Ess,K.Schindler,L. van Gool *"You'll Never Walk Alone: Modeling Social Behavior for Multi-target Tracking"*. Proc.IEEE Int.Conf.on Computer Vision, Kyoto, Japan,pp 261-268,2009.

[3] Dirk Helbing and Peter Molnar *"Social force model for pedestrian Dynamics"*, Physical Review E. 51(5):4282-4286,1995.

[4] E.T. Hall *"The Hidden Dimension"*. Man's use of Space in public and Private, Anchor Books, Re-issue,1990.

[5] Thositaka Amaoka, Hamid Laga, Suguru Saito, and Masayuki Nakajima *'Personal Space Modeling for Human-Computer Interaction"*.Proc. 8th Int.Conf. on Entertainment Computing,Paris, France,pp. 60-71,2009.

[6] A.Lacaze,Y.Moscovitz,N.DeClaris,and K.Murphy *"Path Planning for Autonomous Vehicles Driving over Rough Terrain "*. Proc.IEEE Int.Sym. on Intelligent Control,Gaithersburg,MD,USA,pp.50-55,1998.

[7] Andrew Y.Ng, and Stuart Russell *"Algorithms for Inverse Reinforcement Learning"*. Proc.of the 17th Int. Conf. on Machine Learning, Standford,CA,USA,pp.663-670,2000.

[8] José L. Balcázar, Francesco Bonchi, Aristides Gionis, Michèle Sebag *"Machine Learning and Knowledge Discovery in Databases"*European Conference,ECML PKDD 2010, Part II, Springer Verlag Berlin ,2010.

[9] G.Albers,J. Mitchell, L. Guibas, T.Roos *"Voronoi Diagrams of moving points"*. International Journal of Computational Geometry and Applications,1995.

[10] Boris Lau, Christoph Sprunk, and Wolfram Burgard *"Improved Updating of Euclidean Distance Maps and Voronoi Diagrams"*. Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Taipei, Taiwan 2010.

[11] Sridhar Mahadevan *"Learning Representation and Control in Markov Decision Processes: New Frontiers"*. Foundation and Trends in Machine Learning, vol 1, no 4, pp 403-565, 2008

[12] J.C. Baxter *"Interpersonal Spacing in Natural Settings"*. Sociometry 33(4), pp 444-456, 1970

[13] D.B. Henson *" Visual Fields "* Oxford: Oxford University Press, 1993.

[14] W. Angenendt, and M.Wilken *" Gehwege mit Benutzungsmoeglichkeiten fuer Radfahrer "* In Schriftenreihe Forschung Strassenbau und Strassenverkehrstechnik, Heft Nr. 737, Bonn 1997

[15] Osama Masoud and Nikolaos Papanikolopoulos *" A Novel Method for Tracking and Counting Pedestrians in Real-Time Using a Single Camera "* IEEE Transactions on vehicular technology, VOL.50, NO.5, September 2001

[16] Scott Gaffney and Padhraic Smyth *" Trajectory Clustering with Mixtures of Regression Models "* Technical Report No. 99-15 Department of Information and Computer Science, University of California,Irvine. Mrch 1999

[17] Steven M. La Valle *" Planning Algorithms "* Cambridge University Press,2006

[18] Jesus A.De Loera, Jorg Rambau and Francisco Santos *" Triangulations: Sturctures for Algorithms and Applications "* Springer-Verlag Berlin Heidelberg, 2010

[19] Peter Su and Robert L.Scot Drysdale *"A Comparison of Sequential Delaunay Triangulation Algorithms "* Springer-Verlag Berlin Heidelberg, 2010

[20] L.Guibas,D.Knuth, and M.Sharir *"Randomized incremental construction of Delaunay and Voronoi diagrams "* Algorithmica 7: 381-413,1992

[21] M. de BErg, O.Cheong, M.van Kreveld and M.Overmars *"Computational Geometry: Algorithms and Applications "* Springer-Verlag Berlin Heidelberg, 2008

[22] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein *" Introduction to Algorithms "* Second Edition. MIT Press and McGraw-Hill, 2001

[23] Shannon, C.E. *"A Mathematical Theory of Communication"*, Bell System Technical Journal, vol. 27, pp. 379–423, 623–656, July, October, 1948

[24] N.R. Draper and H. Smith *"Applied Regression Analysis"*, New York: John Wiley and Sons, 2nd Edition, 1981

[25] G.J. McLachlan and T.Krishnan *"The EM Algorithm and Extensions"*, New York: John Wiley and Sons, 1997

[26] Gergely Csibra, Gyorgy Gergely, Szilvia Biro, Orsolya Koos and Margaret Brockbank *"Goal attribution without agency cues: the perception of "pure reason" in infancy"* Elsevier, Cognition 72(1999) 237-267, 1999

[27] Maren Bennewitz *"Mobile Robot Navigation in Dynamic Environments "*.Phd-thesis, University of Freiburg,2004.

[28] Hannah Dee, and David Hogg *"Detecting inexplicable behavior "*. In British Machine Vision Conference, volume 477,pp.486,2004.

[29] Simon Thompson, Takehiro Horiuchi, and Satoshi Kagami *"A probabilistic Model of Human Motion and Navigation Intent for Mobile Robot Path Planning "*. In autonomous Robots and Agents,2009.

[30] Jur van Berg, Stephen J. Guy,Ming Lin, and Dinesh Manocha *"Reciprocal n-body Collision Avoidance "*.International Symposium on Robotics Research (ISRR), Sep. 2009.

[31] Reginald G.Golledge *"Defining the criteria used in path selection "*. Technical Report, University of California,Transportation Center,1995.

[32] H.C.Yen,H.P.Huang, and S.Y.Chung *"Goal-directed pedestrian model for long-term motion prediction with the application to robot motion planning "*.IEEE International Conference on, pages 216-219, August 1990.

[33] K Lewin *"Field Theory in Social science "*.New York: Harper & Row,1951.

[34] F.Hoeller,D.Schulz,M.Moors, and F.E Schneider *"Accompanying persons with a mobile robot using motion prediction and probabilistic road maps "*.In intelligent Robots and Systems, 2007.

Appendix A

Implementation

A.1 Dijkstra Algorithm

The pseudo code is presented in the following figure:

```
 1  function Dijkstra(Graph, source):
 2      for each vertex v in Graph:           // Initializations
 3          dist[v] := infinity ;             // Unknown distance function
     // from source to v
 4          previous[v] := undefined ;        // Previous node in optimal path
     // from source
 5      end for ;
 6      dist[source] := 0 ;              // Distance from source to source
 7      Q := the set of all nodes in Graph ;
        // All nodes in the graph are unoptimized - thus are in Q
 8      while Q is not empty:                  // The main loop
 9          u := vertex in Q with smallest dist[] ;
10          if dist[u] = infinity: \\
11              break ;                         // all remaining vertices
       // are inaccessible from source
12          fi ;\\
13          remove u from Q ;\\
14          for each neighbor v of u:          // where v has not yet been
      // removed from Q.
15              alt := dist[u] + dist-between(u, v) ;
16              if alt < dist[v]:              // Relax (u,v,a)
17                  dist[v] := alt ;
18                  previous[v] := u ;
19              fi  ;
20          end for ;
```

```
21      end while ;
22      return dist[] ;
23  end Dijkstra.
```

We use the algorithm to find only the shortest path from the pedestrians actual position to a specific target. So the algorithm can be interrupted at line 13 with the condition $if(target = u)$.

A.2 Dataset

A.2.1 Description

In this thesis we consider two datasets. Both are provided by the ETH Zurich. One represent the scene in front of the entrance of the main building of the university and the other a city scene filmed from the roof of a hotel in the center of Zurich.
We consider first the University-video: A typical city situation is shown (Figure (4.11)). Humans walk straight on a sidewalk or turn to the university entrance, other pedestrian leaves the building.

In the other video (Figure(4.12)), is shown a tram-station and the entrance of a

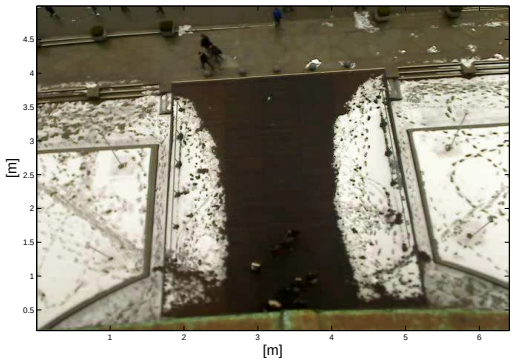


Figure A.1: Environment in front of the ETH Zürich

hotel and therefore really crowded. Another difference to the previous video are four static obstacles in the middle of the scene (trees and a banquette).

In the following section we describe the details how we elaborate this videos and

Figure A.2: Environment in front of a hotel

what information we can get from them. All this notations are valid for both.

A.2.2 Analysis and Preparation of the Dataset

The most significant information for a pedestrian trajectory are:

- Position in the euclidean space
- Time associated to every position
- Target of the trajectory
- Obstacles and other pedestrians in the same scene

All this data are presented as a numerical description for the videos. Our objective is to combine all the relevant informations.
The (x,y)-positions and the velocity (v_x, v_y) are represented in world coordinates. In general all the calculations are done in this coordinate system ,with exception of the obstacle map which is explained later on.
The problem is that the positions are ordered in sequence of time. It means all the pedestrian in the same frame are presented one after the other. But with the identify-number of every pedestrian it is simple to summarize all the positions of one trajectory.
The next step is the extraction of the time associated to every position. It is given indirectly through the frame number. We know, the frame-rate of the video ($2.5 fps$) and the difference of the frame number between two following positions. So it is simple to estimate the time-difference.

The obstacle map is given as a png-image of the scene and to handle with it easier during the different operations we convert it in a matrix. White (obstacle) is 255

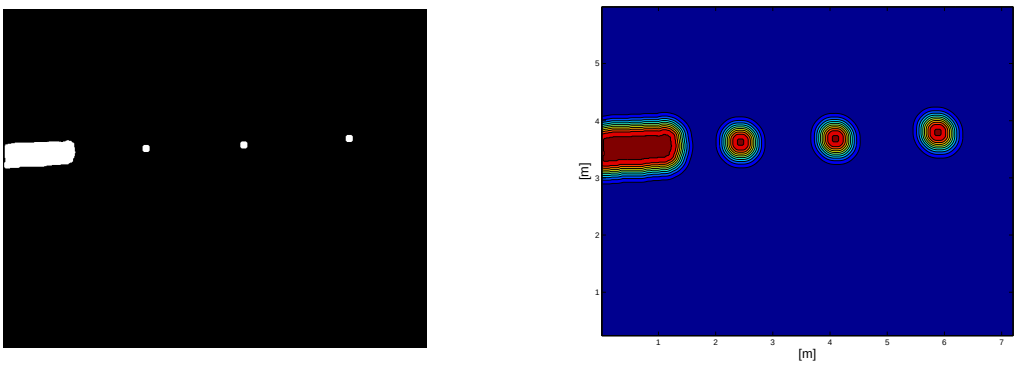


Figure A.3: Obstacle map extracted from a video and the respective energy map (hotel video)

and black (free space) 0 indicates if in a cell is an obstacle or not. The positions of the obstacles are introduced in the image-coordinates.

To combine this environment factor with the other informations we must convert the map with the homography matrix H in the world coordinates.

We need another information of the obstacle maps. The corner of each obstacles are necessary to create the configuration space and to determine the Delaunay triangulations. This step solved with a corner-detection-algorithm.
The last information is the target of every pedestrian. For the University-video we have four and for the other sixteen targets for all the trajectories. The association of the exact goal to every trajectory is realized with a geometrical algorithm. We consider the direction of the pedestrian, described by the angle between the initial and end position of the trajectory, and the direction to all the different goals. The target with the most similar direction of the pedestrian is chosen as the exact one. Finally applying all this different algorithms we estimated 360 different pedestrian trajectories for the university-video and 390 for the hotel-video.

A last consideration is done about the length of the trajectories. For the learn algorithm it is not helpful to choose too short trajectories ($< 1m$) or completely static pedestrian. How explained before this pedestrian are modeled as a static obstacle. Also we must eliminate pedestrian which walks in groups, because in this case the standard considerations about the personal space and other human behaviors are not satisfied. This assumption is a common rule applied also in all the previous mentioned papers [1],[2] [28].

Appendix B

Simulation

In the first part we predict the most probable trajectories only for one pedestrian in the scene. After we simulated a scene with two pedestrians and include also the interactions between them. The obstacles, the initial velocity and initial position are determined artificially.

B.1 Single pedestrian

In the first simulations we check if all the different cost functions are exactly implemented. We simulate a scene with multiple pedestrians, but they are all considered as static obstacles. It means that we do not use the formula (3.8) to describe the human behavior but only (3.7). So the person are considered as a static obstacle. In the first paragraph we check the different cost functions. Then we examine the implementation for only one pedestrian and in the last section for multiple pedestrians.
It is important to note, that in this cases we used the first realization for the ego-graph. The field of view was reduced to 120 degrees and the accelerations in the y-direction was only positive.

B.1.1 Single cost function

We calculate separate the single cost function function. For this we use the following cost weight combinations:

$w_{str} = 10$ all the other weights are reseted to zero.

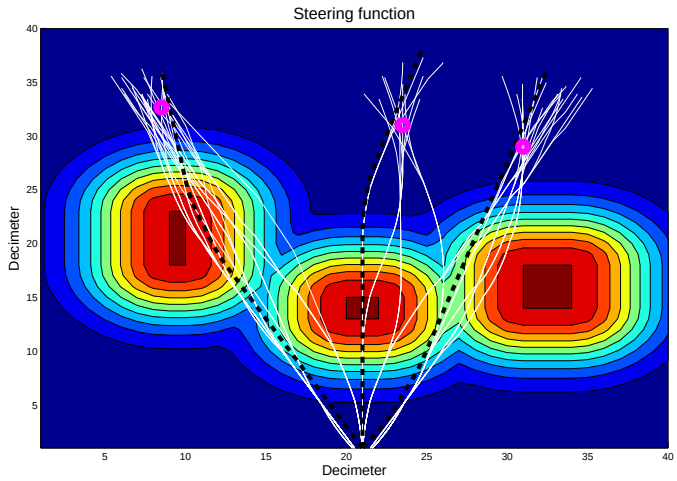


Figure B.1: Minimum-steering-path

$w_{dist} = 10$ all the other weights are reseted to zero.

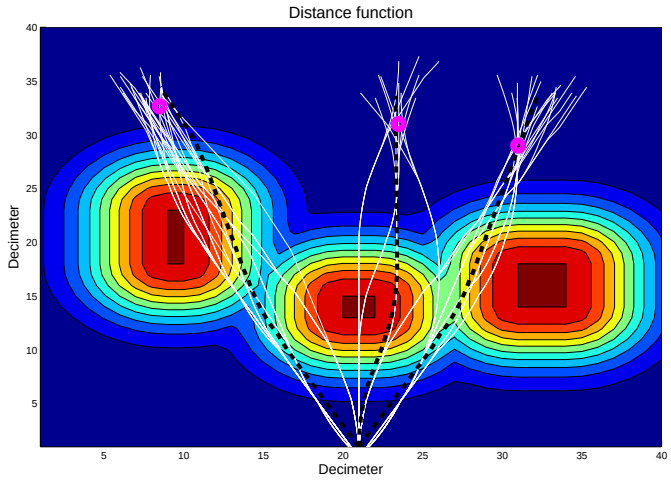


Figure B.2: Minimum-distance-path

$w_{obs} = 10$ all the other weights are reseted to zero.

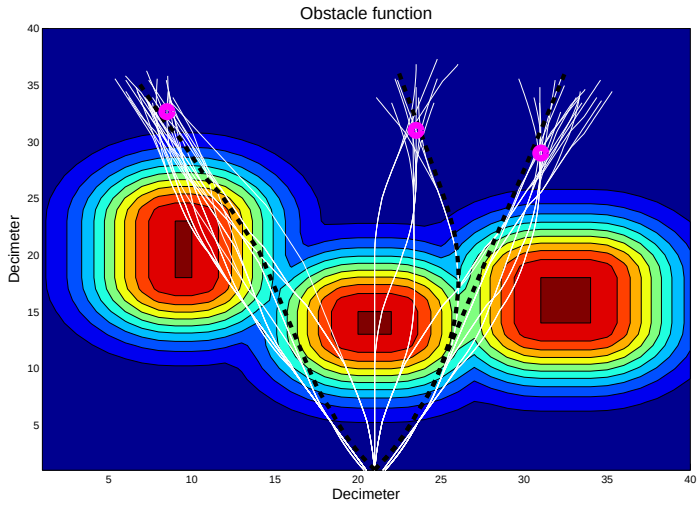


Figure B.3: Minimum-cost-path considering only the static obstacles

B.1.2 Combined cost function

In this section we show the different steps of the implementation:
Generation of the ego-graph:

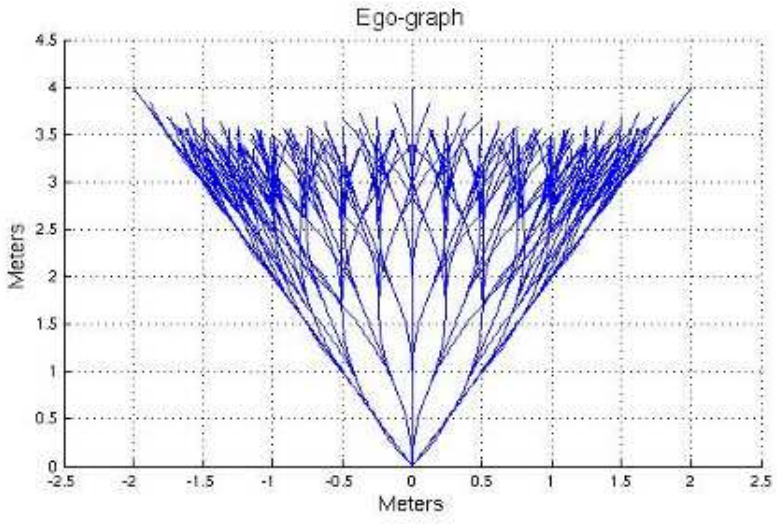


Figure B.4: Ego-graph

Clustering of the ego-graph:

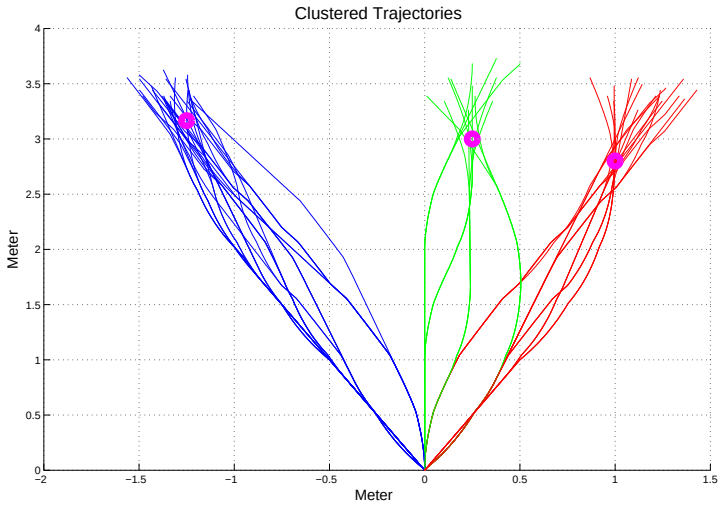


Figure B.5: Clustered trajectories

Combination of cost functions and Ego-graph:

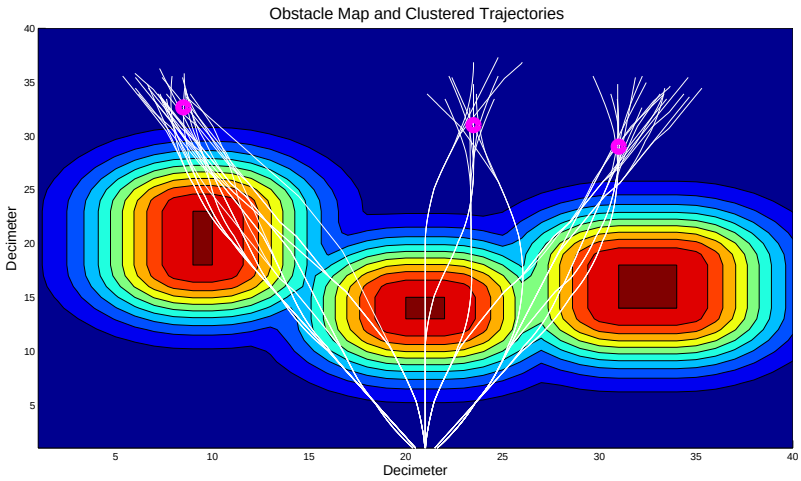


Figure B.6: Clustered trajectories over the obstacle map

Determination of the minimum cost trajectory:

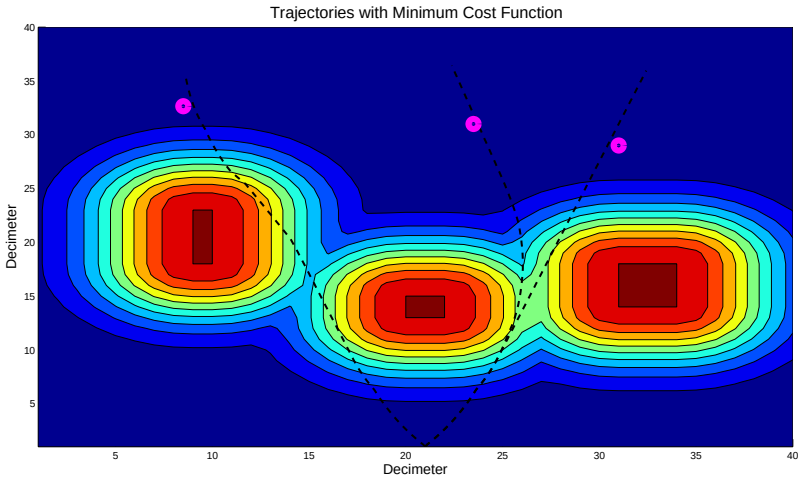


Figure B.7: Minimum-cost-path

B.2 Multiple pedestrians

In this scene two pedestrians are considered. We generate two ego-graphs and implement the cost function (3.8) to model the human interaction between pedestrians. Ego-graph generated for two pedestrians:

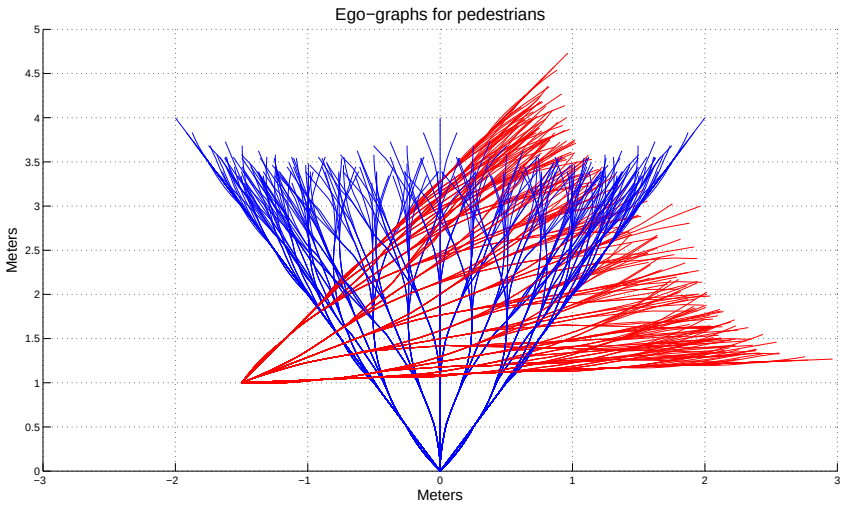


Figure B.8: Scene with two pedestrians

Clustered trajectories for different trajectories:

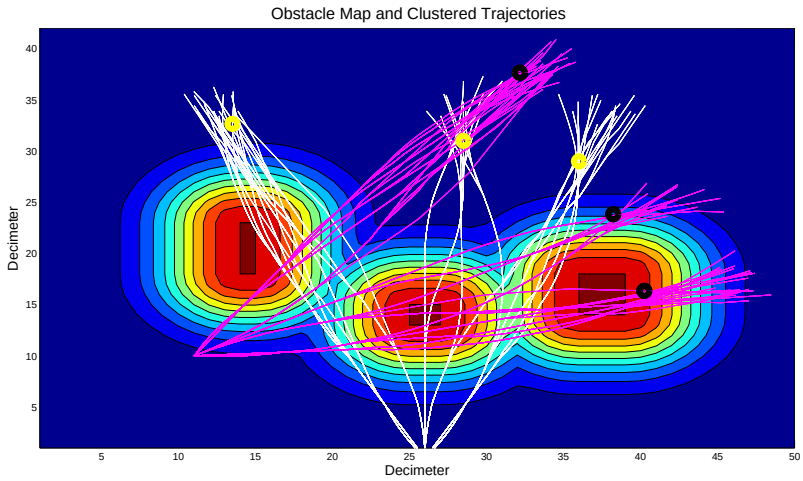


Figure B.9: Clustered trajectories for the different goals

Best trajectories for the two pedestrians:

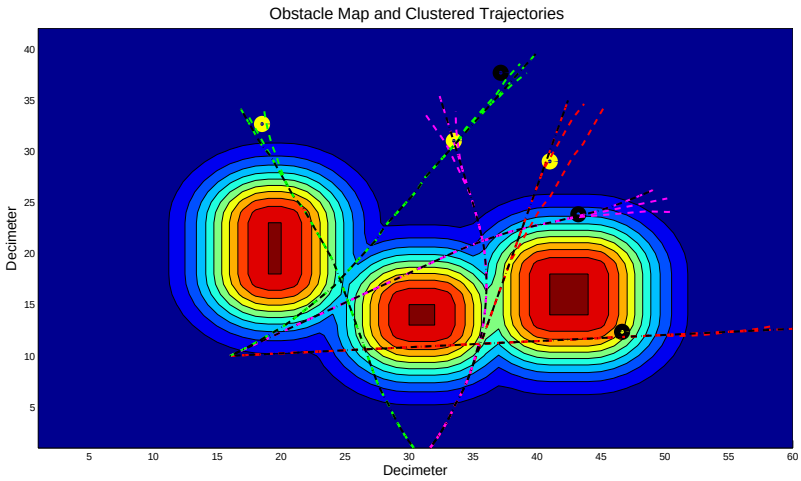


Figure B.10: Minimum-cost-path