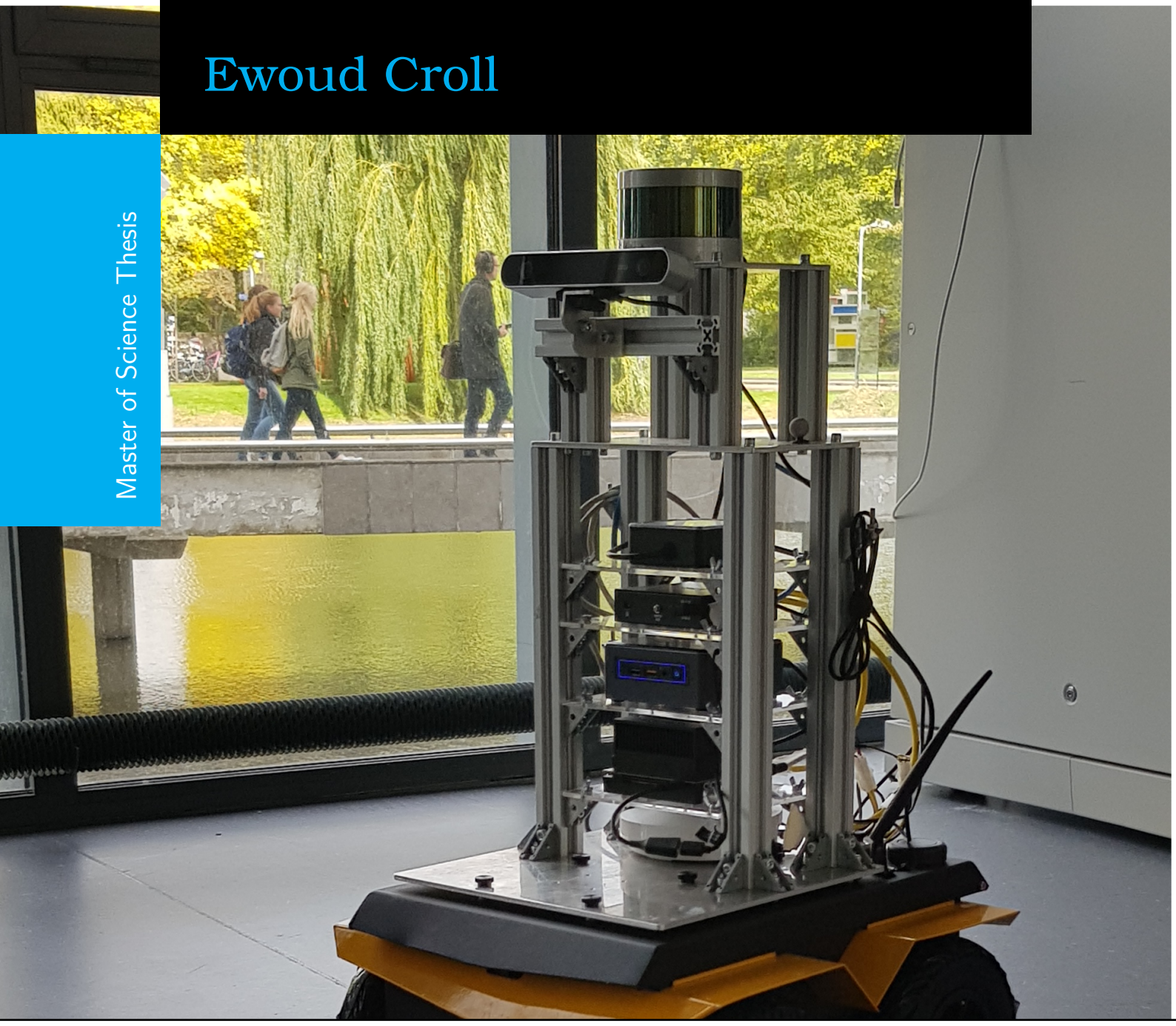
Safe and natural navigation in dynamic environments

learned from human behavior

Ewoud Croll

Master of Science Thesis



TU Delft Delft University of Technology

Department of Cognitive Robotics

Safe and natural navigation in dynamic environments

learned from human behavior

Master of Science Thesis

Ewoud Croll

March 31, 2020

Department of Cognitive Robotics (CoR)
Faculty of Mechanical, Maritime and Materials Engineering (3mE)
Delft University of Technology

This work is done under supervision of Dr. Javier Alonso-Mora and PhD Student Bruno de Brito.

Abstract

Social Navigation is the task of robot motion planning in an environment shared with humans. This is an especially hard sub-problem of motion planning because the planner has to deal with a dynamic, continuous and unpredictable environment. We present a local motion planner, namely Neural Network Model Predictive Control, for autonomous ground vehicles in highly dynamic environments. A neural network is trained to plan local trajectories based on human behavior data. It has therefor learned to mimic how a person would behave in such a situation. The trajectory plan of the neural network is used as guidance and initialization of a model predictive controller. This MPC creates a kinematically feasible trajectory and assures collision avoidance with the static and dynamic obstacles in the environment within its receding horizon. This combined planner and controller is tested in simulation and showed on a real autonomous robot.

Table of Contents

Preface

In front of you lies my master thesis work, a years worth of diving deep into the topic of motion planning techniques. I got the chance to try and wrap my brain around the paradoxes of interactions between robots and humans and the models that try to describe them. Most interesting is the realization that much of the behavior we expect from the robot results in a discussion on how humans make their decision and how they learned to come to that decision. Humans, just like neural networks, need a lot of experience before they understand what to do in a certain situation. Similar to a model predictive controller we can only look so far ahead in life until the variables become so uncertain that you just have to trust it will turn out all right. We have no other choice than to resort to heuristics that are handed down from person to person and although there is no proof those are still the right way to do things, it is a good starting point of the optimization that life is. Its interesting to see that now, during the COVID-19 crisis, human have to reinvent their own motion planning and solve the planning puzzle much more conscious in order to keep the required 1.5 meter distance. It gives a little taste of the actual complexity of the task a robot has to deal with.

There are several people that I want to thank especially.

From the research group I want to thank Bruno, my daily supervisor, who provided me with a great starting point to build further on. He was always available to answer questions and able to come up with possible solutions for the problems I encountered. I admire is working drive and wish him all the best in finding the perfect motion planner for his PhD research. I also want to thank my professor, Javier Alonso-Mora, for his critical eye and vast vast knowledge of the research area. During the meetings he was able to quickly identify shortcomings in any solution I presented which gave direction to my research. This, together with the casual conversations we had in the hallways and during lunch, made a really nice working experience.

I want to thank my parents, who supported me through my entire studies. They gave me advice were necessary, but just as importantly, backed me on every decisions I took. They are a stable factor, which i can always turn to. I am thankful for their financial investment, and for the great launch they gave me into this world.

My friends in the studies Achin, Yannick, Anoosh and Chadi, for the interesting discussion on neural networks, motion planners and life. I also want to thank them for the fun times in the lab and during the breaks, motivating me to come to university day after day.

My friends from back in Enschede, especially Robin, Paul, Kruiter and Schijvens from Vulpes; Bart, Tom, Robin and Rogier from my house het Riet, the guys from my dispute Heres, the people from my first house Castillo del Conjo and all the other great people I met along the way. I also want to thank all the friends I made in Delft, including everybody from Plankenkoorts, Carli and her gang and my current roommates who gave me the welcome distractions and made studying a time to never forget.

Figure 1: Jackal in the rain.

Chapter 1

Introduction

Autonomous navigation for robots is a research area in the spotlights nowadays. Autonomous cars promise increased mobility, safety and efficiency. Robots can support people who work in dangerous environments and can do jobs that people don't want to do such as very repetitive work or work that is though on the human body. Highly automating a robot has the advantage that it needs less supervision or can even perform it's work completely on its own.

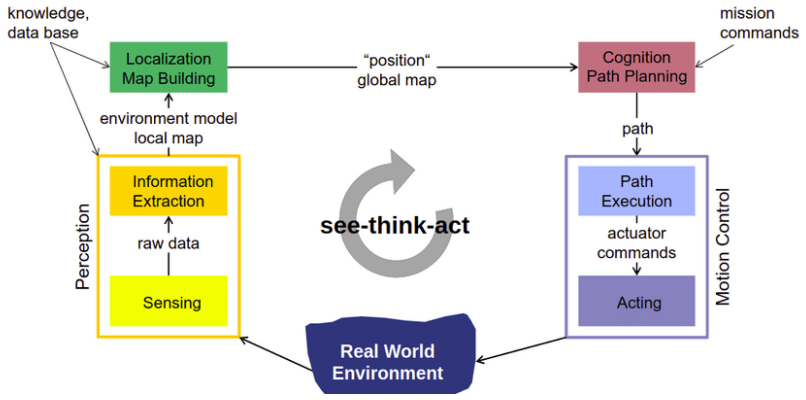


Figure 1-1: Typical automated robot cycle

Typically robot automation is divided into three separate components that run sequentially to make the robot move. This is called the "See, think, act" cycle and consists of a perception component, a logic component and a controller. The perception component is responsible for handling the raw sensor data and tasks such as filtering. The logic component has three main responsibilities: map building, localization and to coming up with a path for the robot on that map. The controller is responsible for calculating motor commands to follow the path as close as possible. There are multiple way to divide the robot's intelligence over the different components. For example the perception component could already disregard obstacle's that are definatly not going to obstruct, such as obstacles that are hanging from the ceiling for a ground robot. It could be designed simpler and just pass the information about the object

on to the logic component and let it make the decision by itself. A lot of robots uses this structuring of tasks, but research is done on combining several of these components. Path planning and controlling can be combined in a model predictive controller for example. The whole pipeline can be replaced by an end to end neural network which generates a command directly from the raw sensor inputs.

1-1 Human interaction

Robots are finding their way from factories where they are isolated from human interactions to everyday situations such as offices, hospitals and households. These are environments where robots and humans share the same space. Robots are expected to not get in the way of humans but still perform their task efficiently. One of the problems that arise is the motion planning of the robot where the robot has to understand the applicable social norms and has direct interactions with humans. This problem set is called social navigation [1]. Which lies on the cross section between the fields of robot motion planning and human-robot interaction (HRI). The topic of social navigation is one that has existed for quite some time but has gotten more attention in recent years. A reason for this development is the availability of adequate hardware platforms. The developments in perception of the environment make it possible to detect static and dynamic obstacles. The next step is improving the decision making and the motion planning capabilities in dynamic and hard to predict environments.

Figure 1-2: Two pedestrians blocking the hallway, the robot needs to predict the interaction to find its path to the end of the hallway. [2]

Autonomous robots currently in operation typically follow a pre-planned path or a path derived from markings on the road. It has limited capabilities in going around any obstructions in its path, most robots will stop and only proceed when their planned path is free of obstacles again. In research, methods have been applied for planning a route through an unseen static environment. They are shown to work in real time and give reasonable trajectories, however it becomes a lot more difficult to plan in an environment with other pedestrians walking around. While walking around one has to predict where the other pedestrians are going and what is the right way to proceed in the environment. The right way is determined by many social rules, which are not captured by exact science as of today. Examples of these social rules are keeping to the right side of a hallway, not crossing through groups of people and keeping a certain distance to others depending on how crowded the area is. A robot that can understand

these rules will be able to navigate through crowded environments with much more ease and have a higher acceptance than a robot that is only able to follow a fixed path. In the situation of figure 1-2 gives an example for this. Two people are walking side by side, leaving too little room for the autonomous wheelchair to pass. A planner which assumes a static or constant velocity world model would not find a feasible path because of the pedestrians blocking the way. A smarter planer would predict that one of the pedestrians will probably move out of the way when to robot gets close, this interaction is vital for the robot to reach its goal. The general problem of robots stopping because they are unable to predict the interactions with humans in the environment is referred to as the freezing robot problem [3].

1-2 Research Aim

This research aims to further discover the possibilities of teaching a robot to navigate through a crowded environment. We are looking to create a method that works in a real environment and therefor demonstrate the workings on an autonomous robot platform. We aim to build a method that learns its planning rules from humans, this is a complex model where we expect deep learning to be a viable solution for replicating human behavior. At the same time we want the robot to always operate in a safe way, a receding horizon controller can come up with trajectories and check them for collisions over a limited horizon. In this research we will find the answer to the question: Can we use deep learning based on human behavior to improve a constraint trajectory optimization planner?

1-3 Outline

The core part of this report is covered in chapter 2 which consist of a scientific research paper in which the method, experiments, results and conclusions are discussed. The method consists of two parts: the creation of a neural network for planning trajectories and the integration of this planner into a model predictive controller. Multiple simulations are done in a simulation and a proof of concept is showed on a real robot. After the scientific paper in chapter 3 we go into a more in depth discussion on several advantages and challenges of the presented approach together with recommendations for future research. In the appendix we further discuss the following parts of the thesis:

- Appendix A: More results from the simulation experiments.
- Appendix B: Discusses variations of the neural network we tried, together with tests and results.
- Appendix C: Discusses variations for integration of the neural network and the model predictive controller, together with simulations experiments and results.
- Appendix D: Shows two alternatives for the used static free space search.
- Appendix E: Explains the approach used in the data gathering experiment and shows results of using this real data for the neural network.

Chapter 2

Scientific Paper

NN-MPC: Safe and natural navigation in dynamic environments learned from human behavior

Ewoud Croll

***Abstract*— Social Navigation is the task of robot motion planning in an environment shared with humans. This is an especially hard sub-problem of motion planning because the planner has to deal with a dynamic, continuous and unpredictable environment. We present a local motion planner, namely Neural Network Model Predictive Control, for autonomous ground vehicles in highly dynamic environments. A neural network is trained to plan local trajectories based on human behavior data. It has therefor learned to mimic how a person would behave in such a situation. The trajectory plan of the neural network is used as guidance and initialization of a model predictive controller. This MPC creates a kinematically feasible trajectory and assures collision avoidance with the static and dynamic obstacles in the environment within its receding horizon. This combined planner and controller is tested in simulation and showed on a real autonomous robot.**

I. Introduction

This paper proposes a new method for safe navigation of Autonomous Ground Vehicles in dynamic environments. Commonly dynamic environments are treated like static environments in which the robot has to re-plan its trajectory often to account for the changes in the environment. In this way valuable knowledge about movement in the area is lost. Often the navigation is split into two separate problems, the motion planner and the controller [1]. Motion planner techniques however usually don't take the trajectory smoothness and kinodynamic feasibility into account. This can cause the controller to be unable to follow the trajectory. The controller can then choose to follow the planned path as closely as possible but then the guarantees about obstacle avoidance given by the planner are dismissed and a collision with a static or dynamic obstacle is very possible. Instead we would like to have method that does both the local planning and control simultaneously.

In highly dynamic and uncertain environments static motion planners are not adequate for modeling the movement in the surroundings of the robot. local motion planning becomes a necessity for collision avoidance. For this nowadays usually reactive methods are used to guide the AGV away from a collision. Artificial potential field methods [2] Social force methods [3] and Dynamic window methods [4] are examples of this, however they lack convergence guarantees and are very limited in their decision making and reasoning skills. They cannot make a move that is unfavorable at the current time but results in a overall favor in the long run. This can lead to the robot getting stuck at a local minimum of its artificial potential field and not being able to move further. On top of that once an environment surpasses a certain dynamic complexity the robot decides all paths have a risk of collision and the robot decides to stop driving, this is called the freezing robot problem [5]. A solution is to better model the interactions between the robot and surrounding humans, this gives knowledge about which trajectory plans are likely to be safe.

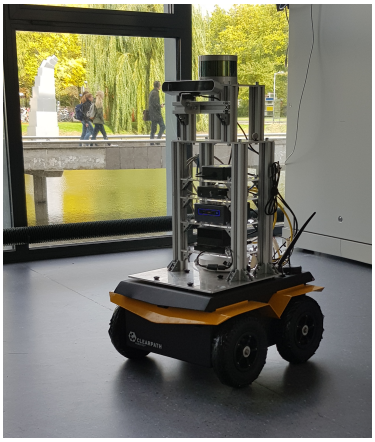


Fig. 1: The mobile robot platform

A. Related work

More recently researchers have been interested in using data from human behavior to guide the robot. For example by using inverse reinforcement learning [6][7] to model human behavior in a predefined set of features. This model is then used on-line to guide the robot. In these models however allow no goal position for the robot to be specified and therefor it is only able to drive as straight as possible while exerting human like behavior. Driving human like is not a goal in it self, so these robots are still a step away from being useful as autonomous agents.

Another way to learn from examples is to create an end-to-end neural network to guide the robot, trained using reinforcement learning [8]. In this work the robot is rewarded for reaching its goal and penalized for collisions with pedestrians and for disregarding predefined social norms. The neural network does not give any guarantees for not colliding with obstacles, therefor an extra layer to stop the robot in unsafe situations is necessary. When this layer is activated the robot will stop and it is not able to find a safe path until the situation changes. We would like to have more flexibility and be able to still drive the robot safely even though the trajectory planned by the neural network violates the collision constraints.

A solution proposed by B Brito [9][1] from our research group is to use Local Model Predictive Contouring Control (LMPCC) which integrates a local planning stage with the controller. This method allows to compute safe, smooth and kinodynamic feasible trajectories, while also computing the optimal control inputs over a planning horizon. The planner is encouraged to follow a predefined spline while being restricted to not collide with the static and dynamic obstacles as explained in section II.

There are however some limitations in this controller that can cause it to show undesirable behavior. It is difficult to define the cost function and its weights such that the behavior of robot is as expected in different scenario's. This is undesirable because we strive to create a robot that is flexible in its use and can be introduced into a new environment without having to change the system. Another issue is that the gradient based nature of the optimizer will search in the neighbourhood of the previous solution and will not come to a solution where the reward is lower at first. This effect is amplified by the very limited number of optimization steps in each cycle. In this work the free space is calculated in the neighborhood of the previous calculated trajectory which again limits the planner to the area where it has planned to go to before. The supplied reference trajectory to the LMPCC also has a big influence on the robot behavior and can be limiting if the robot has to deviate far from this reference due to a busy dynamic situation. Instead we would like to have a more flexible setup that can quickly change its planned path when drastic changes in the environment occur. The above problems we are looking to solve by introducing a neural network planner to support the model predictive controller in multiple ways.

B. Contributions

We propose a solution of the motion planning problem by combining the rigorousness and reliability of Local Model Predictive Controller with the flexibility and scene understanding of a Deep Neural Net trained on human behavior data. An overview of the different components of this planner can be seen in figure 2. Building on LMPCC [9] we made the following changes and contributions.

1) Create a planner neural network to guide the robot in a natural way instead of using a predefined reference trajectory.
2) Define a MPC cost function and initialization that uses the trajectory generated by the neural network, while still maintaining obstacle avoidance capabilities for both the dynamic and static obstacles.
3) Performance results and demonstration in simulation and real world situations using a mobile robot.

II. PRELIMINARIES

A. Robot description

The robot B is assumed to operate in a 2 dimensional plane $\mathcal{W} = \mathbb{R}^2$ and its dynamics can be described by

[1]Available at: www.alonsomora.com/docs/19-brito-ral.pdf

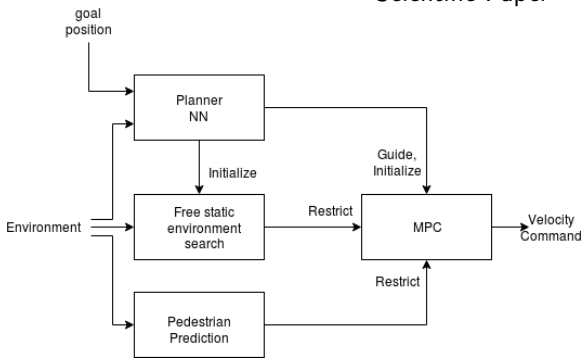


Fig. 2: Overview of the combined neural net and MPC planner

discrete-time nonlinear system where the future state depends on the current state and the inputs to the robot.

$$\boldsymbol{z}(t+1) = f(\boldsymbol{z}(t), \boldsymbol{u}(t)), \tag{1}$$

Where $\boldsymbol{z}(t)$ is the state of the robot which consists of its position, orientation and velocities. $\boldsymbol{u}(t)$ is the input to the robot defined by a linear and angular velocity. The area occupied by the robot is approximated by a circle encompassing in the whole of the robot. We will use x, y, and θ to describe the position and heading of the robot in the global frame and v and ω for its linear and angular velocity. the subscribed nn is used to describe the trajectory generated by the neural network, the subscript gt is used for the ground truth positions from the dataset and the subscript mpc to describe the trajectory generated by the model predictive controller. The design weight weights Q are used for the MPC cost model J.

B. Obstacles

Static obstacles in the environment denoted by $\mathcal{O}^{\text{static}} \subset \mathcal{W}$ are encoded in an occupancy grid map This map is created before the experiment either manually or using a SLAM approach. During experiment the map is used in both path planning, during the real experiments the map is also used for localisation. During an experiment the current readings of the sensors are compared to the static map and all obstacles that are not found on the map are considered to be dynamic obstacles. All the dynamic obstacles i are represented by ellipses moving obstacle of area $\mathcal{A}_i \subset \mathcal{W}$ with a major and minor axis. The area occupied by all moving obstacles is defined by $\mathcal{O}_t^{\text{dyn}} = \bigcup_{i\in\{1,\dots,n\}} \mathcal{A}_i$. Their future position can be predicted using a constant velocity model or using a neural network. To remove some uncertainties in the measurement of the positions of the moving obstacles we use a linear Kalman filter.

C. MPC formulation

We are tasked with generating a trajectory for the robot N time steps into the future which is collision free to both static and dynamic obstacles. For this we formulate the problem as minimizing a cost function J that includes deviations from

the neural network generated path γ defined by waypoints over the planning horizon.

$$J^* = \min_{\boldsymbol{z}_{0:N}, \boldsymbol{u}_{0:N}, \gamma_{0:N}} \sum_{k=0}^{N} J(\boldsymbol{z}_k, \boldsymbol{u}_k, \gamma_k)$$

$$\text{s.t.} \quad \boldsymbol{z}_{k+1} = f(\boldsymbol{z}_k, \boldsymbol{u}_k), \tag{2a}$$

$$\mathcal{B}(\boldsymbol{z}_k) \cap (\mathcal{O}^{\text{static}} \cup \mathcal{O}_k^{\text{dyn}}) = \emptyset, \tag{2b}$$

$$\boldsymbol{u}_k \in \mathcal{U},\ \boldsymbol{z}_k \in \mathcal{Z},\ \boldsymbol{z}_0 \text{ given}. \tag{2c}$$

The MPC optimizes its trajectory over a horizon and the first velocity of this plan is executed on the robot. The NNMPC problem of Eq. 2 is nonconvex. To solve this optimization problem online in real time we use ACADO with its and its C-code generation tool. For the robot dynamics we use a unicycle model with linear and angular velocity. The model is then discretized directly in ACADO using a multiple-shooting method combined with a Gauss-Legendre integrator of order 4, no Hessian approximations, and a sampling time of 50 ms. qpOASES is used to solve the resulting QP problem. At a maximum of 10 iterations we require a KKT tolerance of 10^{-4}. When the solver is not able to find a feasible solution the robot is stopped.

D. LMPCC

We compare the NN-MPC to the existing method in the group: the LMPCC. It uses a trajectory spline generated from manually defined way points. In the cost function of the model predictive controller there are terms to encourage to following the generated trajectory by with a contour and lag error term. The optimizer is restricted to find its solutions that do not collide with static and dynamic obstacles as described in the previous sections. The optimizer is initialized with the result of the previous iteration.

III. Method: Planning network

A. Network structure

The planning network we use is based on the pedestrian prediction network developed in the group. This network already has capabilities of capturing human behavior around obstacles and other humans. It is altered to make it is suitable for planning a trajectory to a goal position instead of only predicting a trajectory based on historical data. The inputs of the neural network are an occupancy grid, a radial pedestrian grid, current state of the robot and distance to the goal.

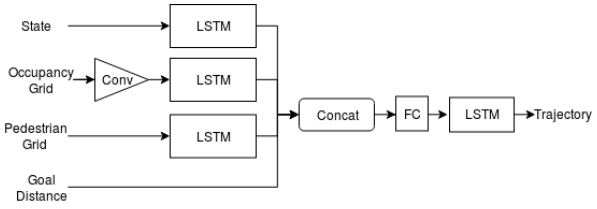


Fig. 3: Structure of the planning network

The occupancy grid is a 60x60 grid section extracted from a global occupancy grid, only showing the surroundings in the neighbourhood of the robot. The size of occupancy grid would give too many parameters for the network to learn, therefor it is passed through convolutional layers that are trained using an autoencoder [10]. The radial pedestrian grid input consists of a vector of 72 elements, each representing a 5 degree angle around the robot. The value of the element is the distance between the robot and the closest pedestrian in that 5 degree section, with a max of 6 meters. The state input is defined as the heading in radians and the absolute velocity of the robot with respect to the robot goal frame. The input for the distance to goal is added to make sure the robot stops when it reaches its goal. The direction of the goal will be discussed later.

The network outputs the trajectory as a single vector consisting of x and y velocities for the planned horizon. We chose 15 steps into the future, which results in an output vector of size 30. The time between the steps is 0.15 seconds. This gives a plan for 2.25 seconds into the future, which is enough to avoid imminent collisions coming up without over burdening the MPC optimization. At a velocity of 1.0 m/s, each step is about 0.15 meters apart. This is considerably less than the size of the robot, so we can guarantee no collisions on the path in the MPC restriction.

As can be seen in figure 3 all the inputs of the neural network, except the goal distance, go through separate LSTM (long short-term memory) layers and after concatenation together with the goal distance go through a fully connected layer, another LSTM layer and another fully connected layer. The LSTM layers have memory built in the cell, which gives it capabilities of handling sequences of data. This is useful because we want the network to take historic data into account when planning its path. With that it can learn more complex interaction between pedestrian depending of the direction and speed they are walking at. Additionally these LSTM cell give some inertia to the network causing it to retain its chosen path to a certain extend. This is useful because we want the robot to keep the same trajectory at least for a little bit. Because of the dependence on historical data the network is always trained and tested using sequences of data.

The loss function looks as follows.

$$\mathscr{L}_R = \mathbb{E}_{\mathbf{v}\sim\mathbf{D}} \left[\frac{1}{T} \sum_{k}^{T} |\hat{\mathbf{v}}_k - \mathbf{v}_k| \right] \tag{3}$$

where $\hat{\mathbf{v}}_k$ is the predicted velocity vector at step k from a sequence with length T sample from the dataset, $\mathbf{D}$ and, $\mathbf{v}$ is the ground-truth. Since we train using multiple sequential positions we define the loss function of an entire sequence as the mean of the loss function of each predicted trajectory in the sequence. The final loss function is found in equation 3. For training we use the RMSProp optimizer and the learning rate decays exponentially.

B. Auto encoder

For the occupancy grid we use several convolutional layers. The purpose of these layers is to lower the dimensionality of the input. Starting with a grid of 60x60 = 3600 variables, we apply three convolutional layers. The first layer has size 5, stride 2 and 64 kernels. The second layer has size 3, stride 2 and 32 kernels. The third layers has size 3, stride 2 and 8 kernels. This leaves us with 8 images of size 8x8. These images are flattened to form a vector of length 512. This is much smaller than the original input of 3600 variables. To train the weights of the encoder we create an auto encoder by attaching a decoder consisting of the same convolutional layers in reverse. An auto encoder works by having the output resemble the input of the auto encoder as closely as possible. The bottle neck is the smallest layer, the auto encoder needs to learn a most meaningful and dense representation of the occupancy grid to reproduce it with the decoder. An example of an input grid and the produced output of the auto encoder can be seen in figure 4. When the auto encoder shows a satisfactory result we take the learned weights from the encoder layers and use them in the model. The weights of these encoder model are now set fixed and not anymore changed in when training the rest of the model.

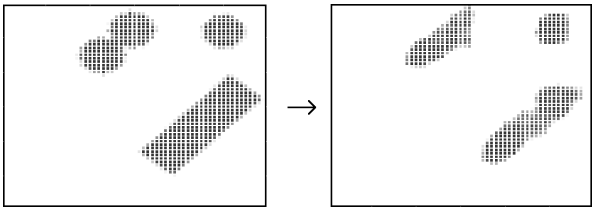


Fig. 4: Auto encoder input and output grids

C. Planing towards the goal

To make the network plan towards the goal position we define a vector v pointing from the robot's position towards the supplied goal position. All the inputs are rotated by negative the angle between vector v and the global map's x-axis such that they are expressed in the frame with the vector v as x-axis. This ensures that as seen in this frame the goal always lies on the x-axis. In this way the neural network is trained to always try to drive the robot in the direction of the x-axis as much as possible. This rotation is easily done on the radial pedestrian grid an on the state, but is more complex to perform on the occupancy grid. To make sure the rotation does not leave us with cutoffs and unknown area's in the grid, first a bigger selection is made. This section is rotated and then cut to the required size as seen in figure 5. The output trajectory of the neural network is rotated back by the same angle to the global frame. The rotation trick makes for a much more homogenenous data set, where the network just has to learn to plan towards the right with obstacle avoidance. We proof this has better performance than adding the relative goal as another input. We choose to center the occupancy grid on a point that lies 2 meters in front of the robot in the direction of the goal instead of on the robot itself. This gives more the information about the environment in front of the robot, which is more valuable than the information about obstacles behind the robot.

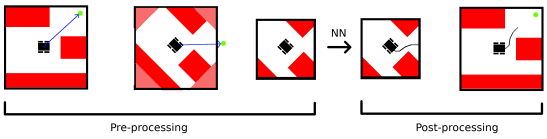


Fig. 5: Rotation of the occupancy grid to the goal

D. Training data

First we tried to use a dataset of real humans walking around. There is no dataset available which has both the human interactions and obstacles avoidance we collected a dataset our self. In a lab of 10x10 meters a table is setup at each side a the room and an some obstacles are placed in the center. The people are instructed to solve four puzzles of which the pieces are spread out over all the tables. They are allowed to only take one piece of the puzzle at the time. This makes sure people there is a lot of walking in between the different tables, with a changing combination of obstacle avoidance and interaction with the other humans. The people quickly become occupied with the task so they will no longer pay attention to the experiment and show natural walking behavior. The network however did not produce good results with this dataset. The most likely reason was that the dataset was much to small. We checked this by training the network on a portion of the simulation dataset with the same size as our real dataset, this also did not give a working planner network. More information about the experiment setup and filtering can be found in the appendix.

Instead we use a dataset generated by a simulation of 15 pedestrians walking around in an area with small obstacles. The behavior of the pedestrians is simulated by a force based

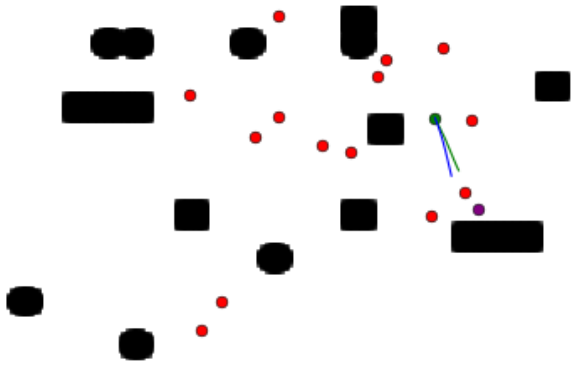


Fig. 6: Still from dataset used for training

simulation that includes social forces based on the research by Helbing [11] to simulate real human behavior. A still from this dataset can be seen in figure 6, where each red dot is a pedestrian, the green dot is the ego pedestrian who is moving to the purple goal position. The data set is split into two parts: 90% of it is used for training and the other 10% is used for testing. This is done such that we can check if the model is not over trained on the training set. For each trajectory in the set, at the goal position 20 positions are appended where the pedestrian is standing still on the goal position. This is done such that the planner will better learn it has to stand still when it reached its goal.

E. Validation

In table I the loss of the neural network on the test set can be seen. The neural network is compared to a planner that simply plans straight to the goal and has the same velocity as the people from the data set. The loss from the neural network is lower than the loss from the simple planner. This is a good indication that a neural network learned grasps the structure of the data and is able to mimic the pedestrian trajectories in similar cases. This however does not give guarantees that it actually produces reasonable trajectories. To if the network is capable of steering the robot we run a simulation where the robot is put in an environment with many obstacles. The neural network produces a trajectory at a rate of 10Hz and the first velocity of each of the trajectory plans is executed. The resulting trajectory can be found in figure 7. For this figure it can clearly be seen that the robot is able to navigate from its starting point (green dot) to the goal (red dot) and will avoid the obstacles in its way. In chapter V we will show more experiments and results for the neural network planner.

	Loss on test set
Neural network planner	0.11
Straight to goal planner	0.29

TABLE I: Loss of planner neural network

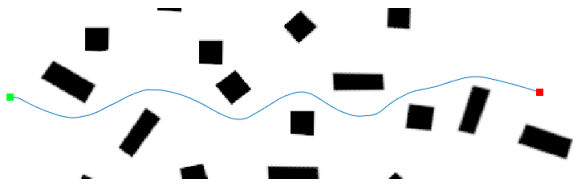


Fig. 7: Neural network planner avoiding static obstacles

Different variations of the network architecture have been tried, such as having the relative goal position as input to the network instead of using the frame rotation strategy, using Cartesian or polar coordinates for the state representation, centering the grid on top of the robot or shifted to the front and different representations for the pedestrians. A full explanation of these variations and their results can be found in Appendix A.

We noticed that the amount of avoidance the neural network showed with pedestrians is small. For further use we made some adjustments to how the pedestrians are added to the grid such that effect of humans on the planner network is larger. The pedestrians are added to the occupancy grid as completely black circles instead of Gaussians and their position was calculated 1.5 seconds into the future using a constant velocity assumption so that the neural network planner starts avoiding the pedestrians earlier.

IV. Method: Combining MPC and NN

Our method uses a neural network for planning a local trajectory towards the goal taking into account the static world and dynamic pedestrians. This planned trajectory is used in combination with a model predictive controller. We discuss four different ways how to make this combination. The first way is to use the neural network trajectory to guide the MPC through its cost function, we can use the neural network to initialize the MPC and we try using the neural network trajectory to initialize the search routine for the static free space around the robot. Besides that, we can use a separate neural network to predict to path of the pedestrians in the environment and use this prediction to restrict the MPC. An overview of the whole system is found in figure 2. In the next sections we discuss each of the parts.

A. Guiding MPC using NN

The neural network is integrated into the MPC cost function in such a way that the MPC is encouraged to follow the neural network generated trajectory but is still able to deviate when the restrictions on the MPC ask for it. We designed a cost function which motivates the mobile robot to follow a position reference provided by the NN, $\mathbf{x}_{\text{nn}} = [x_{\text{nn}}, y_{\text{nn}}]$, and a velocity reference v_{nn} calculated from $\mathbf{x}_{\text{nn}}$ and the timestep.

$$J(\mathbf{x}_k, v_k) = J_{\text{tracking}}(\mathbf{x}_k) + J_{\text{vel}}(v_k) \tag{4}$$

$$J_{\text{tracking}}(\mathbf{x}_k) = Q_x||x_k - x_{\text{nn}}||_2^2 + Q_y||y_k - y_{\text{nn}}||_2^2 \tag{5}$$

$$J_{vel}(v_k) = Q_{vel}||v_k - v_{\text{nn}}||_2^2 \tag{6}$$

where $\{Q_x, Q_y, Q_{vel}\}$ are the design weights.

When investigating alternatives for this formulation we also tried a different way to guide the heading of the robot. Instead of the error in the position we can use the error in the heading of each point in the planning horizon. We also investigated if using a fixed reference velocity instead of the velocity produced by the neural net, more tests and results on this can be found in appendix C.

B. Initialization of MPC optimizer

The initialization of the optimization strategy could be of big importance. The optimization method relies on gradient descent and is therefor prone to local minima. On top of that the online setting dictates that we have only very limited time to come to the next control action and can therefor

only do a limited number of optimization iterations. A good initialization of the optimizer is therefor vital. The usual way to do this is to initialize the MPC with the previous result, where the timestep is shifted one step to account for the progress of time. This causes optimization of the new step to start looking in the same direction as the previous step. This can cause to the optimization to get stuck in a local minima, with a much better alternative around. Therefor we choose to initialize our optimization with the trajectory generated by the neural network. This allows the optimization to easily switch its plan to the other side of an obstacle if the neural network decides that is a better option. Note typically warm staring of an MPC is done by initializing the MPC with results that have come from offline optimizations. Typically these optimizations can have a longer running time or can look further ahead. In our experiments however the initialization does not come from MPC results but from a neural network trained on human data.

C. Static free space search

The model predictive controller needs to take the static obstacles into account when planning its trajectory. These static obstacles are stored in the form of an occupancy containing the obstacles that are found during the mapping operation plus the obstacles that are found with the LiDar during operation. The model predictive controller is limited to find a solution within a free rectangle. These rectangles are calculated by taking the previous trajectory and expanding a rectangle on each of the trajectory points until it hits an obstacle. More details on the free static space search can be found in Appendix D. The drawback of this method is that the free space is only calculated in the neighbourhood of the previous MPC solution. If the dynamic environment dictates a better trajectory around the other side of a static obstacle this free space search routine does not allow that. As alternative we start the static free space search on the trajectory points generated by the neural network. This allows the static free space to jump to the other side of a static obstacle if the neural network finds that to be the better side.

D. Neural network for pedestrian predictions

The robot needs to be able to plan its trajectories with many people walking around the robot. We are using a model predictive controller, which optimizes its planned trajectory over a planning horizon. This is only possible when we have information about how the environment is going to look in the future. It is fine to assume the static environment is not going to change, but that is not the case for the dynamic part of the environment. Pedestrians are quite hard to predict, because an uninformed observer has no knowledge about where they are going to. Besides that there are numerous reasons why a pedestrian would follow an unexpected route, perhaps they forgot something and turn in a different direction or they walk into a friend and stop to make a chat. It is unthinkable that a robot would be able to predict these situations exactly. In the end this does not have to be a problem, since humans among each other often cannot predict this either and they are able to navigate through busy hallways. It is however useful to predict the pedestrians as good as possible, because in that case the planned path of the robot is still valid in the next time steps which leads to a smoother drive and a more optimal route.

A constant velocity prediction can be used to predict the pedestrians walking in the neighbourhood of the robot. The velocity of the pedestrians is calculated from the a number of position measurements and smoothed using a Kalman filter. If a pedestrian is walking towards a wall, a constant velocity prediction will predict that the pedestrian will walk through the wall. This is however very unlikely, a human is able to derive from the situation that the pedestrian is probably going to make a turn. Instead of the constant velocity assumptions we use a neural network to predict the pedestrians. The network is similar in structure and training to our planner network seen before. The difference is that now the inputs are in the frame of the walking direction of the pedestrian instead of towards to goal, and there is no goal distance input. The network is trained on the same data set as the planner network. We will experiment with using this neural network prediction instead of the constant velocity model as a restriction on the model predictive controller. The pedestrian predictions do not have to be not used in the neural network planner because it can create its own understanding of pedestrian interactions.

E. Validation

Several experiments where done to investigate which of the combination strategies between the neural network planner and the model predictive controller give an improved result. The simulation tests and results can be found in appendix C. We found that both the guidance and initialization of the MPC by the neural net gives improved results in terms of a low number of collisions and a quick route to the goal. We found that initializing the static free space search with the neural network generated path does not give satisfactory results because it fails when the neural network plans slightly through a static obstacle. In that case the free space search could no longer come up with a free space and the MPC would stop the robot. We also found no improved results by using a neural network to predict the pedestrians over using the constant velocity prediction. This could be because there is only a limited amount of interaction in our model and therefor the two method of predictions are very similar. Also the predictions are trained on pedestrian to pedestrian interaction and not on pedestrian to robot interactions. In the following experiments we use the guidance and initialization using the neural network and a constant velocity prediction for the pedestrians.

V. Simulation experiments

Several experiments are performed in simulation to find the best configuration of the system and to make a fair comparison with existing approaches.

A. Software setup

The simulation is based on ROS Gazebo [12] where the model of the robot is supplied by its manufacturer Clearpath [13]. The pedestrians are not modeled into gazebo but their positions are calculated using the open source PedSim package [14]. This package is meant to simulate large crowds, for example in the case of evacuation. However it can also be used for individual pedestrians. PedSim allows to define obstacles in the form of walls and also the start and goal positions for each pedestrian. It uses a force model to determine the walking direction of the pedestrians which includes an attractive force to the goal and to pedestrians within it own group and a repulsive force from obstacles and other pedestrians. The simulated robot is inputted as a pedestrian such that the forces of the other pedestrians take the robot into account as if it was just another pedestrian. The used social forces model is not extremely accurate but it is light weight and works sufficiently for its purpose. Since we don't forward the pedestrian locations to the simulations it is necessary to check for collisions ourselves. A collision is defined when the distance between a robot and a pedestrian is less than 0.4 meters.

B. Scenario

First we test two scenario's where previous LMPCC method was already performing well to ensure the performance of the new method is this working in those situations. The first scenario is a pedestrian crossing a hallway from the side, and a robot has to decide to avoid by going in front of the person or behind it as seen in figure 8. The second scenario is a hallway where three people approach the robot, as in figure 9. The pedestrians have a separation of 6 meters and their vertical position is randomly drawn from a uniform set of -2 to +2 meters around the center of the hallway.

The third experiment is similar to the second experiment but now there are three pairs of pedestrians coming from the opposing side. We found that the LMPCC has difficulties in such a scenario. The scenario can be seen in figure 10. We will run four different control strategies on this scenario. We will compare our proposed NN-MPC with the previously developed LMPCC method. The third method is 'straight to goal' where the MPC is guided by a straight line from to current position to the goal through the same cost function as is used for the NN-MPC. This should give us insight if it is necessary to use a neural network for guidance. The fourth method is 'neural net only' where we do not use the MPC to steer around obstacles and only steer the robot using the neural network generated trajectory. This should give insight if the obstacle avoidance of the model predictive controller gives an added advantage. In these experiments the pedestrians are non-cooperative. This means they act walk as if the robot is not there. If we would make the pedestrians cooperative the fastest method would be for the robot to just drive straight to goal and let the pedestrians move out of the way for him, our main metric of comparison is the number of collisions so we need our simulated pedestrians to not avoid these collisions by them self.

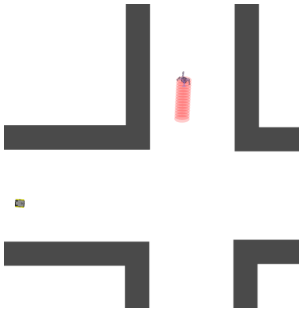


Fig. 8: Scenario 1, intersection with crossing pedestrian

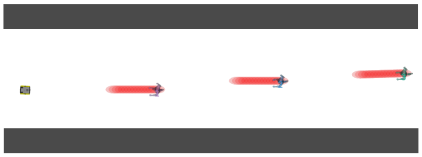


Fig. 9: Scenario 2, hallway with opposing pedestrians

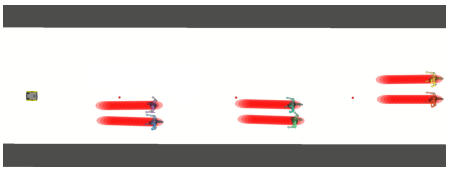


Fig. 10: Scenario 3, hallway with pairs of opposing pedestrians

	Successful runs	Crashes	Time outs	Time to goal	TtG on 2	TtG on 3	TtG on 4	Average solve cycles
1. NN-MPC	50	0	0	15.3	15.5	15.1	15.5	2.4
2. Straight to goal	20	30	0	20.5	20.5	-	-	3.5
3. Neural net only	20	30	0	14.6	-	14.6	-	-
4. LMPCC	27	22	1	24.0	-	-	24.0	4.0

TABLE II: Results of the simulation experiment 3

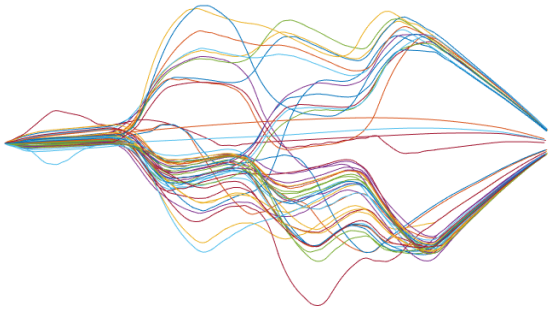


Fig. 11: NN-MPC trajectories of the simulation experiment

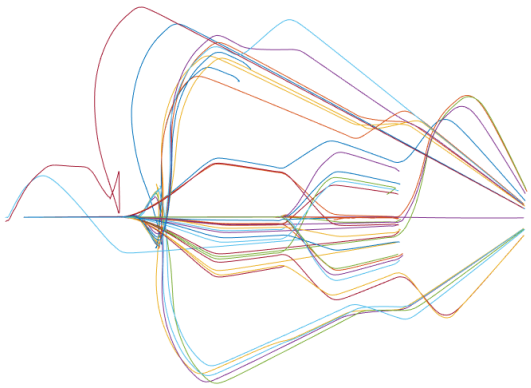


Fig. 12: Straight to goal guidance trajectories of the simulation

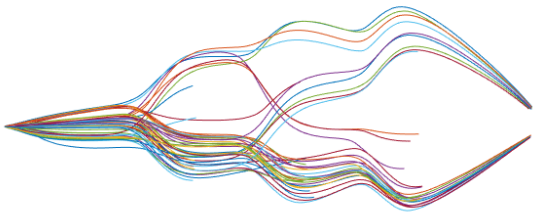


Fig. 13: Neural network only trajectories of the simulation experiment

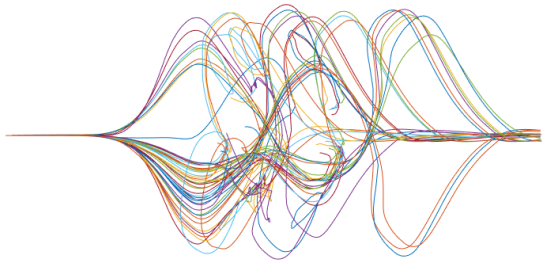


Fig. 14: LMPCC trajectories of the simulation experiment

C. Results

The first two scenario's where both 100% successfully by both the nn-mpc and the lmpcc. We will not go into details here, the trajectory's can be found in Appendix A. The results of the third experiment can be found in table II, the followed trajectories in figure 11 to 14. The NN-MPC is able to complete the scenario successfully 100% of the time and is nicely able to steer around the pairs of pedestrians and move to the side of the hallway when necessary. The LMPCC only has 54% succesful runs, it has trouble because of conflicting cost terms from the pedestrians and the guidance path. It makes the robot drive backwards and the optimizer cannot come to feasible solutions from time to time which causes collisions. Both the straight to goal guidance and the neural net only guidance do not perform well and are unable to avoid the pedestrians very often. From these results it is clear that the combination of the neural network and the MPC is essential for the method to work and they re-enforce each other. We also calculated the average time to goal over the successful runs. To make a fair comparison the time to goal for the NN-MPC is also calculated over the instances where the other methods had successful runs, of which te results can be seen in the columns marked 'TtG on 2', 'TtG on 3' and 'TtG on 4'. It can be seen that NN-MPC is much faster than LMPCC, even though the robot drives with the same velocity. This is because the LMPCC drives a longer route to avoid the pedestrians. The neural network only method has a similar time to goal as the NN-MPC on successful trajectories, this is expected because the trajectories where the neural network only is successful is almost identical to what NN-MPC produces in those situations.

D. Case studies

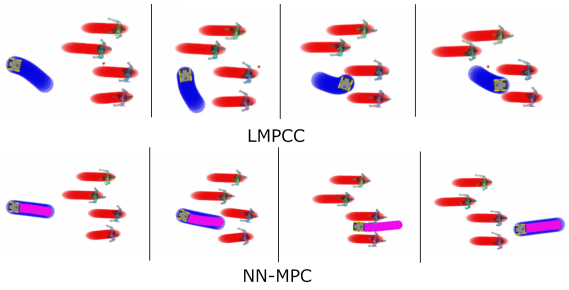


Fig. 15: Row of people approaching robot, comparison between guidance strategies. Red: pedestrian prediction, pink: NN generated trajectory, blue: MPC solution

To illustrate some other advantages of NN-MPC we take a closer look at two scenario's. In the first scenario we look at scenario where four pedestrians are approaching the robot as can be seen in figure 15. In this scenario the pedestrians are cooperative, that means they see the robot as another person and will move around the robot by their social force. The LMPCC is unable to pass the pedestrians because it has a cost that increases when it gets closer to a pedestrian. It will not see the possibility of the pedestrians cooperating

and making room for the robot. The Neural network does see the possibility. The pedestrians cooperate slightly and let the robot through, in the case that the pedestrians would not cooperate enough, the mpc restriction would predict a collision and stop the robot until the pedestrians pass along sides and the robot can continue driving again.

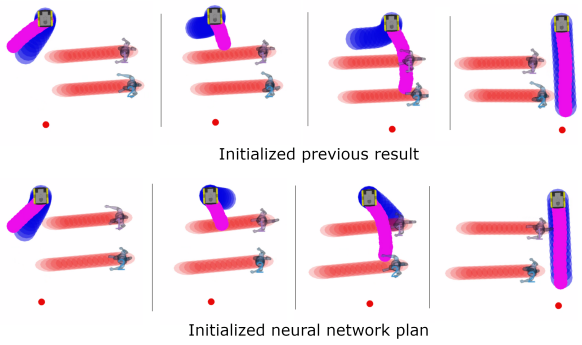


Fig. 16: Pedestrians cross robot path, comparison between MPC initialization strategies. Red: pedestrian prediction, pink: NN generated trajectory, blue: MPC solution

To illustrate what effect initialization with neural net has, instead of initialization with the previous MPC result we show the scenario in figure 16. Pedestrians are crossing the path the robot wants to take. When initializing with the neural network the MPC switches to going behind the pedestrians much sooner, while the alternative sticks to the solution where it wants to go in front of the pedestrians longer, driving around the pedestrians to long way around.

VI. Real world proof of concept

A. Hardware Setup

For the real world proof of concept of this research a fully autonomous ground vehicle (AGV) is used. The platform is based on Clearpath's Jackal, extended with several sensors and computer modules. The Jackal is a four wheeled differential drive robot capable of outdoor use. The two left wheels and the right wheels are mechanically connected and will spin at the same speed. Through differential drive and sideways slippage of the wheels the robot can turn. The Jackal is equipped with an onboard computer, inertial sensors (IMU), encoders, GPS, Wifi and Bluetooth. It is fully integrated with the robotic operating system (ROS) and has a built in controller that takes velocity commands. To make the platform capable of autonomous navigation more precise localisation is necessary. For this purpose a Velodyne 64 layer LiDar is added on top of the robot. A Stereolabs ZED stereo camera equips the robot with the capability to distinguish humans from other obstacles. This is useful such that a specialized pedestrian predictor can be used on them. The Jackal is already equipped with an Intel i5 processor running at 2.6Ghz. This computer is tasked with running the controller and localization modules. An extra Intel i7 NUC mini PC is added to run the motion planner and pedestrian prediction nodes.

Fig. 17: Real robot experiment.

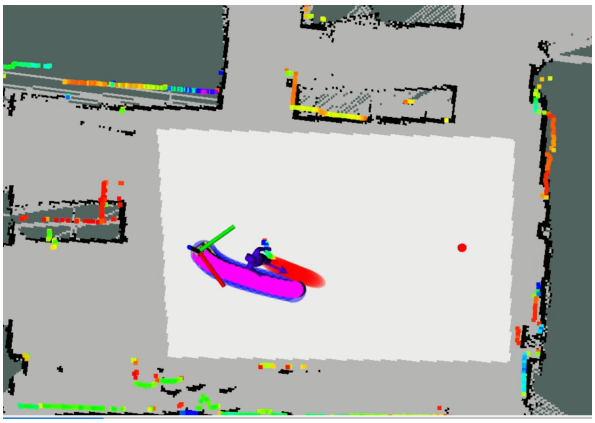


Fig. 18: Real robot experiment rviz overview.

In the lab experiments we want to exclude any errors caused by inaccurate perception. Instead of the on board sensors we make use of the OptiTrack motion capture system during the lab experiments. This system uses a setup of many camera's attached to the ceiling of the lab room. The system is able to track marker dots, where three dots in different planes are enough to measure the position and orientation of a rigid body. The robot is equipped with several of these markers and pedestrians are asked to where a helmet with three tracking dots attached on top of them. A ROS node forwards the frame transformations and functions as the bridge between the OptiTrack system and the motion planning running on the robot.

B. Software Setup

Several external packages are used to operate the robot. The low level control of the jackal is done with the open source ROS joint state controller / diff drive controller. This controller is velocity based and uses the robots odometry as feedback. On board localisation is done by first making a map of the environment using ROS GMapping which uses a simultaneous localisation and mapping (SLAM) approach on the laser data. After the mapping phase localisation is done using the ROS AMCL node, which uses adaptive Monte Carlo localization approach with a particle filter to localize itself on the map. This position is fused with information

gained from the inertial sensors and wheel encoders using the ROS robot pose EKF (extended kallman filter) package.

To recognize the pedestrians in the environment of the robot an open-source pedestrian tracker SPENCER is used. This uses both the RGB-D camera and laser range finder to identify, localize and track pedestrians. These pedestrians are removed from the static map and a prediction is made for their trajectory. These predicted trajectories are inserted as dynamic obstacles into our planner.

C. Results

We were able to make the robot drive to the goal while evading a pedestrian walking through the path of the robot. In figure 18 a screenshot of RVIZ corresponding with the situation in 17 is shown. The rainbow colored dots are the laser measurements which are matched to the walls and other obstacles to localize the robot. The people detector is limited to the light grey area to lower the number of false recognitions. From the picture can be seen the robot has recognized the pedestrians and predicted its future trajectory. The neural network has taken the pedestrian into account and generated a path around the pedestrian. The MPC confirmed that path is clear of collisions and therefor mimics the neural network generated path exactly.

VII. Conclusion

We presented a solution to the planning problem in dynamic and unknown environments for a robot navigating in human populated area's. We learned a neural network to generate a plan based on the static and dynamic obstacles around the robot and combined this plan with a model predictive controller that is able to adjust the plan of the neural network if it would collide with the obstacles. This gives a solution that is both very aware of the surroundings and flexible in its plan generates as well as safe for operation in real environments. We have seen that guidance of the MPC with a neural network planner improve its behavior. Using the neural network trajectory as initialization of the MPC is shown to improve the flexibility of the method and makes it easier to adjust its trajectory to a quickly changing environment.

From the simulations can be seen the planner is able to navigate through a hallway with oncoming pedestrians without problems, while using only the neural network or only a LMPCC causes collisions. With the real experiment we demonstrate it is possible to run the whole perception, planning and control pipeline in real time.

For further research we recommend going further in depth in ways to model interactions of pedestrians and the robot into both the planner and prediction neural networks. This will also require obtaining new data, preferable in a real environment with actual pedestrians. We also recommend looking into other ways to define and calculate the static free space around the robot. Additional discussion and research recommandations can be found in Chapter 3.

References

[1] W. Schwarting, J. Alonso-Mora, and D. Rus, "Planning and Decision-Making for Autonomous Vehicles," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, no. 1, pp. 060 117–105 157, 2018. [Online]. Available: http://www.annualreviews.org/doi/10.1146/annurev-control-060117-105157

[2] S. S. GE, "dynamic motion planning for mobile robots using potential field method," *Proceedings of the 8th IEEE Mediterranean Conference on Control and Automation*, Jul 2000.

[3] F. Zanlungo, T. Ikeda, and T. Kanda, "Social force model with explicit collision prediction," *Europhysics Letters*, vol. 93, no. 6, p. 68005, 2011.

[4] M. Seder and I. Petrovic, "Dynamic window based approach to mobile robot motion control in the presence of moving obstacles," *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 2007.

[5] P. Trautman, J. Ma, R. M. Murray, and A. Krause, "Robot navigation in dense human crowds: Statistical models and experimental studies of human-robot cooperation," *International Journal of Robotics Research*, vol. 34, no. 3, pp. 335–356, 2015.

[6] H. Kretzschmar, M. Spies, C. Sprunk, and W. Burgard, "Socially Compliant Mobile Robot Navigation via Inverse Reinforcement Learning," *The International Journal of Robotics Research*, vol. 35, no. 11, pp. 1289–1307, 2016. [Online]. Available: http://ijr.sagepub.com/content/early/2016/01/04/0278364915619772

[7] M. Pfeiffer, U. Schwesinger, H. Sommer, E. Galceran, and R. Siegwart, "Predicting actions to act predictably: Cooperative partial motion planning with maximum entropy models," *IEEE International Conference on Intelligent Robots and Systems*, vol. 2016-Novem, pp. 2096–2101, 2016.

[8] Y. F. Chen, M. Everett, M. Liu, J. P. How, and R. O. May, "Socially Aware Motion Planning with Deep Reinforcement Learning," *International Conference on Intelligent Robots and Systems (IROS)*, pp. 1343–1350, 2017.

[9] B. Brito, B. Floor, L. Ferranti, and J. Alonso-mora, "Model Predictive Contouring Control for Collision Avoidance in Unstructured Dynamic Environments," 2019.

[10] G. E. Hinton and R. S. Zemel, "Autoencoders, Minimum Description Length and Helmholtz Free Energy," in *Advances in Neural Information Processing Systems 6*, J. D. Cowan, G. Tesauro, and J. Alspector, Eds. Morgan-Kaufmann, 1994, pp. 3–10. [Online]. Available: http://papers.nips.cc/paper/798-autoencoders-minimum-description-length-and-helmholtz-free-energy.pdf

[11] D. Helbing and P. Molnár, "Social force model for pedestrian dynamics," *Physical Review E*, vol. 51, no. 5, pp. 4282–4286, 1995.

[12] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, 2004.

[13] "Mobile robots for research development." [Online]. Available: https://clearpathrobotics.com/

[14] "Pedsim." [Online]. Available: http://pedsim.silmaril.org/

Chapter 3

Concluding Remarks

3-1 Discussion

3-1-1 Limited experiences in the dataset

The datasets we used for training the planner model are limited in its diversity. In general machine learning works well on data that is close to its training data but generalizes less well to cases that are much different from the training data. The datasets we used consist of multiple pedestrians walking each to their own goal with relatively small obstacles in their way. For scenario's that are much different we are certain that the robot will behave safely, but it is uncertain if the robot will behave as a human would in that situation. Examples are the behavior around groups of people walking together and behavior around bigger unexpected obstacles. A solution is to collect more diverse data geared towards the environment in which the robot is expected to operate.

For this research we use dataset that consists of humans interacting with each other, we assume that the interaction between humans and the robot will be similar. This isn't necessarily the case. To improve the interaction model, data has to be collected by having actual humans interact with the robot in real life. The behavior of the people around the robot is expected to depend on the appearance of the robot so using data collected by a different robot might not be representative. The behavior of the surrounding people will also depend on the behavior of the robot itself. This creates a loop where the robot will adapt its behavior based on its experiences and the surrounding agents will adapt on that again. This requires several experiments of data gathering before the system comes to an equilibrium.

3-1-2 Using a real dataset

In this research we worked with a dataset obtained from simulated pedestrians. This is done because we can easily generate a very big dataset of simulated pedestrians while it is hard to obtain a similar size dataset of real pedestrians. During the data gathering experiment we

also found that gathering real data requires a lot of filtering and manual work to get a clean dataset. Another challenge with real data is that people are inherently random. Even in a controlled environment people will walk the same start to finish trajectory in many different ways. A neural network needs even more data to be able to generalize over these different trajectories.

On the other hand, using a real dataset would increase the argument of the NN-MPC method a lot. In the current method the neural network only mimics the rules of simulation it was trained on. In fact the simulation rules could have been used in the MPC directly. The neural network does has the advantage that it decreases the query time and is able to directly finds a trajectory for the whole horizon, while the simulation only finds the movement direction for the current time step. By using a neural network we did show it is possible to integrate one into a receding horizon controller to result in a good planner.

3-1-3 Limited number of agents in MPC

In our implementation of the model predictive controller each pedestrian adds a restriction on the solver. It restricts the position of the robot to be outside of the predicted position of the pedestrian. The restrictions have to be defined beforehand and only the parameters of each restriction can be changed online. With these parameters it is possible to disregard some of the restrictions. So it is possible to insert less pedestrians into the solver than we defined beforehand, but it is not possible to insert more.

With every additional restriction on the solver, it will take more time to do a optimization step. We need a number of optimization steps to get to a feasible trajectory that is close enough to the optimal result, we also need to have a high frequency of output trajectories for our controller to follow. These two restriction result in a maximum number of pedestrians we can take into account before the calculations get too slow. In our setup we found this maximum to be six pedestrians. It is very likely that the robot is going to operate in an environment with more than six pedestrians around, in this case we are only inserting the six closest pedestrians into the solver. This works well until there are more than six pedestrians in very short proximity from the robot, now we cannot guarantee that we will evade all of the pedestrians around.

For safety we programmed the robot to stop driving in the case that there are more than six pedestrians very close to the robot. But this is not desirable behavior to happen regularly. A solution has to be found so more pedestrians can be taken into account. One method is by reducing the planning horizon, but this will result in a less optimal trajectory in the global sense. Another way would be to choose a faster solver package. We have indications that the package that we used (ACADO) is not the fastest available and we could switch to another package, such as Forces Pro.

3-1-4 Tunability of behavior

This research is based on finding a better planner and controller combination by learning from human behavior. This means the behavior of the robot will be directly based on the dataset of examples supply to it. Besides changing the dataset we have very limited amount

of control to manually tune the final behavior of the robot. One characteristic that we might want to tune is the aggressiveness of the robot. This aggressiveness can be best described by an example: if the robot is driving behind a person walking rather slowly, does the robot need to overtake or does it need to stick behind the person, for which speed does it need to do what? And when operating in a busy hallway, to what extend can the robot expect the other people to move out of the way? In our research these decisions are all made by copying the behavior found in the human data.

This raises the question: do we actually want robots to behave in a similar way as humans do? This all depends on how robot will be perceived in the future, will they be purely machines that are submissive to human needs or will we attribute some emotional attachment to them, no definite answer can be given to this question at this point. Besides that the allowed aggressiveness might be different for the purpose of the robot. A robot that is driving an AED to a patient with cardiac arrest is allowed much more aggressive driving than a robot that has all day to clean the floor.

In our current setup it is possible to put some limitations on the robot behavior through the model predictive controller, we can put limits on the linear and angular velocity and accelerations. But the core behavior has to come from the neural network planner, which is purely based on the training data. One way to influence this neural network planner is to give it an additional inputs that specify the required aggressiveness to the network. When this aggressiveness input is set low the model should come up with a submissive trajectory and when its set to a high number the trajectories should be very aggressive. To learn this behavior it has to be labeled in the data, this can be done either manually after the data is recorded or by instructing the test subjects to behave in a more or less aggressive manner.

3-2 Conclusion

In this work we demonstrated it is possible to combine a deep neural network planner with a model predictive controller in a successful way. The neural network has the advantage it can come up with a good trajectory taking into account obstacles in the environment and the inference of the model is fast. The model predictive controller makes sure the generated trajectory is safe over the planning horizon and can enforce the kinematic constraints. The combination of the two is shown in simulation and real experiments to be a promising option for the combined planning and control problem in dynamic and unpredictable environments. All the listed discussion points have viable solutions, which can be researched and implemented.

3-3 Further work

There are several ways this work can be continued. The first essential question is if we need to switch to training the neural network on real pedestrian data. Using real data has the advantage that actual interactions can be learned which improves accuracy of predictions and can include more social characteristics in the planning behavior. The difficulty with real data is that it is very diverse and has some apparent randomness which is not realistic to capture in a model. The network needs to be complex to capture much of the reasoning in

the trajectories. As alternative to reduce the necessary complexity data can be gathered in a restricted lab experiment. The environment can be designed and the people instructed such that they only show certain types of behavior. The robot will then learn to perform that behavior.

Research needs to be done for ways to calculate the static free area around the robot and restrict the model predictive controller to calculate its solutions inside this area. The current solution of the expanding rectangle is too restricting. The method should be fast because it needs to be recalculated every cycle. Also the restrictions it poses on the solver should be rather simple such that it does not slow down the optimization iterations.

A standard method need to be developed which can give a quantitative judgement to quality of a social motion planning solution. Factors like social compliance are too vague and different solutions are very hard to compare because they are tested under vastly different circumstances and conditions. The number of collisions is a clear metric but we now advance into an area where have multiple methods that can drive to a goal without collisions but one has a more favorable path than the other. Metrics like path length, time to goal and average distance to pedestrians are not conclusive. If the goal of the planner is act close to how a human would an adversarial network can be used to judge if the trajectories are indeed similar to what a human would do. A different way of judging the performance of a planner could be by doing a study where a group of participants is asked to rate the robot behavior as a sort of user study.

A different approach to continue this work would be to decrease the task of the neural network. Instead of the output being a trajectory it could also be a binary decision, such as: should I overtake or not, and should i overtake on the left or on the right of this pedestrian. This decision has to be made taking into account all the pedestrians in the scenario including their walking direction and speed. After this, the MPC should be encouraged to come up with a trajectory that meets this decision. This lowers the complexity of the neural network, because it does not have to deal with dynamic obstacle avoidance anymore.

Appendix A

More results

We tested the NN-MPC on three simulation scenario's. The first simulation scenario in figure A-1 is an intersection the pedestrian is approaching from the left as seen by the robot. The scenario is run 5 times where the pedestrian starts at 5 different y position's 1 meter apart so that it arrives at the intersection at different times. The robot has to decide correctly if it can pass in front of or behind the pedestrian. Experiment 2 is a hallway with three pedestrians approaching the robot as in figure A-2. The horizontal spacing between the pedestrians is 6 meters. The vertical position in the hallway is chosen from a uniformly random distribution. The experiment is executed 50 times to test on enough pedestrian positions.

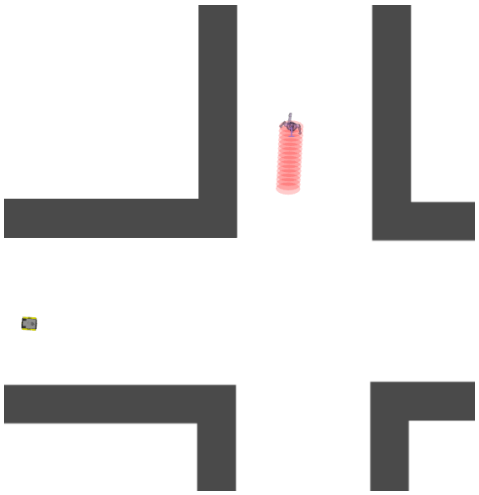


Figure A-1: Experiment 1, intersection

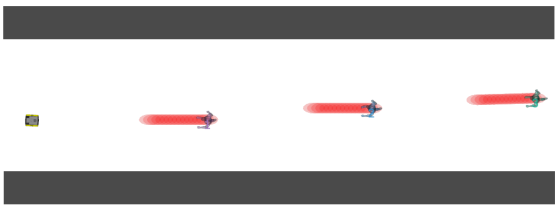


Figure A-2: Experiment 2, hallway three people approaching

The trajectories of the first experiment using the NN-MPC method can be found in figure A-3 and using LMPCC in figure A-2. Both are sucesfull in reaching the goal position for all pedestrian positions. The LMPCC is a bit smoother, but goes in front of the pedestrian where is was better to go behind. It has to accelerate a bit to make sure it does not come

to a collision. The LMPCC goes back to its reference trajectory quickly, while the NN-MPC takes a new straight path to the goal position.

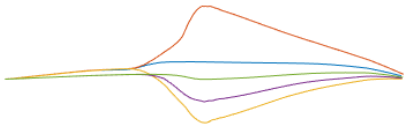


Figure A-3: NN-MPC trajectories on experiment 1.

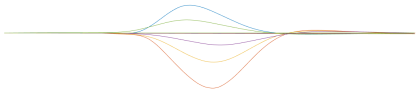


Figure A-4: LMPCC trajectories on experiment 1.

The results for the second experiment for the NN-MPC is found in figure A-5 and for LMPCC in figure A-6. Both the methods are succesfull in all random placements of the pedestrians. Again the LMPCC tends to return to the reference trajectory, while the NN-MPC creates a trajectory that points to the goal position.

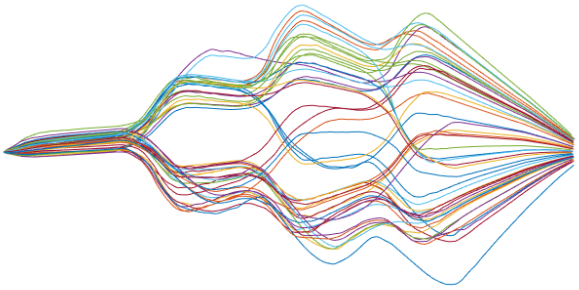


Figure A-5: NN-MPC trajectories on experiment 2.

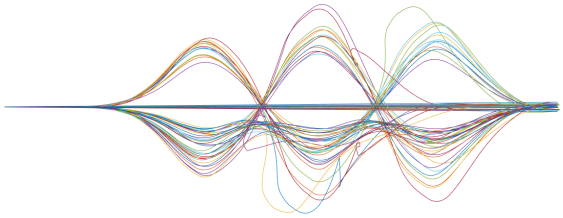


Figure A-6: LMCPC trajectories on experiment 2.

Appendix B

Neural network variations

In our search for better network architectures we tried several approaches to improve the planning behavior of the neural networks. These alternatives did not provide better results in when we tried them, but the ideas could be useful for further research.

B-1 Variations

There are many parameters in the neural network setup that have multiple sensible options. It is impossible to compare all of them, but we selected the most interesting ones and made a comparison.

State input in Cartesian coordinates or Polar coordinates. The state input gives the neural network an information about the velocity and direction of the robot. Previously the velocity was expressed as Cartesian coordinates (x and y velocity), but these have the disadvantage that they do not express the orientation of the robot when it's velocity is zero. Instead we want to use polar coordinates which expresses the velocity as an orientation and absolute velocity, such that we do have orientation information at zero velocity. We make a comparison to make sure the network performance is still good when using polar coordinates. (Variation 2)

Front shifted grid. The occupancy grid was previously centered on the position of the robot. This gives an equal amount of information in front of and behind the robot. In the experiments we found that the neural network can sometimes not look far enough ahead to take into account the obstacles that are in front of him. The information behind the robot is much less relevant because the chance that is has to go backwards is much smaller than it going forward. Simply increasing the size of the occupancy grid has the drawback of increasing the complexity too much, therefor we tried the approach of centering the occupancy grid several meters in front of the robot. (Variation 3)

Rotate to goal or goal as input. To make sure the neural network plans towards the goal position, all the inputs are rotated towards the goal position and the result is rotated back

to the global frame. We compare this to the approach where input the goal into the neural network as relative x and y positions or in polar expression (Variation 6 and 7). We also see what happens if we rotate the frame with the orientation of the robot. (Variation 4 and 5)

Representation of agents. Lastly we try different methods of inputting the surrounding pedestrians to the neural network. One way is to use the radial pedestrian grid, another way is to use the pedestrians as obstacles in the occupancy grid. This can be done using a Gaussian distribution centered on the pedestrian. We try both possibilities and the combinations of them. (Variations 8, 9 and 10).

B-2 Results

The experiments variations and results can be found in table B-1. From variation 2 we can see that the way the state is defined does not give noticeable difference on the network performance, the same goes for variation 3 in which the occupancy grid is centered on top of the robot instead of in front of it. From variations 4 till 7 we see setting the goal as an input instead of rotating the frame towards the goal performs much poorer, both when using a Cartesian and polar representation for the goal position. This is the case both for the non rotated frame and for the frame rotated in the direction of movement of the robot.

From variation 8 we see that putting the pedestrian on the occupancy grid or not has no influence as long as we still have the radial pedestrian grid. If we remove the radial pedestrian grid however the losses go up as can be seen from variation 9 and 10.

	state	occ grid center	frame rotation	goal input	ped on occ grid	ped grid		train loss	test loss
1: Proposed method	polar	front	to goal	distance	yes	yes		0.11	0.11
2: Cartesean state	cartesian	front	to goal	distance	yes	yes		0.11	0.10
3: Centered grid	polar	center	to goal	distance	yes	yes		0.11	0.11
4: Rotate with ped, goal cartesian	polar	center	robot heading	cartesian	yes	yes		0.15	0.15
5: Rotate with ped, goal polar	polar	center	robot heading	polar	yes	yes		0.13	0.13
6: No rotation, goal cartesian	polar	center	no rotation	cartesian	yes	yes		0.14	0.14
7: No rotation, goal polar	polar	center	no rotation	polar	yes	yes		0.14	0.13
8: Ped not on occupancy grid	polar	front	to goal	distance	no	yes		0.11	0.10
9: No seperate pedestrian grid	polar	front	to goal	distance	yes	no		0.14	0.15
10: No pedestrians	polar	front	to goal	distance	no	no		0.17	0.17
Straight to goal								0.32	0.29

Table B-1: Different tested variations of the neural network with results on train and test set

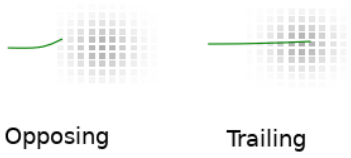


Figure B-1: Pedestrian avoidance trained on pedestrian only data.

B-3 Improve interactions

The aim of the neural network is take the interactions between the robot and the pedestrians into account. In the current setup the amount of interaction is still limited. We trained the neural network on a dataset that consists of only pedestrians, without any obstacles. This way the neural network does not need to develop different behavior for static and dynamic obstacles. Training on pedestrian only data manages to capture an interesting behavior where it comes up with a different trajectory depending on the walking direction of the pedestrian. When a pedestrian in front of the robot is walking in the same direction the robot learned to follow it, while a robot that is approaching the robot is avoided as can be seen in figure B-1. In this instance all the the inputs to the neural network are exactly same, except for the hidden states of the LSTM cells. This shows the memory cells are able to differentiate between these situations and can use the historic paths to create different trajectories. Unfortunately the neural network planners trained on this pedestrian only dataset where not stable when tested in a simulation and where not able to lead the robot to the goal in all cases. This experiment does show that LSTM cells are a promising method to model interactions between pedestrians.

Appendix C

MPC variations

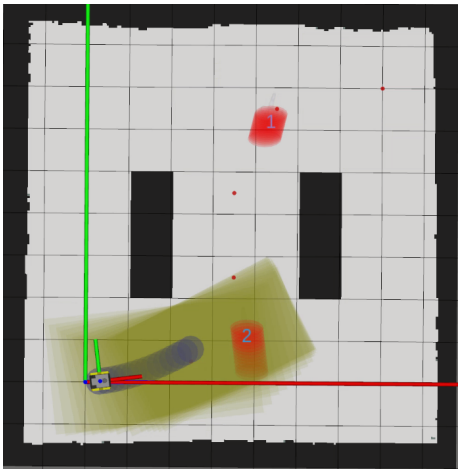


Figure C-1: Simulation scenario two obstacles and two pedestrians

C-1 Variations

In this paper multiple ways of integrating the neural network with the model predictive control are proposed. To analyze which give an improvement to the behavior of the robot, two scenario's are defined in which the different approaches are compared. The first simulation is meant to test the method's capabilities to drive to the goal and do static and dynamic obstacle avoidance in a relatively calm environment. The scenario consists of a 10 by 10 meter room with two static obstacles and two pedestrians. The robot is tasked with navigating to the other corner of the room. The methods are compared on the number times the robot successfully reached to goal against the number of times it collides or gets stuck. We also compare the time it takes the do a successful run, this is an indication of how efficient the route of the robot

was. Each simulation is performed 100 times to account for the randomness in the position of the simulated pedestrians. The list of variations in the parameters can be found in table C-1. Pedestrians can be predicted using the constant velocity assumption or using the neural network prediction. The free space around the robot can be calculated using the expanding rectangles method initialized with the previous MPC result or using the neural network or they can be hardcoded into the optimizer as discussed in the appendix. The initialization can be done using the previous result or using the neural network prediction. The cost function of the optimization consists of a velocity error based on the neural network output or on a fixed reference velocity and the heading guidance can be done based on a position error or a heading error.

C-2 Results

The results of experiments in the first scenario can be found in table C-1. By comparing experiments 1 and 2 it can be seen that pedestrian prediction by neural network or by constant velocity assumption does not make a noticeable difference. A possible reason for this is that in this simulation the pedestrians walk straight a lot anyway, therefor the constant velocity assumption is actually a very good one in scenario. The neural network prediction has showed to be useful in scenario's where there is an obstacle separating the paths of the pedestrian and the robot, this however does not occur often in real world scenario's. The constant velocity assumption will therefor work just fine.

From variations 3 and 4 can be seen that the free space calculation by rectangles based on Neural networks performs better than those based on the MPC result. They even outperform the default which has the free space calculated hard coded in the optimizer. The hard coded method seems to get stuck more often than the free rectangle based method. The neural network based free rectangle calculation is the best.

When initialization is done in the classical way, by using the result of the previous iteration like in experiment 5, the performance of the planner is much worse. The robot gets stuck quite often. This could happen when the previous iteration of the MPC and a new neural network result differ a lot. The optimization will not come to a new trajectory and there for the stuck situation can not be resolved.

Guiding by reference velocity instead of velocity from the neural network as seen in experiment 6 gives a much worse performance. This is especially seen in the case where the robot is at a standstill. With a fixed reference velocity the optimizer has a much harder task to come to a path with a feasible of increased velocity steps, while with the velocity guided by the neural network these velocity steps are already in place.

Guiding by heading error instead of position error as in experiment 8, gives a worse performance. This could be because it is more favorable for the last position of the plan to match up with the neural network than the last heading.

	Ped Pred	Free space calc	Init	Velocity cost	Heading cost		Succesfull	Crashes	Time outs	Time to goal
1: Proposed method	NN	Hardcoded	NN	NN vel	Pos err		92	4	4	18.6
2: Const vel ped pred	Const Vel	Hardcoded	NN	NN vel	Pos err		94	6	0	17.6
3: Free space: MPC	NN	MPC based	NN	NN vel	Pos err		93	7	0	19.1
4: Free space: NN	NN	Neural net	NN	NN vel	Pos err		96	3	1	18.2
5: Initizilization MPC	NN	Hardcoded	MPC	NN vel	Pos err		87	2	11	20.1
6: Reference velocity	NN	Hardcoded	NN	Ref vel	Pos err		53	17	30	20.4
8: Cost by heading	NN	Hardcoded	NN	NN vel	Heading err		85	14	0	19.6

Table C-1: Different tested variations and results experiment 1

Appendix D

Static free space calculation

The optimizer of the MPC is restricted to planning within a calculated free space. This free space has to be calculated very often and therefor has to be computationally efficient. The method relies on having an occupancy map available to calculate the free space from. In the LMPCC paper the free space is calculated based on the locations of the result calculated in the previous iterations of the model predictive controller. At each of the waypoints in the trajectory a region is inflated until it reaches an occupied or unknown cell. He proposes two methods for defining these region: one is a rectangular region and one is a circular region. In the image you can find the rectangular inflated region align with the orientation of the vehicle. As can be seen from the image, not the entire free region around the robot is covered by the free space calculation. This has a direct result that the robot will not be able to use the entire free space. This becomes a limitation in an environment with many static obstacles or in one with tight passage ways.

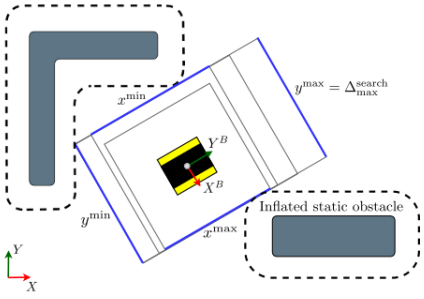


Fig. 7: Vehicle aligned search routine.

Figure D-1: The free space search routine

Another limitation of the current method is that the free space is calculation based on the previous optimal result. Therefor the new optimization can only be done in the proximity of the previous result. This is a major disadvantage in changing environments because the environment may change in such a way that the robot needs to drastically change its plans.

It could be a better plan to go around an obstacle from the other side and the current method of calculated the free space will not allow that. We propose two methods to overcome this and make the planning of the robot more flexible.

The first alternative is to do the free space calculation based not on the previous result of the MPC but on the result of the neural network. This will limit the MPC optimization to only be able to come up with solutions in the vicinity of the neural network generated trajectory, but this does not seem to be a problem because they are suppose to be close together anyway. One condition is that the neural network has to be good enough for this to reach satisfactory results. This could give problems when the neural network plans a trajectory that partially goes through an obstacle, because then there is no free space for that position in the optimization and no feasible trajectory can be calculated.

The second alternative is to program the obstacles directly into the optimizer. This is only possible when the static space is known completely beforehand and can be described with a couple of simple geometric shapes. It is not possible to use a logical OR in the restrictions of a MPC because this would not be differentiable. It is therefor not possible to to limit the x and y coordinates directly. For example if there is an obstacle spanning from x=4 to x=6, one cannot say x should be smaller than 4 or bigger than 6. We need a single equation to describe both possibilities. For this we developed a method to encode a rectangular obstacle in the space. This uses the fact that we can determine if a point is outside of a diamond shaped figure around to origin with a single formula:

$$|x| + |y| >= a \tag{D-1}$$

Where a is the height of the diamond (here chosen to be sqrt(2)/2) such that the sides are 1). the solver is unable to allow the absolute value of an online variable, because the absolute function is not differentiable at the origin. Instead we use this approximation:

$$|x| \ = \sqrt{x^2 + c} \tag{D-2}$$

Where c is a small number The only thing left is to transform the rectangular obstacle and test point such that the obstacle has the same shape as the diamond. This is done by a translation, scaling and a rotation. As can be seen in figure D-2, point A inside to original obstacle and point B is outside. After the transformation point A is still inside and point B outside. The formula for determining if a point is inside or outside of the rectangle will agree with this.

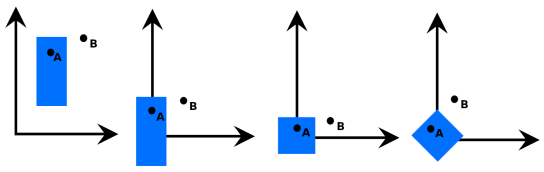


Figure D-2: Transformation from obstacle to diamond shape

In our experiments we noticed that using the neural network to guide the free space search already improves the flexibility but is still a limitation in the trajectory calculation. Since

getting the best method for finding free space is a thesis by it self and not within the scope of this work and because our experimental spaces are easily described by a rectangular external wall and rectangular obstacles we choose to program the static obstacles into the solver directly. In future research or real world implementation a satisfactory solution for this problem has to be found.

Appendix E

Data gathering

We performed a data gathering experiment to create a data set that captures how people walk around in an environment. There are several data sets with pedestrians available online, but they are not satisfactory. There are no data sets available with both static obstacles and pedestrians interacting. Since we want our robot to learn both we need a dataset where the two are combined in one experiment.

E-1 Experiment setup

The goal of this research is to abstract the reasoning of humans when they are walking around in an environment and teach a robot to behave in a similar way. Since we are using a supervised learning approach it is necessary to have good training data. We performed an experiment with four pedestrians walking around and recorded their positions. They way the experiment is setup and the way the people are instructed is vital for their behavior and with that the future behavior of the robot. The subjects in the experiment where explained that they should walk around as they naturally would, this is hard for people if they are very aware that they are being recorded. Therefor we introduced a game that has the sole purpose of making the subjects walk around the room with goal positions. On four sides of the room a small table is setup with a tan gram puzzle. The pieces of the puzzle are spread out over the other tables in the room. The subjects are instructed they can only use pieces from the other tables and they can only walk around with one piece of the puzzle in their hand at the time. This forces the subjects to walk back and forth between the tables often and that the paths of the subjects cross frequently. In the room several static obstacles are placed as well. This will supply us with data on how pedestrians deal with these kind of static obstructions. The location of these obstacles is measured so they can be used in the map later on. The pedestrians are tracked using a motion capturing system, which uses many camera's attached to the ceiling of the room. The pedestrians all where a helmet with several tracking dots. The camera's can distinguish these dots and use the pattern to identify the separate pedestrians.

Figure E-1: Data collection experiment

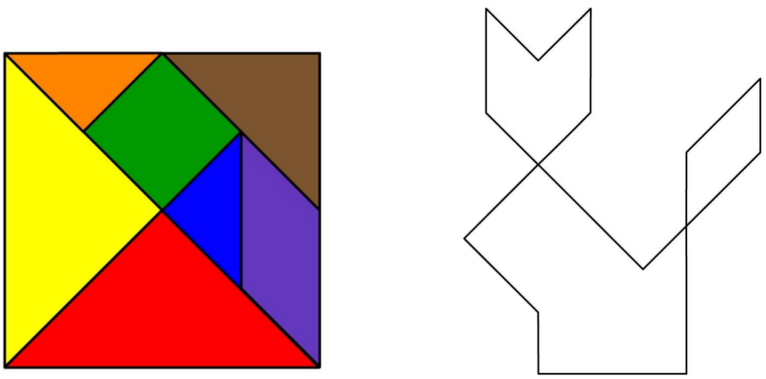


Figure E-2: Tangram puzzle pieces and puzzle

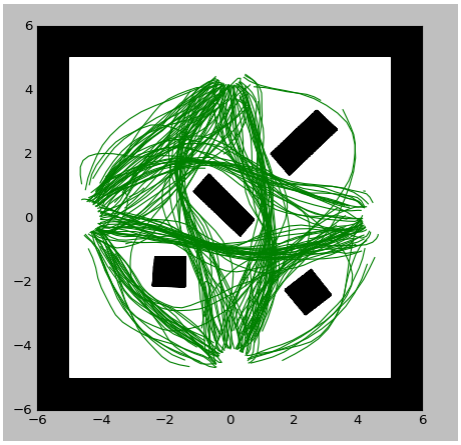


Figure E-3: Filtered trajectories and static obstacles

E-2 Post processing

The recording of the positions of the pedestrians is done at a rate of 100Hz. Our neural network learns data with a time step of 0.3 seconds so the data needs to be sub sampled. For this we use a cubic interpolation that tries to match to a pieces of the trajectory as close as possible, this has the added advantage that it can smooth measurement noise out of the data. On top of that we use a Kalman filter that smoothens the trajectories a bit more. The behavior when standing by the tangram puzzle is not interesting for our algorithm, therefor we remove all the data points where they are within a range of 0.5 meters from one of the tangram puzzles. This leaves with a set of trajectories walking from one table to anther. The goal position is selected as the final point of such a trajectory piece. This results in these trajectories as seen in the image.

E-3 Training on real data

When using the neural network training in the same way as we did on the data set as seen from the simulation, the performance of the model is not great. It learns to predict towards the goal and to take the current heading of the robot into account, but does not learn obstacle avoidance. We expect this is because there is simply not enough data. To support this claim, we take a part of the simulation data set with the same size of the real data set and train on this. The network also does not learn proper obstacle avoidance, while it does learn obstacle avoidance while training on the full simulation dataset.

Gathering much more data is not an option, so we investigate ways to improve the learning with this real data. First we try warm starting the training with the simulation data. This does not yield better results, the model does not end up with better results in either the loss or on the unit tests.

In second attempt to learn more from the gathered data we combined and mixed the real and simulated datasets. For every batch of 16 trajectories that are used in the training every time, 8 will come from the real dataset and the other 8 will come from the simulated dataset.

This very much improves the results when tested against the real dataset. The performance is even much better then when just training on the real dataset for a longer time.

E-4 Discussion on real data

One thing that that was noticed in the experiment is that there where several occasions where a tracking helmet was lost from the motion capturing system. This was all ways when a person was standing bent over their Tangram puzzle. This is not a problem by itself, because the data gathered at these position was filtered out anyway. However the motion capture system had a difficult time identifying the different patterns of the hats. It happens that when a hat is lost from the capturing the systems thinks it sees that hat on the place where another hat actually is. We now have a big jump and two pedestrians on top of each other. To mitigate this problem we cut the trajectories in two when there is a big jump. This results in some short pieces of trajectories where two pedestrians are on top of each other. This could give some confusion when using this dataset later on.

Bibliography

[1] T. Kruse, A. K. Pandey, R. Alami, and A. Kirsch, "Human-aware robot navigation: A survey," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1726–1743, 2013.

[2] M. Kuderer, H. Kretzschmar, and B. Wolfram, "Teaching Mobile Robots to Cooperatively Navigate in Populated Environments," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3138–3143, 2013.

[3] P. Trautman and A. Krause, "Unfreezing the Robot: Navigation in Dense, Interacting Crowds," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 797–803, 2010.