Learning Interaction-Aware Trajectory Predictions for Multi-Robot Motion Planning

F. Martínez Claramunt

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Learning Interaction-Aware Trajectory Predictions for Multi-Robot Motion Planning

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

F. Martínez Claramunt

September 25, 2020

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

Multi-robot motion planning without a central coordinator usually relies on the sharing of planned trajectories among the robots via wireless communication in order to achieve predictive collision avoidance. Path planners found in the literature that feature this scheme usually boast levels of performance comparable with their centralized counterparts. However, in practice, communication tends to be unreliable and lead to significant performance degradation, which may eventually result in collisions among robots.

This thesis proposes a data-driven decentralized approach for multi-robot motion planning in dynamic environments. Instead of requiring robots to share their intentions with each other, a model based on recurrent neural networks (RNNs) is used to predict them. This model is trained on data obtained with a centralized sequential model predictive control (MPC)-based motion planner in which intended trajectories of neighboring robots are available while planning. The learned model can be efficiently run online and provide accurate predictions for each robot in the environment based on a horizon of past observations of all robots' states. This model is then incorporated into the MPC planning framework so that it may be run in a decentralized manner. It is finally shown in simulations that the proposed approach achieves a similar level of performance to its centralized counterpart.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1-1 Motivation

Historically, the use of robots has been mainly restricted to industrial areas under controlled conditions due to their high cost and associated safety hazard. Navigating in unknown, dynamic environments entails different challenges to those found in industrial settings, problems which were accentuated in the past by limited onboard computing power and battery technology. However, advancements in electronics and other engineering disciplines such as control theory, motion planning and artificial intelligence have opened new possibilities for commercial applications of mobile robots that were previously not possible. Well-known examples of these are autonomous cars, delivery drones and surveillance robots.

These applications often require the robot to move among other decision-making agents, such as humans or additional robots. This circumstance makes the collision avoidance problem much more complicated than navigating in static environments, since in addition to avoiding obstacles, agents must also avoid colliding with each other. In the simplest case where all agents are controlled robots and they are in a known environment, it may be possible to implement a centralized approach and let a central coordinator find the best paths for all of them. However, such approaches are usually not implemented in practice because it is more computationally efficient for large teams of robots to distribute the solution of the path planning problem among their onboard computers and share relevant variables via communication. This approach is also more fault tolerant, as it removes the unique point of failure in the form of the central coordinator. Nevertheless, situations may arise where communication with an agent is lost, compromising the system and causing a significant degradation in performance or even a complete failure. There are also applications for which communication is simply not possible because one or several agents are not under our control, as is the case when navigating among humans. All of these circumstances motivate the development of fully decentralized approaches, which rely on local sensor data to plan collision-free trajectories instead of using communication.

The goal for this thesis is to develop a data-driven decentralized approach for multi-robot motion planning in dynamic three-dimensional environments. Instead of requiring robots to

share their intentions with each other to achieve efficient collision avoidance, a model based on recurrent neural networks (RNNs) will be used to predict them. This model will be trained on data obtained with a centralized sequential model predictive control (MPC)-based motion planner in which intended trajectories of neighboring robots are available while planning. This model will then be incorporated into the MPC planning framework so that a decentralized solution may be implemented.

1-2 Contributions

The main contributions of my work are:

- A RNN-based trajectory prediction model that works in multi-robot three-dimensional environments with an arbitrary number of robots and dynamic obstacles. It relies on three sources of information which can be obtained from sensory data: a past horizon of velocities of the robot we are trying to predict a trajectory for, the set of observed past relative states of the other robots in the scene with respect to the one of interest over a horizon, and the set of current relative states of the dynamic obstacles in the scene.
- A communication-free decentralized MPC-based multi-robot motion planner that works in dynamic 3D environments and considers robots other than the controlled one as dynamic obstacles with positions forecast by the previously trained prediction model, and assumes regular dynamic obstacles to follow a constant velocity model (CVM). This method achieves a collision-avoidance performance comparable to the centralized sequential version of the path planner in which robots know the intentions of each other while planning for themselves.

1-3 Thesis Outline

The main part of this thesis will start with Chapter 2, where I will contextualize my work within the literature by presenting some relevant multi-robot motion planning and trajectory prediction approaches. In Chapter 3, the theoretical foundations on which the proposed method builds on are explained. In Chapter 4, my method will be exposed in detail, from the trajectory prediction part to the path planning itself. In Chapter 5, I will put the proposed approach into practice on some simulated scenarios and will separately discuss the achieved results of the prediction model and of the complete path planner, comparing them to some baselines. Finally, in Chapter 6, I will outline some conclusions that may be reached from my work, and will present some directions for further research. This thesis has also lead to the writing of a research paper which we intend to submit to the IEEE Robotics and Automation Letters (RA-L) with the option of a presentation at 2021 IEEE International Conference on Robotics and Automation (ICRA). An initial draft of the article is included in Appendix A.

Chapter 2

Related Work

2-1 Multi-Robot Collision Avoidance

Multi-robot collision avoidance is a mature research topic that is still very actively researched. Traditional single robot rule-based collision avoidance methods such as the dynamic window approach [1] and artificial potential fields [2, 3] have been adapted in the past to work in the multi-robot case. These can be implemented in a decentralized manner, but they are extremely short-sighted due to their reactive nature and may lead to deadlocks. Velocity obstacles (VOs) were also extended to work for decentralized multi-robot systems in [4] under the name of reciprocal velocity obstacles (RVOs). Further improvements to this framework led to the well known optimal reciprocal collision avoidance (ORCA) algorithm from [5], and its extension to non-holonomic robots in [6]. However, these methods ignore the robots' dynamics and only plan for one step ahead into the future.

There have also been a number of receding horizon optimization approaches that have been tried in the literature, which can be classified in centralized [7–10], distributed [10–12] and decentralized (communication-free) [10, 13]. Centralized methods do not scale well to bigger teams of robots and the requirement of a central planner makes them unsuitable for most remote outdoor environments. As it has been already introduced, distributed methods often do not work as well in practice as they do in theory because of the tendency of wireless communication to be unreliable. Regarding the cited communication-free decentralized model predictive control (MPC)-based methods, they use a constant velocity model (CVM) to predict the trajectories of robots other than the controlled one, which leads to a significant performance degradation with respect to their centralized MPC counterparts due to the poor prediction accuracy of the model. In this work I close this performance gap between the centralized and decentralized approaches by developing a much more accurate recurrent neural network (RNN)-based prediction model which is trained on simulations using a centralized sequential MPC.

There are also many learning-based techniques that have been used to achieve decentralized multi-robot collision avoidance, such as interacting Gaussian processes [14–16] and reinforcement

learning [17–20]. However, they have only considered environments of up to two dimensions and they do not provide any collision avoidance guarantees. [21] presents a reinforcement learning-based method that works in three dimensions, but they only consider single robot static environments, and the trained policy does not guarantee collision avoidance. My method, on the other hand, offers a collision avoidance guarantee over a finite future horizon through the use of an MPC algorithm for planning.

2-2 Motion Prediction

The topic of motion prediction has raised significant interest in recent years due to the need to account for human future intentions to achieve safe robot navigation in urban environments. One of the first motion prediction approaches that was introduced in the literature is the Social Force model [22]. This rule-based method models pedestrian behavior through the use of attractive and repulsive potentials. Even though it is commonly used as a baseline due to its simplicity, it requires manual parameter-tuning and its accuracy is far from that of the current state-of-the-art.

There have also been several notable attempts to use game theory to predict behavior of decision-making agents [23–25]. A more sophisticated approach is presented in [26], where game theory is combined with a tool from social psychology called social value orientation (SVO) in order to quantify agents' degree of selfishness or altruism before predicting their behavior. However, these methods restrict themselves to very specific scenarios and to the best of my knowledge have only been applied to two dimensional spaces.

The class of approaches that have produced the best results so far are based on machine learning. Some of these include: dynamic Bayesian networks [27], random forests [28], inverse reinforcement learning [29–31], RNNs [32, 33], variational autoencoders [34] and generative adversarial networks (GANs) [35, 36]. To the best of my knowledge, no prior paper has explored the topic of trajectory prediction with robots. As a consequence, none of these methods has been developed for 3D environments, which is necessary when dealing with aerial robots. Out of the cited methods, those which account for obstacles do so only for static ones through the use of 2D grid maps of the environment, which is processed by a convolutional neural network (CNN), an approach which cannot be straightforwardly extended to 3D. Another main difference between the pedestrian and robot prediction problems is that pedestrians tend to follow quasi-constant speed trajectories, except when they interact with other elements of the road, in which case they usually start and stop suddenly. Robots, on the other hand, tend to show smoother changes in velocity when they reach their goals or start to move towards new ones. Their behavior variability is also much smaller than with humans, since (usually) they all follow exactly the same policy and have theoretically equal dynamics. For these reasons, I develop a new RNN-based architecture that can better leverage the robots' dynamics and which can deal with an arbitrary number of other robots and static or dynamic 3D obstacles.

Chapter 3

Preliminaries

In this chapter the theoretical background necessary to understand the proposed motion planning approach is presented. I will start by presenting the general multi-robot motion planning problem in Section 3-1. Then I will formulate it as a receding horizon optimization problem in Section 3-2. Finally, I will overview the usage of recurrent neural networks (RNNs) for the sequence-to-sequence learning problem in Section 3-3.

3-1 Multi-Robot Motion Planning

Multi-robot motion planning deals with the problem of finding how a set of robots may reach some target positions from their starting ones while avoiding collisions with each other and with any potential external obstacles in the way. Throughout this thesis I assume that robots have assigned goals, as the problem of task assignment is out of my scope.

Before analyzing the multi-robot problem in Section 3-1-2, I outline the simpler single robot problem in Section 3-1-1. This is interesting not only because it lays the foundations to better understand the complete problem, but also because it presents the challenges of decentralized multi-robot motion planning, which are the ones we are concerned with.

3-1-1 Single robot problem

Formally, the purpose of a path planning algorithm is to find a continuous path $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$, such that $\tau(0) = q_{init}$ and $\tau(1) = q_{goal}$, where $\mathcal{C}_{free}$ is the free configuration space (often abbreviated as C-space), q_{init} is the initial configuration and q_{goal} is the goal one (see [37] for further details). Additionally, it is usually desirable to do so in an optimal way based on a number of requirements that depend on each specific application. We may want the robot to reach its goal as quickly as possible, or traveling the shortest distance, or spending the least amount of energy, or any combination of these.

Given this problem formulation, a number of algorithms have been developed in the literature in order to find a solution. Classical algorithms can be divided in two different types of

approaches: sampling-based and combinatorial. The first of them relies on sampling different configurations from C-space and checking whether they are collision-free or not. The problem is then reduced to connecting the initial and goal configurations through these sampled ones. The main advantage of this approach is that it does not require the explicit modeling of the obstacle regions in C-space, although it comes at the cost of these algorithms being only *probabilistically* complete[1]. Combinatorial methods, on the other hand, use explicit representations of the environment in order to construct roadmaps of free space. They then query these to find a path from an initial to a target configuration using a graph search algorithm. These methods have the advantages that they are complete, and are also much more efficient than sampling-based ones when the problem has certain features, such as low dimensionality of the space and obstacle convexity. However, when these properties are not present they might not be suitable.

One limitation of the presented formulation is that it assumes that obstacles are static. This means that collision avoidance is not guaranteed if the environment changes from the instant when the planning was performed, as a changing environment implies a time-varying $\mathcal{C}_{free}$. Thus, planning in C-space is insufficient in dynamic environments. In this setting we need to plan in the state space $X = \mathcal{C} \times T$, where $\mathcal{C}$ is the configuration space and $T \subseteq \mathbb{R}_{\geq 0}$ is the time interval. The time interval may be bounded ($T = [0, t_f]$), if the goal position must be reached before a certain time limit $t_f > 0$, or unbounded if no such restriction exists ($T = [0, \infty)$). One of the main differences when planning in X is that paths need to follow the additional constraint that they can only move forward in time. Formally, we want to compute a path $\tau : [0, 1] \rightarrow X_{free}$, such that $\tau(0) = x_{init}$ and $\tau(1) = X_{goal}$, where X_{free} is the set of collision-free states, x_{init} is the initial state, X_{goal} is the set of goal states such that $X_{goal} = \{(q_{goal}, t) \in X_{free} \mid t \in T\}$. The monotonic constraint of time can be expressed as the condition that for any $s_1, s_2 \in [0, 1]$ such that $s_1 < s_2$, with $(q_1, t_1) = \tau(s_1)$ and $(q_2, t_2) = \tau(s_2)$, it must hold that $t_1 < t_2$ (the reader is once again referred to [37] for further details).

Although planning under this new formulation is more challenging than in the case where the environment is static, sampling-based algorithms can usually be adapted from $\mathcal{C}$ to X. Adapting combinatorial methods is also possible and they may be used in some cases, but it is more difficult due to their poor scaling capabilities. In both cases it will be necessary to take into account the monotonicity of the time dimension, which will result in roadmap graphs becoming directed. Instead of planning in X, another option is to decouple the time component from the robot configuration one. This involves first planning a path in $\mathcal{C}$ and then adjusting the pace at which it will be followed so that moving obstacles are avoided. This simplifies the problem, but collision avoidance cannot be guaranteed in general.

Another challenge of planning in X is that it requires the knowledge of the moving obstacles' positions at every future time instant in order to be able to plan around them. A common approach to solve this problem is to assume that obstacles will maintain a constant speed, which may work well in practice when accelerations are low and planning is performed on real time at a sufficiently high rate. However, this approach is unsuitable when the dynamic obstacles the robot is trying to avoid are other robots which are also trying to avoid colliding

[1] A path planning algorithm is said to be complete if, in the case that a path to the goal exists, the algorithm will always find it. Such an algorithm will also be able to discover if a path does not exist. A probabilistically complete planner has a probability of finding a solution which tends to one as time tends to infinity.

with the ego robot itself. Thus, in the next section I will present a more formal approach to deal with this scenario.

3-1-2 Multi-robot problem

We have studied the single-robot problem in static environments, and we have seen how adding the time dimension to the state space enables collision-free motion planning in dynamic environments. Following a similar reasoning for the multi-robot problem, we may consider a state space that consists of the individual configurations for all the robots. This state space can be expressed as $X = \mathcal{C}^1 \times \mathcal{C}^2 \times \cdots \times \mathcal{C}^m$, which has a dimension $\dim(X) = \sum_{i=1}^{m} \dim(\mathcal{C}^i)$, where m is the number of robots. Under this setting, collisions of two types must be avoided: with external obstacles and among robots. Formally, the obstacle region in X can be defined as:

$$X_{obs} = \left(\bigcup_{i=1}^{m} X_{obs}^{i} \right) \bigcup \left(\bigcup_{ij,\, i \neq j}^{m} X_{obs}^{ij} \right),$$

where X_{obs}^{i} and X_{obs}^{ij} are the robot-obstacle and robot-robot collision states, respectively [37]. As in the single robot case, the goal of a path planning algorithm will be to find a continuous path $\tau : [0, 1] \rightarrow X_{free}$ such that $\tau(0) = x_{init}$ and $\tau(1) = x_{goal}$, with $X_{free} = X \setminus X_{obs}$.

Even though planning in this newly defined joint state space might seem to be a straightforward extension from planning in X for a single robot, in reality it is not so. The main issue is that the space dimension of X scales linearly with the number of robots, making the problem intractable for any decently sized team of robots. This is especially problematic for combinatorial methods, which are only efficient for low-dimensional spaces, as previously stated. Even though sampling-based methods scale better, they are still unlikely to be suitable when there are many robots present. This prompts us to look for alternative approaches which can deal with these types of settings.

A common solution is to perform prioritized planning, which in its simplest form involves sequentially solving m separate single robot motion planning problems, where each robot considers the ones with higher priority as dynamic obstacles to be avoided and ignores the lower ones. Another popular option is to find collision free paths for each robot independently, completely ignoring the others in the scene, and then schedule the motions in order to avoid collisions. Even though these options have been shown to work quite well in practice, the results will rarely be the most efficient given that there is little to no cooperation among the robots.

Another issue that appears when planning for multiple robots is the choice of who will perform the planning. All of the presented solutions assume that full information about the system is available, which is the case when a central computer is tasked with performing the planning for all the robots. Although this is the most straightforward solution, using one unique computer to perform the planning introduces a single point of failure and also scales poorly with the number of robots in the team. An option which is much more fault tolerant and scales better is the use of a distributed algorithm, where each robot plans for itself and communicates with the others in order to achieve joint collision avoidance. If communication among the robots were not possible, there exists the third option of relying only on sensory information in order for each robot to perform path planning by itself. This solution also scales well with team

size, but the lack of communication limits maximum achievable performance and requires the use of alternative algorithms to efficiently avoid collisions with other robots, which may increase the total computational load.

3-2 Model Predictive Control

3-2-1 Introduction

Model predictive control (MPC) is an optimization-based control strategy which relies on an explicit model of the plant in order to predict how it is going to behave over a finite future time horizon. The cost function is defined so as to reflect the control goals and specifications, and the resulting optimization problem is solved at each time step in a receding horizon fashion, with only the optimal action for the next time step being applied (see [38–40] for further details). This powerful control algorithm is capable of dealing with nonlinear dynamics, multivariable systems, future references, and input, output and state constraints. With the development of increasingly faster hardware over the past decade, the aforementioned features have made MPC a popular choice for the control of complex dynamical systems.

Formally, the MPC formulation may be expressed as:

$$
\begin{aligned}
\min_{\mathbf{u}_N} \quad & V(x^0, \mathbf{u}^N) := \sum_{k=0}^{N-1} \{\underbrace{J^k(x^k, u^k)}_{\text{stage cost}}\} + \underbrace{J^N(x^N)}_{\text{terminal cost}} && \text{(cost function)} \\
\text{s.t.} \quad & x^{k+1} = f(x^k, u^k) \quad \forall k \in \{0, 1, ..., N-1\}, && \text{(system dynamics)} \\
& u^k \in \mathbb{U}, && \text{(input constraints)} \\
& x^k \in \mathbb{X}, && \text{(state constraints)} \\
& x^N \in \mathbb{X}_f, && \text{(terminal constraint)}
\end{aligned}
$$

where V is the cost function, J^k is the stage cost, J^N is the terminal cost, N is the horizon length, and x^k and u^k are the state and input vectors at instant k, respectively. x^0 is the state at the current time instant, $\mathbf{u}^N := (u^0, u^1, ..., u^{N-1})$ is the input sequence, and f is a possibly nonlinear function that describes the discrete dynamics of the plant. The system constraints are expressed in a set theoretical-manner, with $\mathbb{U}$ and $\mathbb{X}$ representing the input and state constraints, respectively. Additionally, the state at the final prediction step may be further constrained by a terminal set $\mathbb{X}_f \subseteq \mathbb{X}$ to which the system needs to be driven.

3-2-2 Multi-robot motion planning using MPC

An advantageous feature of MPC is that casting the control problem as a constrained optimization provides it with great flexibility, in addition to making the tuning process relatively intuitive. This, combined with its ability to deal with complex, nonlinear dynamics make it an attractive option to solve the problem of robot motion planning. Although many different specific implementations have appeared in the literature [7–13], they all involve representing the robot's objective in the cost function (e.g. to minimize traveled distance to the goal and

required control effort), and defining obstacles as constraints on the states or as additional terms in the cost function.

An important difference of MPC with respect to the methods discussed in Section 3-1 is that it is a local method that is meant to be run recursively at each time step. This makes it one of the preferred options when dealing with dynamic environments, as it does not require perfect prior knowledge of the obstacles' trajectories, as adjustments can be made over time. However, this also means that the resulting start to goal path may not be globally optimal, as it only takes into account a finite future horizon at each time step. In practice, one may alleviate this issue by using MPC in combination with a global planner. The latter will consider a simplified model of the robot and the environment to generate a global path, which the former will track while dealing with obstacles that were not initially taken into account. It must be noted that in addition to being a local motion planner, MPC may be additionally used to stabilize robots with unstable dynamics such as quadrotors, as it still is a standard control technique.

Another interesting feature of this method is that the extension to multi-robot systems is relatively straightforward. In its most basic form, it simply requires the redefinition of the cost function based on the joint objectives of the robots and the addition of the necessary constraints to avoid robot-robot collisions. The issue with taking such an approach is that the time to find a solution will grow significantly for every robot that we add to the system, severely restricting the maximum team size that can be considered. However, the decoupled nature of multi-robot systems' dynamics makes it possible to take smarter approaches which scale much better with the number of robots. In the following two sections we will look into some of these solutions.

Centralized sequential planning

As it has been stated, solving the full MPC problem for all robots at the same time in a centralized manner becomes very computationally expensive for large teams of robots. A different approach is to formulate an optimization problem for each one of the robots and solve them sequentially, considering the other robots as dynamic obstacles to avoid. The future trajectories that will be considered for the other robots are the predicted state trajectories from each of the individual MPC problems. This strategy is similar to the prioritized approach presented in Section 3-1-2 for classical motion planning methods, but in this case there is no notion of priority. This is possible because the MPC algorithm is run at each sampling step, whereas the previously presented global methods were intended to be run once and then make the robots follow the full paths. Thanks to this difference, at each time step the optimal future state trajectories computed at the previous step will be available. Thus, it will be possible to use these as predictions for the robots that are later in the sequence when planning for the earlier ones. This approximation works well when models are sufficiently accurate and the sampling rate is high enough, as in this case the predicted trajectories should not significantly change from one step to the next. Although there will be a slight performance loss when compared to the solving the MPC problem for all robots at once, this approach scales much better with the number of robots.

Distributed and decentralized planning

Another nice property of the previous approach is that it can be easily distributed by letting each robot to solve its own MPC problem and then communicate its results with the others, so that these may be used at the next time step as trajectory predictions. With fast enough communication, the performance of this method should (theoretically) be almost the same as the one from the centralized sequential approach. The main difference is that since all robots will be planning at the same time, they will all need to use others' predicted trajectories from the previous sampling step.

The next natural extension would involve decentralizing the path planning algorithm. However, this requires an alternative way to predict the future trajectories of other robots, as optimal planned trajectories cannot be shared via communication. Obtaining an analytical prediction model is not a viable option, since the future trajectory of each robot depends on the solution to its own MPC problem and therefore it cannot be generally expressed in closed form. Thus, one could assume other robots to keep their velocity at the last step, an approximation which is frequently made in the literature when no better model is available. However, even though this may still make it possible for robots to reach their goal positions while avoiding others, the overall performance will be greatly degraded. A different approach to construct a model when it cannot be done so analytically is to use machine learning techniques. In the next section I will present the necessary background to do this, as it will be the approach that will be taken for this thesis.

3-3 Recurrent Neural Networks

3-3-1 Introduction

The field of machine learning studies algorithms that enable a computer to learn to perform tasks without being explicitly programmed to do them, by instead extracting hidden patterns in data. Deep learning covers the subset of these algorithms that use artificial neural networks as the function approximators to capture these hidden patterns (see [41] for more in depth explanations). With a structure inspired by the human brain, artificial neural networks consist of multiple nodes (or neurons) which are connected by synapses, as can be seen in the example of Figure 3-1. Neurons are grouped by layers, and the outputs of each layer (also known as activations) are computed exclusively forwards. These activations are calculated as a nonlinear function of the neuron's inputs, which are a weighted sum of the outputs from the previous layer plus a bias (see Figure 3-2). The weights and biases of the network are the optimization variables of the learning algorithm, which will try to minimize some measure of how well the network can correctly predict a set of outputs based on a set of inputs.

3-3-2 Recurrent neural networks

RNNs are a type of neural network which is capable of processing sequential data. They are able to learn temporal dependencies in data, and are often used for time series forecasting or natural language processing. Recurrent neurons keep an internal state which evolves based

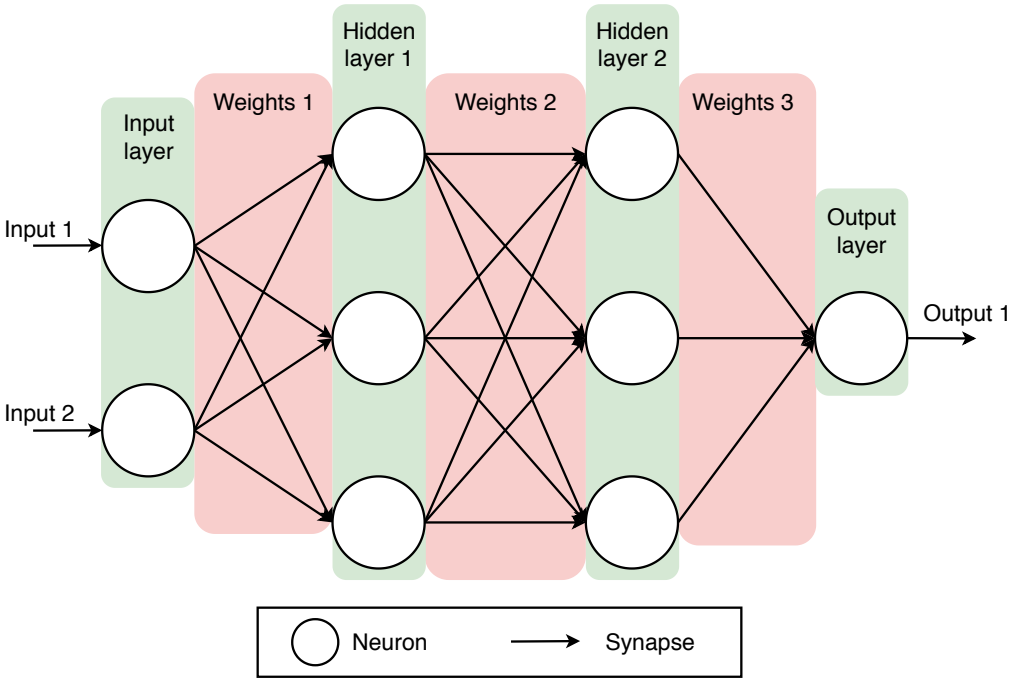


Figure 3-1: Diagram of a simple feedforward network with two inputs, one output, and 2 hidden layers with 3 neurons each.

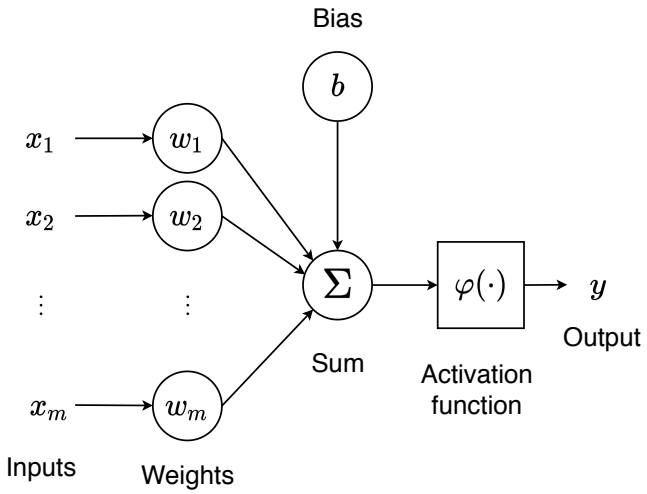


Figure 3-2: Diagram of an artificial neuron.

on its previous value and the value of the input, as can be seen in Figure 3-3. Thanks to this, they can process sequences of different lengths.

However, traditional RNNs have issues learning long-term temporal dependencies, since the gradient of the loss function decays exponentially with time (vanishing gradient problem [42]). This means that it is hard for them to learn how inputs early in the sequence affect the output that needs to be predicted. To mitigate this drawback, a type of RNN called long short-term memory (LSTM) was developed [43,44], introducing a memory cell and a set of gates. The cell

is able to keep information over prolonged periods of time and learn long-term dependencies. The gates control the flow of information in and out of the cell, and when information needs to be forgotten.

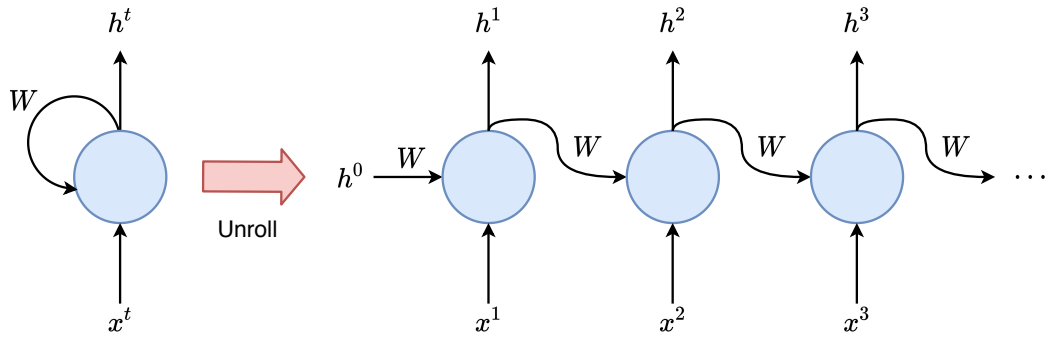


Figure 3-3: Basic diagram of a traditional recurrent neuron, where W is the weight matrix, and x_t and h_t are the values of the input and activation of the neuron at step t, respectively.

3-3-3 Sequence-to-sequence learning

For this thesis I am not only interested in processing sequential inputs, but also in producing sequential outputs, since a trajectory prediction is essentially a sequence of states. These types of problems are known as sequence-to-sequence learning, and RNNs are a suitable architecture to solve them. This is the case because they are not only able to process sequential data, but also to output it by keeping a record of its internal state at each time step.

A popular architecture that is used for this sequence-to-sequence learning problem is the so-called encoder-decoder. As it is shown in Figure 3-4, this architecture uses an RNN to encode the input sequence into a flat vector z, from which the future data is predicted using another RNN. The way in which a flat vector may be fed into an RNN is by repeating the vector along the sequence dimension. Formally, this means that $x^t = z \quad \forall t \in \{1, 2, 3, ..., N_{pred}\}$, where $N_{pred} = T_{pred} - T_{obs}$ is the prediction horizon. Building on these concepts, in this thesis I will design a network architecture that is capable of predicting robots' future positions in dynamic environments, as the key component to my decentralized motion planning approach.

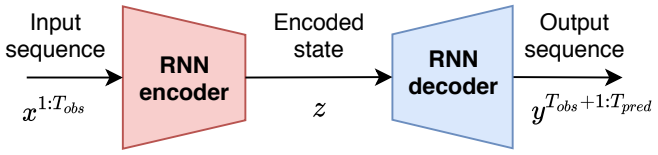


Figure 3-4: Diagram of the encoder-decoder network architecture for future sequence prediction.

Chapter 4

Approach

4-1 Introduction

In this chapter, the proposed interaction-aware multi-robot motion planning approach is described in detail. It is designed to work for sizable teams of robots navigating in either 2D or 3D dynamic environments without communication among them. To provide each of them with interaction-awareness about the others, a recurrent neural network (RNN)-based prediction model is developed, such that it will be able to forecast how any robot in the scene will move in the near future based on its observed past behavior and that of the neighboring robots. It also takes into account moving obstacles in the scene, which are restricted to have a fixed size, since in practice they mainly represent humans walking among the robots and this greatly simplifies the prediction challenge. These obstacles are assumed to move at constant speed over the considered future horizon.

To perform decentralized motion planning, I use a model predictive control (MPC) algorithm that considers robots other than the one we are planning for as dynamic obstacles, the positions of which are forecast using the trained RNN-based model. This framework allows us to easily generate data to train the neural network by running simulations with a centralized sequential version of the planner. As previously described in Section 3-2-2, in this formulation the MPC problems for each of the robots will be solved sequentially by a central computer, using the latest available planned trajectories for a robot as its predicted positions for the planning horizon.

The outline of this chapter is the following. In Section 4-2, rigorous formulations of both the path planning and the trajectory prediction problems are introduced. In Section 4-3, a centralized sequential MPC-based solution to the path planning problem is implemented in order to generate suitable data for training and validating a trajectory prediction model. The proposed RNN-based model is then presented in Section 4-4, in addition to the used evaluation metrics and details of the training procedure. Finally, in Section 4-5, the proposed decentralized interaction-aware multi-robot motion planning solution is described, building on the information introduced in the previous sections.

4-2 Problem Formulation

4-2-1 Multi-robot motion planning

The multi-robot motion planning framework that is employed in this thesis is based on that of [10]. A team of n robots moving in a shared workspace $\mathcal{W} \subseteq \mathbb{R}^3$ is considered, where each robot $i \in \mathcal{I} = \{1, 2, ..., n\} \subset \mathbb{N}$ is modeled by their enclosing spheres of radius r. Robots are represented by these primitive objects because checking collisions between arbitrarily shaped 3D objects is computationally expensive. Furthermore, I consider n_o (moving) obstacles in the scene, each of these $o \in \mathcal{I}_o = \{1, 2, ..., n_o\} \subset \mathbb{N}$ being approximated by a non-rotating enclosing ellipsoid with principal axes of length a, b and c, common to all of them. The reason why I assume the orientation and size of all obstacles to be the same is because in practice these will mainly represent humans walking among the robots. Thus, it is reasonable to assume that they will always be upright and of a similar volume, and it will furthermore simplify the task of designing the trajectory predictor later.

The dynamics of each robot $i \in \mathcal{I}$ are described by a discrete-time equation

$$x_i^{k+1} = f(x_i^k, u_i^k), \tag{4-1}$$

where $x_i^k = [p_i^k, v_i^k, \phi_i^k, \theta_i^k, \psi_i^k] \in \mathbb{X} \subset \mathbb{R}^{n_x}$ denotes the state of the robot, which includes its position, velocity, pitch, yaw and roll, and $u_i^k \in \mathbb{U} \subset \mathbb{R}^{n_u}$ the control input, both at time step k. $\mathbb{X}$ and $\mathbb{U}$ are the admissible state space and control space, respectively. All the robots are assumed to have the same dynamics, described in discrete time by function f.

On the other hand, each of the obstacles $o \in \mathcal{I}_o$ is assumed to follow a constant speed model, such that

$$x_o^{k+1} := \begin{bmatrix} p_o^{k+1} \\ v_o^{k+1} \end{bmatrix} = \begin{bmatrix} I & \Delta t \cdot I \\ 0 & I \end{bmatrix} \begin{bmatrix} p_o^k \\ v_o^k \end{bmatrix}, \tag{4-2}$$

where $x_o^k = [p_o^k, v_o^k] \in \mathbb{X}_o \subset \mathbb{R}^{n_o}$ denotes the state of obstacle o at time k, which includes its position and velocity. I is the identity matrix and Δt is the sampling period.

Any pair of states for robots i and j from the team are considered to be mutually collision-free at time k if $\|p_i^k - p_j^k\| \geq 2r, \forall i \neq j \in \mathcal{I}$. Unfortunately, checking collisions for robots and obstacles is not as simple, as determining whether a sphere and an ellipsoid intersect each other cannot be performed with a closed form expression [45]. Thus, to check collisions between each obstacle o and each robot i at time k, the former will be approximated with an ellipsoid enlarged by the radius of the latter, and then it will be verified if the robot's position is within it. This condition can be formally expressed as $\|p_i^k - p_o^k\|_\Omega \geq 1, \forall i \in \mathcal{I}, \forall o \in \mathcal{I}_o$, with $\Omega = \text{diag}(1/(a+r)^2, 1/(b+r)^2, 1/(c+r)^2)$.

Each robot is assumed to have an assigned goal location g_i, which in practice may come from a high-level planner or have been specified by a user. The objective of the multi-robot motion planner will therefore be to compute a control action u_i^k for each robot in the team which brings the robots closer to their respective goal locations g_i while respecting the dynamic constraints and ensuring that they are collision-free for a future short time horizon τ.

By writing these objectives as an optimization problem, the following MPC formulation is

obtained:

$$\begin{aligned}
\min_{x_i^{1:N}, u_i^{0:N-1}} \quad & \sum_{k=0}^{N-1} J_i^k(x_i^k, u_i^k) + J_i^N(x_i^N, g_i) \\
\text{s.t.} \quad & x_i^0 = x_i(0), \\
& x_i^k = f(x_i^{k-1}, u_i^{k-1}), \\
& \left\| p_i^k - p_j^k \right\| \geq 2r, \\
& \left\| p_i^k - p_o^k \right\|_\Omega \geq 1, \\
& u_i^{k-1} \in \mathbb{U}, \quad x_i^k \in \mathbb{X}, \\
& \forall j \neq i \in \mathcal{I};\ \forall o \in \mathcal{I}_o;\ \forall k \in \{1, \dots, N\},
\end{aligned} \tag{4-3}$$

where J_i^k is the stage cost for robot i at time k, J_i^N is the terminal cost for robot i, and N denotes the number of steps of the MPC future horizon, such that $\tau = N\Delta t$. The stage cost will consist of the sum of two terms:

$$J_i^k(x_i^k, u_i^k) \coloneqq \left\| u_i^k \right\|_{Q_u} + \sum_{j \in \{\mathcal{I} \setminus i\} \bigcup \mathcal{I}_o} \frac{Q_c}{1 + \exp(\lambda_c (d_{ij}^k - d_{ij}^{\text{safe}}))}, \tag{4-4}$$

where the first one penalizes large control actions through a weighted norm of the inputs, and the second one is a collision potential cost based on the logistic function, such that robots are incentivized to stay away from each other and from obstacles. In this expression, Q_u is the weighting matrix of the control input cost, Q_c is the weighting coefficient of the collision potential cost, λ_c is a parameter controlling the smoothness of the potential, d_{ij}^k is the distance between i and j at time k and d_{ij}^{safe} is a safety distance. It must be noted that the distance metric varies depending on whether it is between two robots or between a robot and an obstacle, such that:

$$d_{ij}^k \coloneqq \begin{cases} \left\| p_i^k - p_j^k \right\|, & \text{if } j \in \{\mathcal{I} \setminus i\} \\ \left\| p_i^k - p_j^k \right\|_\Omega, & \text{if } j \in \mathcal{I}_o \end{cases} \tag{4-5}$$

The terminal cost consists of a goal navigation term, such that the robots are driven towards their goals, and its expression is:

$$J_i^N = (x_i^N, g_i) \coloneqq \left\| p_i^N - g_i \right\|_{Q_g}, \tag{4-6}$$

where g_i is the goal of robot i and Q_g is the weighting matrix of the goal navigation cost.

Under this formulation, motion planning involves solving the constrained optimization problem in Equation (4-3) for each robot $i \in \mathcal{I}$ at each step and executing the first control action from the resulting optimal sequence $u_i^{0:N-1}$, in a receding-horizon fashion.

4-2-2 Interaction-aware trajectory prediction

The trajectory prediction problem that is solved in this thesis involves estimating the future positions of robots other than the controlled one for the future N_{pred} prediction steps. I refer

to these other robots as "query" robots, as is common in the trajectory prediction literature. Even though my scope is limited to performing predictions for robots following the control strategy that was described in the previous section, the presented formulation is highly general and could be used for other trajectory prediction problems. I denote the predicted trajectory for a query robot q as

$$p_{q,\text{pred}}^{t+1:t+N_{\text{pred}}} \coloneqq \left(p_{q,\text{pred}}^{t+1}, p_{q,\text{pred}}^{t+2}, ..., p_{q,\text{pred}}^{t+N_{\text{pred}}}\right), \tag{4-7}$$

with the ground truth being denoted as $p_{q,\text{target}}^{t+1:t+N_{\text{pred}}}$ and defined analogously. It must be noted that since I aim to use the predictions within the MPC, N_{pred} needs to be equal or greater than the MPC planning horizon N. For simplicity, I will consider them to be equal.

Even though the final goal is to obtain a prediction $p_{q,\text{pred}}^{t+1:t+N_{\text{pred}}}$ that is as similar as possible to $p_{q,\text{target}}^{t+1:t+N_{\text{pred}}}$, in practice I will aim to predict to velocities instead. The reason for this comes from the fact that I will be using deep learning to develop a prediction model, and employing absolute positions in this framework may inadvertently lead to training a neural network that outputs different predictions for different predictions in the environment. This is clearly not coherent with how the MPC-based planner from Section 4-2-1 works, and therefore using this formulation can lead to overfitting and poor generalization of the model. Thus, I aim to predict a horizon of velocities $v_{q,\text{pred}}^{t+1:t+N_{\text{pred}}}$ so that they match $v_{q,\text{target}}^{t+1:t+N_{\text{pred}}}$, as these are completely independent from where the robots are located in the scene. I will later simply need to numerically integrate the sequence of velocities starting at the robot's current position using the sampling period Δt in order to obtain position predictions.

To achieve accurate predictions, I rely on three different sources of information. Firstly, I consider the past horizon of N_{past} velocities of the query robot $v_q^{t-N_{\text{past}}+1:t}$, defined in an analogous manner to the future sequences seen earlier. Secondly, I take into account the past horizon of N_{past} relative states[1] of the robots other than the query one

$$\mathcal{X}_{r,q}^{t-N_{\text{past}}+1:t} \coloneqq \left\{x_{iq}^{t-N_{\text{past}}+1:t} \mid i \in \mathcal{I} \setminus q\right\}, \tag{4-8}$$

with

$$x_{iq}^{t-N_{\text{past}}+1:t} = \left(x_{iq}^{t-N_{\text{past}}+1}, x_{iq}^{t-N_{\text{past}}+2}, ..., x_{iq}^{t}\right) \tag{4-9}$$

and

$$x_{iq}^t \coloneqq x_i^t - x_q^t = \begin{bmatrix} p_i^t - p_q^t \\ v_i^t - v_q^t \end{bmatrix}. \tag{4-10}$$

Finally, I account for the current relative states of the dynamic obstacles, which are defined similarly:

$$\mathcal{X}_{o,q}^t \coloneqq \left\{x_{iq}^t \mid i \in \mathcal{I}_o\right\}. \tag{4-11}$$

Given these definitions, the training of my deep learning model will involve finding an optimal mapping

$$\left(v_q^{t-N_{\text{past}}+1:t}, \mathcal{X}_{r,q}^{t-N_{\text{past}}+1:t}, \mathcal{X}_{o,q}^t\right) \mapsto v_{q,\text{pred}}^{t+1:t+N_{\text{pred}}}, \tag{4-12}$$

[1] Even though the state space of the robots in Section 4-2-1 include position, speed and orientation, I disregard the orientation when performing trajectory prediction because I found that including it did not lead to significantly better results.

such that the error between $v_{q,\text{pred}}^{t+1:t+N_\text{pred}}$ and $v_{q,\text{target}}^{t+1:t+N_\text{pred}}$ is minimized. The cost function that I use for the training is the mean squared error (MSE) between the prediction and the target, over all the prediction steps:

$$\text{MSE}\left(\left\{v_{q,\text{pred}}^{t+1:t+N_\text{pred}} \,\middle|\, t \in \mathcal{T}\right\}\right) = \frac{1}{|\mathcal{T}| \cdot N_\text{pred} \cdot \dim(\mathcal{W})} \sum_{t\in\mathcal{T}} \sum_{k=1}^{N_\text{pred}} \left\|(v_{q,\text{target}}^{t+k} - v_{q,\text{pred}}^{t+k})\right\|^2, \quad (4\text{-}13)$$

where $\mathcal{W}$ is the workspace the robot lives in and $\mathcal{T}$ is the set of time steps for which a trajectory prediction has been obtained. During training $\mathcal{T}$ will be the batch, during evaluation and testing it will usually be the whole used dataset, and when obtaining a prediction at a specific time step it will be just that time step. I consider the MSE as the cost function because it is one of the more common options for regression problems and it has proven to generally work well in practice. Although it may seem that this choice of cost function weighs the directions of movement in which velocity is larger more heavily, I will scale all the data at a later stage to avoid issues of this nature. As a final note, even though minimizing the velocity prediction error is in theory not equivalent to minimizing the error in position, the performance that will be achieved will be better because of the reasons stated at the beginning of the section.

4-3 Dataset Generation

The data collection step is critical to obtain a deep learning-based regression model that is accurate and generalizes well to new data. It is especially important in this case, as the data that the model is trained on will come from a centralized motion planner, but it will be used to predict trajectories from robots using a decentralized version of it. If the model does not generalize well to this new scenario, the performance of the proposed decentralized approach will be significantly affected, since the estimates of the other robots' future motions will be inaccurate.

4-3-1 Simulator

To generate the data with which to train a prediction model, a centralized sequential solution to the path planning problem formulation in Section 4-2-1 will be implemented. This involves solving Equation (4-3) sequentially for each of the robots $i \in \mathcal{I}$, using the planned optimal trajectories of robots $j \neq i \in \mathcal{I}$ as predictions of their future positions. The used MATLAB-based implementation of this planning method builds on the work of [10], which considers drones with dynamics of the Parrot Bebop 2 quadrotor and uses uses Forces Pro [46] as the solver to the MPC problem.

The enclosing spheres of the robots will be assumed to have a radius equal to 0.3 meters. The dynamic obstacles, which will represent humans moving at constant speed, will be approximated by non-rotating ellipsoidal shapes of dimensions 0.4, 0.4 and 0.9 meters for their X, Y and Z semi-principal axes, respectively. A screenshot of the simulator may be seen in Figure 4-1.

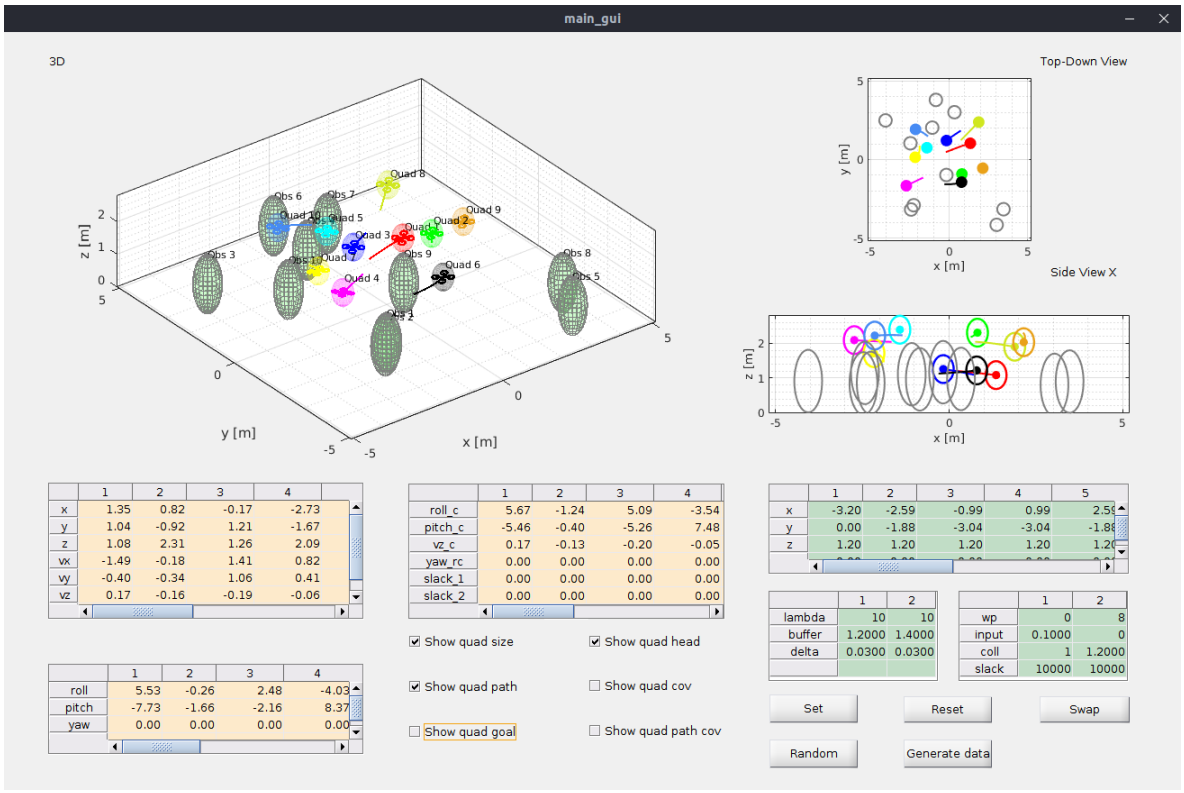


Figure 4-1: Screenshot of the main GUI of the simulator used to generate the data.

4-3-2 Data generation details

In order to generate sufficient training data, I assign the robots random goals which will be changed independently in real time once each of them reaches their own. The random goals are restricted so that there is not any pair of them that is too close together, nor too close to the environment limits or to an obstacle at the time the goal is generated. The condition to consider whether a robot has reached its goal will be based not only on its distance to it but also on its speed. It is important to set the corresponding thresholds low enough so that data of the robots moving at low speeds is collected, but it should not be set so low that the dataset is dominated by this type of behavior. This second problem is especially apparent in very obstacle dense environments, as it is likely that a dynamic obstacle will approach a robot as it is getting close to its goal. Setting very strict stopping criteria in this setting will bias the trained model towards thinking that the robot will engage in obstacle avoiding maneuvers once it is moving at low speeds, since this is what it learned from the data when the robot was close to the goal and had to move away from it because of the incoming dynamic obstacles. In my implementation, I set the thresholds randomly for each independent trajectory to be within a range of 0.1 and 0.3 m for the distance, and between 0.1 and 0.35 m/s for the speed, so that a variety of goal approaching behaviors are reproduced.

Dynamic obstacles have also been included in the environment. These have been assigned a random velocity with a modulus between 0.8 and 1.2 m/s in the XY plane, and between -0.04 and 0.04 m/s in the Z direction, as it is not expected that the humans that these obstacles represent will move fast in the Z direction. Once the obstacles leave the environ-

ment bounds, their position is randomly reset close to them, with their movement direction randomly assigned in a range that does not aim towards the closest bounds. For 20% of the dataset, I have set the obstacles to be static, to ensure that the learned model works adequately in this case as well.

I have set the environment size to be $10 \times 10 \times 3$ m, with 10 robots and 10 dynamic obstacles, as it was observed that these settings produced an adequate amount of interaction among the robots and the environment. Lower numbers may lead to too sparsely occupied environments and not enough interactions among the robots and of the robots with the obstacles. Higher number may bias the trajectory predictor towards being overly reliant on the information about other robots and obstacles rather than using the learned dynamics of the query robot. It must be noted that in the next section I will design a network architecture that can deal with an arbitrary number of robots and obstacles, so the parameters that are set for the training and validation datasets do not restrict the settings in which it will be able to perform predictions.

For the training dataset I have simulated 10^5 steps at a sampling rate of 50 ms, and for the validation one I have run the simulator for $4 \cdot 10^4$ steps. The training and validation data that can be used from these simulations is actually multiplied by the number of robots present, as predictions are made independently for each robot in the scene. I have used datasets with the same number of agents for training and validation because this makes the TensorFlow 2 training script much more computationally efficient (see Section 4-4-3). Otherwise, it would be necessary to create a new instance of the network every time the number of agents in the dataset changed, and then initialize its weights and biases with the trained values. Doing this at each iteration adds a significant overhead and has thus been considered to not be worthwhile. This limitation will not be present during testing, as inference only requires the network to be queried once with all the data we want to obtain a prediction for. Thus, a broader range of scenarios will be tested in Chapter 5.

4-4 Deep Neural Network Design

4-4-1 Network architecture

The proposed network architecture (shown in Figure 4-2) consists of three different modules: the query robot state encoder, the environment encoder and the decoder, all of which will be described in the following sections. All layers in the network except the output one (which is linear) use hyperbolic tangent activation functions and $L2$ regularization for the weights and biases. They all have 64 neurons, except the recurrent decoder, which has 128, and the output layer, which has 3 (one for each coordinate in the workspace).

Query robot state encoder

The first module of the architecture includes a long short-term memory (LSTM) layer that will learn a flat encoding z_q^t from the history of velocities of the query robot $v_q^{(t-N_{\text{past}}+1:t)}$. This encoding must be able to encapsulate the whole past of the query robot, which is critical for the network to be able to predict how it will behave in the future, especially since no

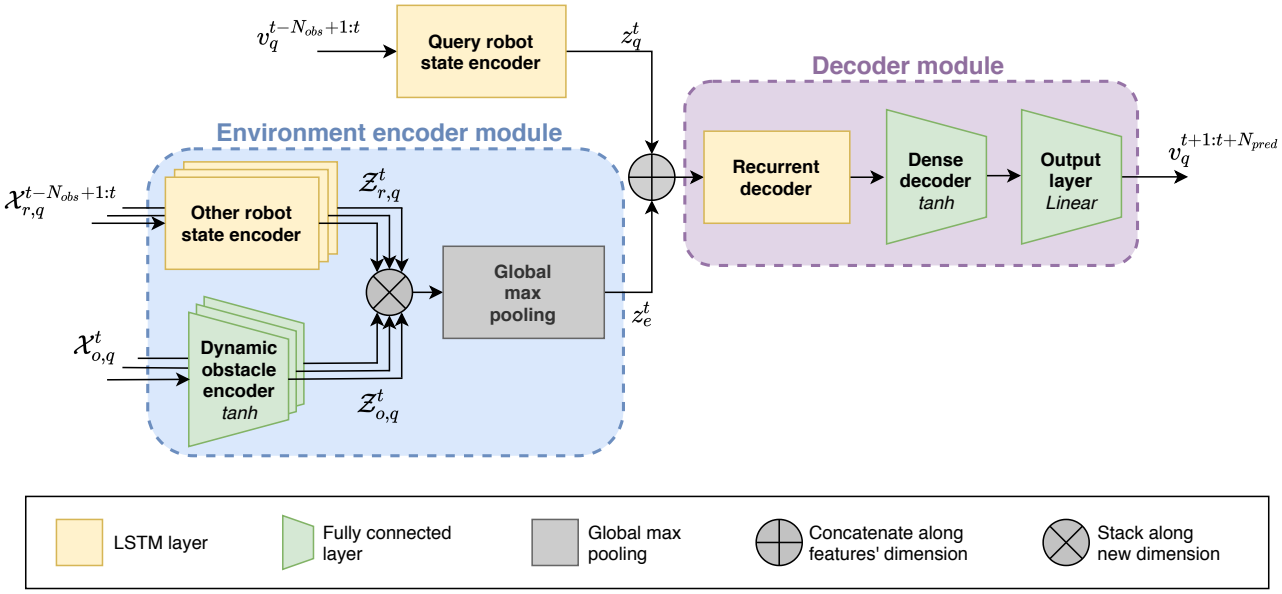


Figure 4-2: Diagram of the proposed neural network architecture. All layers in the network except the output one use hyperbolic tangent activation functions and $L2$ regularization for the weights and biases. They also all have 64 neurons, except the recurrent decoder, which has 128, and the output layer, which has 3 (one for each coordinate in the workspace).

information about the robot's target position is provided. While the other encoding module may be considered to simply add contextual information, this one will need to learn the dynamics of the robot.

Environment encoder module

This module will will learn how the other agents and obstacles that the query robot interacts with will affect its future trajectory. In the considered scenario, these include other robots following the same policy as the query robot, and dynamic obstacles moving at constant speed. Nevertheless, it would be straightforward to extend this module to also consider other types of decision-making agents following different policies by training additional types of encoders.

One of the main challenges of predicting trajectories in dynamic environments is that there may be a varying number of objects in the scene that need to be taken into account. A common solution to deal with this problem is to consider a fixed number of them when designing the neural network architecture, and then use some tricks in order to deal with an arbitrary number. Usually, in the case where there are fewer than expected, the excess inputs to the network are fed with synthetic values such that they will not affect the trajectory prediction. For example, one may input obstacle positions that are far away from the robot. In the opposite case, when there are more obstacles than initially planned for, the most relevant subset of them will be taken into account (usually based on which ones are closer to the query robot).

Although this may work in practice, it is merely an ad hoc solution to the problem. It requires manually tuning the number of objects that the network should consider, and also defining a

metric to determine which of them will have the most influence on the future behavior of the query robot. Using their relative distances as this metric may appear to be a sensible choice, but there are other factors that may be more important, such as the relative velocities and accelerations.

Thus, what I propose is to treat all objects of the same type equally and let the network learn which of them has the most influence in the query robot. To do this, a set of layers with shared weights for each of the object types is first included. In this case there will be two different types: the other robots and the dynamic obstacles. As introduced in Section 4-2-2, the input data for the former will be the set of sequences of relative states $\mathcal{X}_{r,q}^{(t-N_{\text{past}}+1:t)}$, given that the robots have complex dynamics. I encode each of the robots with an LSTM layer, producing the set of encodings $\mathcal{Z}_{r,q}^{t}$. Since the dynamic obstacles simply move at constant speeds, I only consider their relative states at the current time step $\mathcal{X}_{o,q}^{t}$ and use a fully connected layer to process them and produce $\mathcal{Z}_{o,q}^{t}$. It must be noted that all dynamic obstacles are assumed to be of a fixed sized, namely prolate spheroids with horizontal semi-axes of length 0.4 and verical semi-axis of length 0.9. The choice of these values is justified by the fact that in real scenarios dynamic obstacles will normally be humans, the bounding boxes of which resembles the chosen shape.

I then stack the outputs from both of these sets of layers along a new axis, and will perform a global max pooling operation over it, resulting in vector z_e^t. This operation is the one that will determine which specific learned features of the environment objects' motions will have the most effect on the query robot. It must be noted that this requires both sets of layers to have the same number of units, as otherwise it would not be possible to stack them both. This framework can natively deal with any number of obstacles, and is invariant to the order in which they are fed to the network. Furthermore, extending it to deal with a different type of object would simply require us to train an additional specific encoder and stack the output to the others.

I believe this approach of using a global max pooling operation is more reliable than others found in the literature in which a bidirectional LSTM layer is used instead, and in my experiments it has also worked better. The reason for this is that even though LSTMs are suitable for dealing with data of varying length, they are not invariant to the order in which the data appears in the sequence. Thus, it would require the outputs from the previous layer to be arranged in some manner if we wanted to obtain consistent results, and this conflicts with the previous argument against using a manually defined metric to measure the influence of the environment objects.

Decoder module

The final part of the network will take in the concatenated outputs from the query robot state encoder and the environment encoder module, and will reconstruct a prediction for the velocity of the query robot. It includes an LSTM layer which will be fed a sequence of length N_{pred}, with each element in the sequence being the concatenation of z_q^t and z_e^t. I will additionally store the sequence of internal states from this layer, instead of just the last one. This way, I will be able to produce velocity predictions for the N_{pred} future steps in the trajectory.

After it, two dense layer will be included, the operations of which will be applied with the same weights and biases to each of the sequence steps from the LSTM's output. The first of these layers will have a hyperbolic tangent activation function (as all the other layers), and it is included to increase the overall model complexity. The second one will have a linear activation and three units, and it is added so that the output of the network matches the size of the target data, which is a three-dimensional velocity vector.

4-4-2 Evaluation metrics

As it has been introduced in Section 4-2-2, I use the MSE of the velocity prediction with respect to the actual one as the cost function of the training algorithm. Even though comparing different networks based on their loss on a test set might be useful as a rough evaluation of which one is most accurate, it is difficult to understand the scale of these losses and their actual meaning, given that the MSE is computed over the whole prediction horizon N_{past}. It is clear that as this horizon increases, the error will grow larger, since the further into the future we try to make predictions, the more uncertain they will become. This is the case even when predicting velocities instead of positions. However, the MSE does not show that the main contributing factor to a large loss comes from the later prediction steps. Furthermore, since the MSE is computed based on velocities (see Equation (4-13)), the magnitude of the error in positioning is even less clear.

Therefore, to evaluate the performance of the network, I have decided to compute the average displacement error at each of the N_{pred} steps of the prediction horizon. This involves computing the Euclidean distance (or $L2$ norm) between the actual future position of the robot and the predicted one coming from the model, and averaging them over each prediction step. Thus, we will be able to see how the error evolves over time, unlike with the MSE.

4-4-3 Implementation details

I have implemented the described network architecture using TensorFlow 2 and trained it on a laptop with an Intel i7 7850H processor, 16 GB of RAM and an NVIDIA GTX 1060 GPU with 6 GB of VRAM, taking an hour of training on the GPU, but already producing accurate predictions at around 30 minutes. On this same computer it takes around 30 ms to perform inference with 12 robots and 12 obstacles, both on the CPU and the GPU, without changing any settings to accelerate inference. Even though it is unlikely that numbers significantly higher than these will be considered for normal applications, the network scales adequately with bigger team sizes, providing predictions in slightly over 40 ms with a squad of 100 robots and no obstacles.

The size of all the layers in Figure 4-2 is 64 units, except the recurrent decoder which has 128, and the output layer, which has 3. All layers have a hyperbolic tangent activation function except the output one, which is linear. The network has been trained with observed and prediction horizons of length 20 (which is also the planning horizon of the data-generating MPC), with sampling period being 50 ms. The datasets that have been used for training and validation have been independently generated, using randomized goals, obstacle velocities and stopping criteria (see Section 4-3-2 for details). All data has been scaled to be in the [-1, 1] range, and $L2$ regularization has been used for all the layers' weights and biases with a

factor of 0.01. Basic testing has been performed on a third dataset[2] obtained independently from the training and validation ones but with the same number of robots and obstacles. A broader range of scenarios will be tested in Chapter 5.

4-5 Decentralized Multi-Robot Motion Planning

Having implemented a centralized sequential solution to the motion planning problem from Equation (4-3) in Section 4-3-1, and a robot trajectory predictor suitable for this setting in Section 4-4, extending the planning algorithm to work in a decentralized manner is straightforward. It requires each robot to perform inference with the trained neural network to predict the trajectories of neighboring robots, instead of relying on their planned optimal trajectories, which are not available without communication. To be able to perform inference, each robot must be able to measure its own state and those of its neighbors, and keep a history of length N_{past}, as this data is necessary to compute the inputs to the network. It is assumed that this information, in addition to the states of the obstacles in the environment, is available to the robots without error. In the next chapter, the performance of the proposed trajectory prediction RNN and decentralized multi-robot motion planning method will be analyzed in detail.

[2]We use the validation dataset to determine when to stop the training of the network so as to avoid overfitting, and the test one to determine which of all the trained networks performs best.

Chapter 5

Results and Discussion

In this chapter I analyze the performance of both the trajectory predictor by itself (Section 5-1) and of the proposed decentralized multi-robot motion planner (Section 5-2), which relies on the trajectory predictor in order to have interaction awareness.

5-1 Interaction-Aware Trajectory Prediction

In order to make sure that the proposed interaction-aware trajectory prediction model generalizes well to all possible scenarios, I test its performance on datasets collected with the centralized version of the model predictive control (MPC)-based planner which have not been used for training nor validation. I simulate $4 \cdot 10^4$ steps of four different scenarios in which the robots will move towards goals which will randomly change in real time for each of them once they get closer than a distance threshold. This number of steps means a simulation of over half an hour of real time for each scenario, given that the sampling period is 50 ms. The scenarios that will be considered are an open environment with 4 and 10 quadrotors, and the same environments but with a number of dynamic obstacles equal to the number of robots (i.e. 4 and 10, respectively). This will help us test the proposed prediction model with varying team sizes and obstacle densities, and analyze how these parameters affect its performance. In order to objectively analyze the results, I compute the average displacement error (based on the Euclidean distance or $L2$ norm) at each prediction step for each of the datasets.

The baselines to which the proposed recurrent neural network (RNN)-based model is compared are the planned paths of the centralized MPC used to generate the datasets, and a constant velocity model (CVM) which assumes that robots will keep the speed at the current step for the whole prediction horizon. Even though the purpose of the MPC is not to be a prediction model but to perform the planning, using it as a baseline is useful because the prediction model will be substituting the shared trajectories planned by the MPC. It also gives us a reference of how good the trajectory predictions should be for the path planner to work well. To avoid confusions, it must be noted that I do not use these planned trajectories as the

target predictions for the network: I use the real future trajectories that the robot will follow. This means that in theory, it should be possible to obtain predictions that are more accurate than these planned trajectories.

The average errors for each of the 4 different scenarios at 5, 10, 15 and 20 time steps into the future can be observed in Tables 5-1 to 5-4, corresponding to 250, 500, 750 and 1000 ms into the future, given that the sampling period is 50 ms. Continuous plots of the evolution of the mean error over the total 20 prediction steps are shown in Figure 5-1, where the drawn prediction uncertainty is $\pm 30\%$ of the standard deviation. As it was explained in the previous chapter, this choice of the prediction horizon matches the planning horizon of the MPC. Example predictions in the environment with 10 quadrotors and no obstacles can be seen in Figures 5-2 and 5-3, and for the environment with 10 quadrotors and 10 dynamic obstacles, in Figures 5-4 and 5-5.

In Figure 5-1 we can see how the average performance stays consistent throughout the different scenarios for all prediction models, with the optimal planned trajectory from the centralized sequential MPC-based planner being the most accurate, my RNN-based model being a close second, and the CVM performing significantly worse. The only meaningful different is for the case with 4 quadrotors and no obstacles, where the average error of the trained prediction model is slightly worse than in other cases when compared to the MPC baseline. This might seem counterintuitive, but it owes to the fact that the model not only relies on the learned dynamics of the query robot to produce predictions for it, but also on the other robots and dynamic obstacles in the scene. This seemingly high reliance on environment information to produce accurate predictions is likely also a consequence of my choice of an obstacle-dense training dataset with 10 robots and 10 obstacles. Had I used a less populated simulation scenario to generate the training data, we might observe a different variation in performance throughout the considered scenarios.

An important observation to be made from the shown summary tables is that despite my model being close in terms of average error to the MPC baseline, it shows a higher standard deviation throughout all the prediction horizon. This reflects the higher uncertainty of the model's predictions with respect to the MPC's planned trajectories, which are more consistent. However, it must be noted that, unlike the MPC, the neural network is not fed any information about the quadrotor's goals. It infers where the robots are heading based on their state trajectories leading up to the current time step. Even though one would expect this to significantly limit its long-term prediction capabilities, prediction performance degrades over time in a similar manner as the MPC's. The consistency of the MPC baseline is to be expected, as it is the planner that was used to generate the data. Another interesting phenomenon which can be observed in Figure 5-1 is that the average accuracy of the predictions for the scenario with 10 quadrotors does not seem to significantly change when obstacles are added. Instead, the standard deviations of the errors across all the prediction horizon are larger, reflecting a bigger uncertainty in the predictions.

Model	Average $L2$ error at 5 steps (m)	Average $L2$ error at 10 steps (m)	Average $L2$ error at 15 steps (m)	Average $L2$ error at 20 steps (m)
Constant velocity model	0.036 ± 0.017	0.141 ± 0.064	0.312 ± 0.138	0.540 ± 0.234
Centralized MPC planned trajectory	0.001 ± 0.002	0.012 ± 0.015	0.053 ± 0.048	0.142 ± 0.104
RNN-based prediction model	0.021 ± 0.018	0.054 ± 0.046	0.116 ± 0.099	0.216 ± 0.184

Table 5-1: Comparison of the prediction performance of the different baseline models and the proposed prediction model in an environment with 4 robots and no obstacles.

Model	Average $L2$ error at 5 steps (m)	Average $L2$ error at 10 steps (m)	Average $L2$ error at 15 steps (m)	Average $L2$ error at 20 steps (m)
Constant velocity model	0.036 ± 0.016	0.142 ± 0.064	0.311 ± 0.137	0.535 ± 0.233
Centralized MPC planned trajectory	0.001 ± 0.002	0.012 ± 0.015	0.055 ± 0.048	0.145 ± 0.103
RNN-based prediction model	0.010 ± 0.006	0.029 ± 0.027	0.076 ± 0.075	0.160 ± 0.154

Table 5-2: Comparison of the prediction performance of the different baseline models and the proposed prediction model in an environment with 4 quadrotors and 4 dynamic obstacles.

Model	Average $L2$ error at 5 steps (m)	Average $L2$ error at 10 steps (m)	Average $L2$ error at 15 steps (m)	Average $L2$ error at 20 steps (m)
Constant velocity model	0.036 ± 0.015	0.141 ± 0.061	0.309 ± 0.131	0.532 ± 0.222
Centralized MPC planned trajectory	0.001 ± 0.001	0.012 ± 0.014	0.054 ± 0.047	0.143 ± 0.101
RNN-based prediction model	0.011 ± 0.007	0.032 ± 0.027	0.079 ± 0.074	0.165 ± 0.151

Table 5-3: Comparison of the prediction performance of the different baseline models and the proposed prediction model in an environment with 10 robots and no obstacles.

Model	Average $L2$ error at 5 steps (m)	Average $L2$ error at 10 steps (m)	Average $L2$ error at 15 steps (m)	Average $L2$ error at 20 steps (m)
Constant velocity model	0.037 ± 0.092	0.142 ± 0.141	0.307 ± 0.200	0.521 ± 0.279
Centralized MPC planned trajectory	0.002 ± 0.071	0.016 ± 0.115	0.062 ± 0.150	0.154 ± 0.189
RNN-based prediction model	0.011 ± 0.091	0.032 ± 0.130	0.081 ± 0.169	0.167 ± 0.224

Table 5-4: Comparison of the prediction performance of the different baseline models and the proposed prediction model in an environment with 10 quadrotors and 10 dynamic obstacles.

5-2 Decentralized Multi-Robot Motion Planning

Having verified that our proposed RNN-based prediction model performs as expected, I now integrate it into the MPC framework, so that we may perform decentralized multi-robot motion planning without need of communication, by predicting other robot's trajectories with our model. I compare the performance of this approach to the centralized MPC-based planner as well as a distributed version in which planned trajectories are shared among robots via communication, as described in Section 3-2-2. Furthermore, I also use a decentralized

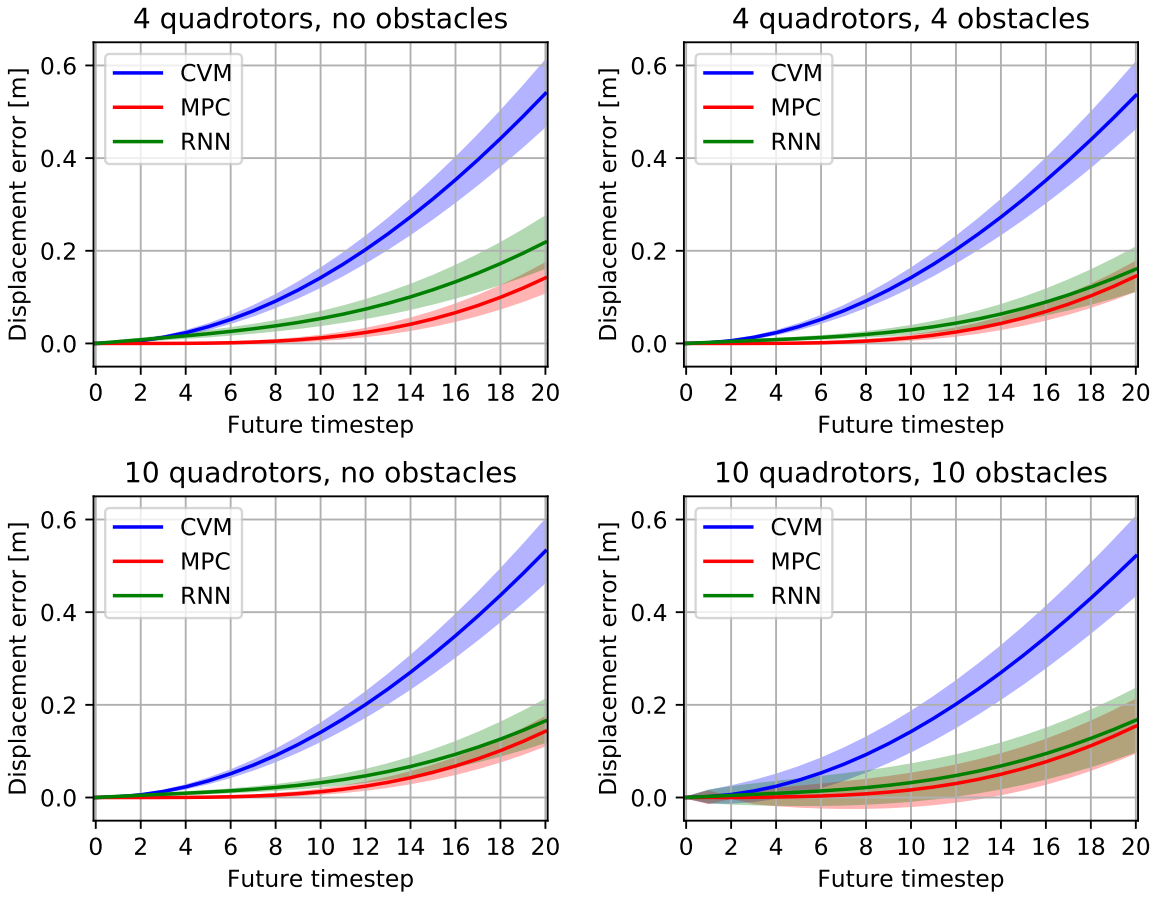


Figure 5-1: Plot showing the displacement error of our model (RNN) and two other baselines (CVM and MPC), averaged over each of the prediction time steps, and computed based on the $L2$ distance between the prediction and the ground truth. Performance is evaluated in four different settings, in all of which the robots' goal positions are changed randomly once they are reached. In the figure it is also drawn $\pm 30\%$ of the standard deviation around the average values, as a measure of the uncertainty in the predictions. The sampling period is 50 ms, so the whole prediction horizon is equivalent to 1 second.

version of the path planner which uses a simple CVM, to analyze whether the more accurate predictions of our RNN-based model lead to significantly better planning performance or not.

For these tests, I focus exclusively on obstacle-free scenarios, since the way in which obstacles are accounted for is equivalent within all considered versions of the MPC: they are all assumed to keep a constant speed, which is indeed how they are programmed to behave in the simulation. Furthermore, we have already seen in the previous section that adding obstacles does not have a meaningful effect on the overall prediction performance, only leading to increased uncertainty in the predictions.

I will analyze 4 different types of test scenarios with 6 quadrotors, each of them with 50 unique sets of start and goal positions. The first of these consists of a *symmetric swap*, in which the 6 quadrotors start from the vertices of a virtual regular hexagon that is parallel to the ground plane and they need to exchange positions with the robot in the opposing vertex. Each of the

different cases within this group will feature different hexagon sizes and rotation along the Z axis. The second class of tests consists of *asymmetric swaps*, which differ from the previous ones in that each hexagon is no longer regular, thus leading to more challenging collision-avoidance problems. All the start and goal positions are still placed at the same height, and the shape and size of the hexagon changes for the different cases within the class. Next, I will require the robots to perform *pair-wise swaps*, in which the six robots are grouped in three pairs and placed at random positions in the environment. Each robot will then exchange positions with its respective pair. This scenario is less challenging because the straight lines that unite start to goal positions are very unlikely to intersect with each other, so robots no longer attempt to go through the center of the environment in order to minimize their trajectory lengths. Finally, I make the robots move from *random* start positions to random goals.

To numerically evaluate the performance of the different motion planners in these experiments I compute the minimum distance between any two quadrotors during the whole simulation, and I also count how many of the 50 sets of trajectories includes at least one collision between a pair of robots. Furthermore, I analyze the trajectory lengths and durations, and I calculate the average linear velocities throughout each scenario. A summary of the obtained results can be seen in Table 5-5. For a qualitative performance analysis, I show sample trajectory trails for a specific set of start to goal trajectories from each of the scenarios for each of the methods in Figures 5-6 to 5-21.

From the metrics shown in Table 5-5, we can see how using our RNN-based prediction model significantly increases performance of the decentralized MPC-based planner when compared to using a CVM, leading to consistently lower trajectory lengths and durations on average. More importantly, our proposed method greatly diminishes the likelihood of collisions in the more challenging scenarios such as the asymmetric swap, where the number of colliding trajectories was reduced from 15 to only 3, out of the total 50. Furthermore, in the symmetric and pair-wise swap scenarios collisions were avoided altogether when using our proposed model instead of the CVM, which produced 4 and 3 collisions in these experiments, respectively. Even more remarkably, the decentralized MPC-based planner using our model achieved similar performance in terms of average trajectory length, duration and velocity as the centralized and distributed counterparts. Nevertheless, despite being comparable in terms of average performance, the standard deviation for the trajectory lengths and times is slightly larger in our proposed method, meaning more variability in the resulting trajectories. That was to be expected, especially after looking at the results from Section 5-1, as the uncertainty of the predictions coming from our model is higher than for those planned with the MPC. This phenomenon is also observed in the trajectory trails that have been plotted. Especially those of the asymmetric swap setting in Figure 5-13, where the resulting path from our method is more erratic than those of either the centralized (Figure 5-10) or the distributed (Figure 5-11) MPC-based planners.

Scenario	Motion planner	Min. dist. (m)	Num. of coll.	Trajectory length (m) Min.	Max.	Avg.	Trajectory time (s) Min.	Max.	Avg.	Avg. vel. (m/s)
Symmetric swap	Centralized	0.80	0	5.46	8.96	7.22 ± 0.85	5.15	6.05	5.62 ± 0.24	1.27
	Distributed	0.80	0	5.52	8.73	7.26 ± 0.84	4.75	6.40	5.51 ± 0.35	1.31
	Decen. (CVM)	0.32	4	5.43	9.55	7.45 ± 0.88	4.95	7.30	6.02 ± 0.55	1.23
	Decen. (RNN)	**0.75**	**0**	**5.41**	**10.60**	$\mathbf{7.35 \pm 0.91}$	**4.75**	**9.7**	$\mathbf{5.59 \pm 0.40}$	**1.30**
Asymmetric swap	Centralized	0.80	0	5.08	9.02	6.77 ± 0.80	4.65	5.85	5.17 ± 0.30	1.30
	Distributed	0.80	0	4.99	8.84	6.82 ± 0.84	4.65	6.25	5.12 ± 0.33	1.32
	Decen. (CVM)	0.16	15	5.32	18.06	7.76 ± 1.89	5.05	7.30	5.96 ± 0.51	1.28
	Decen. (RNN)	**0.37**	**3**	**5.16**	**12.60**	$\mathbf{7.14 \pm 1.10}$	**4.80**	**6.85**	$\mathbf{5.48 \pm 0.42}$	**1.29**
Pair-wise swap	Centralized	0.80	0	1.64	9.92	5.10 ± 1.87	3.50	5.85	4.76 ± 0.44	1.06
	Distributed	0.80	0	1.82	9.75	5.10 ± 1.86	3.60	5.85	4.72 ± 0.43	1.07
	Decen. (CVM)	0.34	3	1.74	17.42	5.54 ± 2.53	3.60	5.75	5.00 ± 0.65	1.06
	Decen. (RNN)	**0.79**	**0**	**1.70**	**9.94**	$\mathbf{4.94 \pm 2.02}$	**3.40**	**5.95**	$\mathbf{4.83 \pm 0.45}$	**1.01**
Random	Centralized	0.80	0	0.39	8.63	4.66 ± 1.94	3.50	5.50	7.72 ± 0.40	0.98
	Distributed	0.8	0	0.39	8.73	4.67 ± 1.93	3.50	5.40	4.70 ± 0.40	0.98
	Decen. (CVM)	0.76	0	0.39	9.53	4.82 ± 2.06	3.60	6.35	4.89 ± 0.57	0.98
	Decen. (RNN)	**0.78**	**0**	**0.39**	**9.19**	$\mathbf{4.36 \pm 2.11}$	**3.85**	**5.95**	$\mathbf{4.76 \pm 0.43}$	**0.91**

Table 5-5: Comparison of the performance of the four considered MPC-based motion planner variations (centralized, distributed, decentralized with CVM and decentralized with our RNN-based model) across the four different types of scenarios that have been tested (symmetric swap, asymmetric swap, pair-wise swap and random goals). For collision detection, the quadrotors' shapes are considered to be spheres with 0.3 m of radius, as explained in Section 4-3-1.

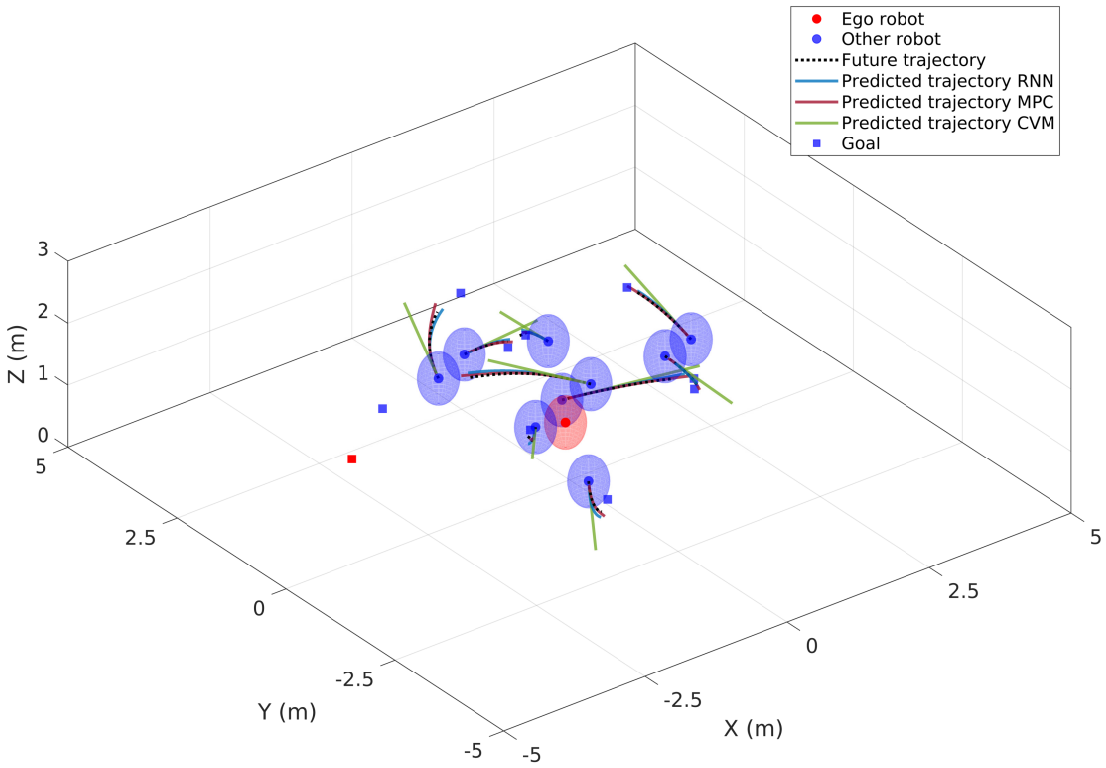


Figure 5-2: Sample screenshot of predictions in an obstacle-free environment.

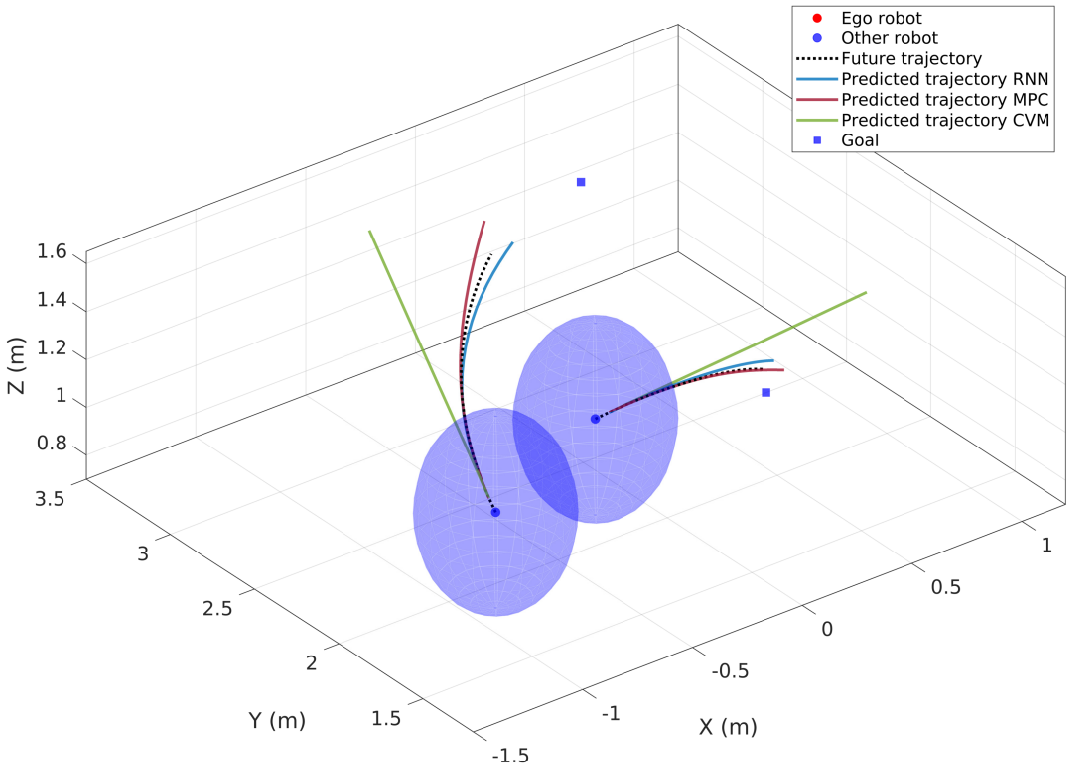


Figure 5-3: Sample screenshot of predictions in an obstacle-free environment (zoomed-in).

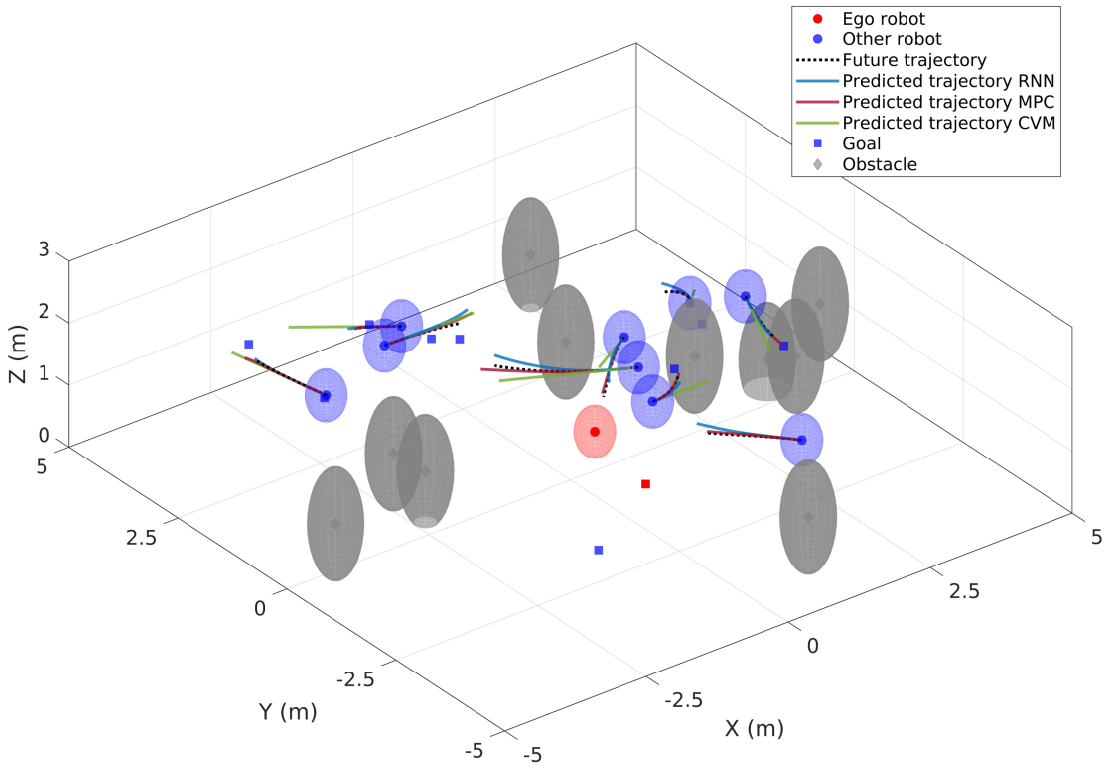


Figure 5-4: Sample screenshot of predictions in an obstacle-dense environment.

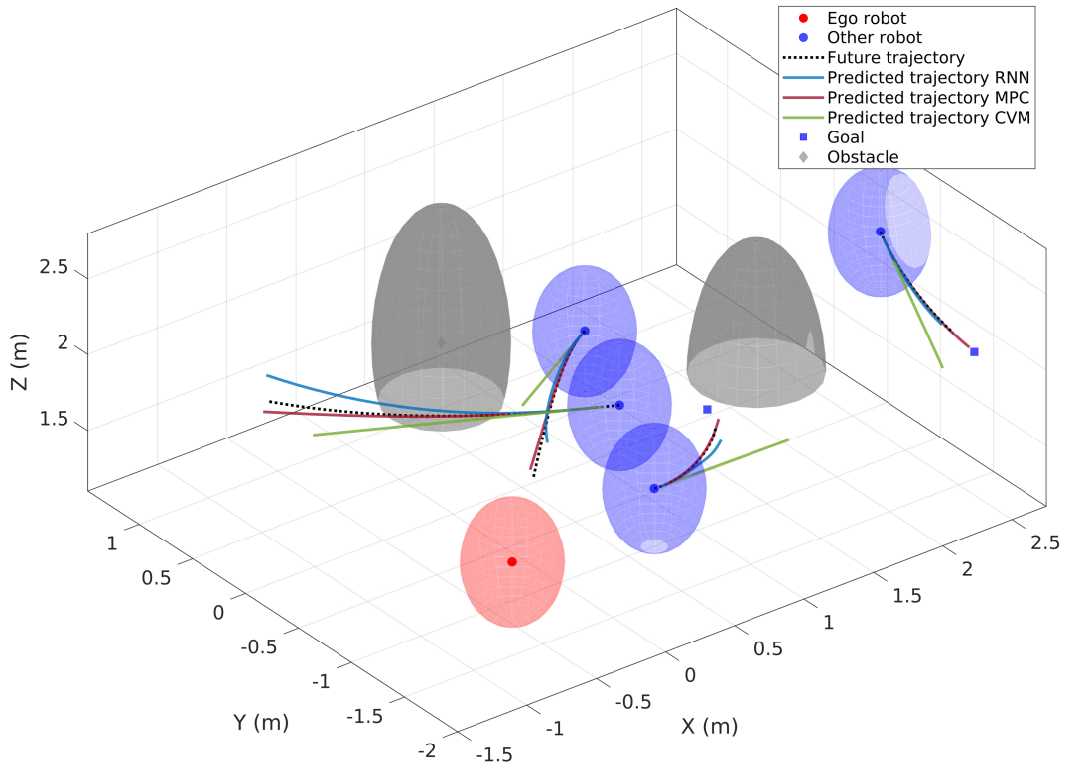


Figure 5-5: Sample screenshot of predictions in an obstacle-dense environment (zoomed-in).

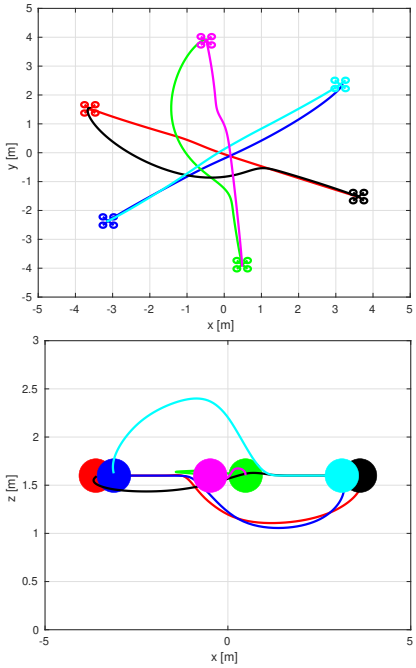


Figure 5-6: Resulting paths for one of the symmetric swap scenarios with the centralized sequential MPC-based planner.

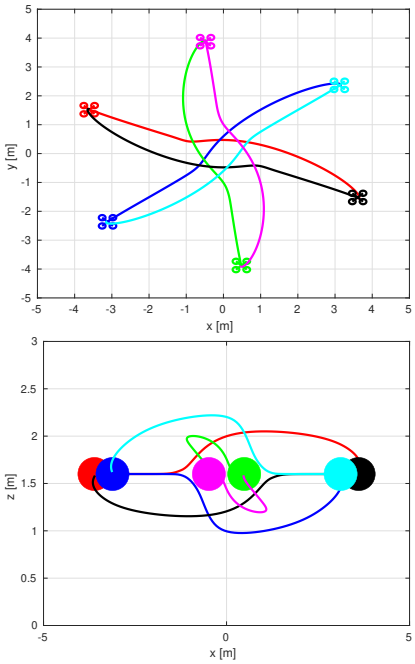


Figure 5-7: Resulting paths for one of the symmetric swap scenarios with the distributed MPC-based planner.

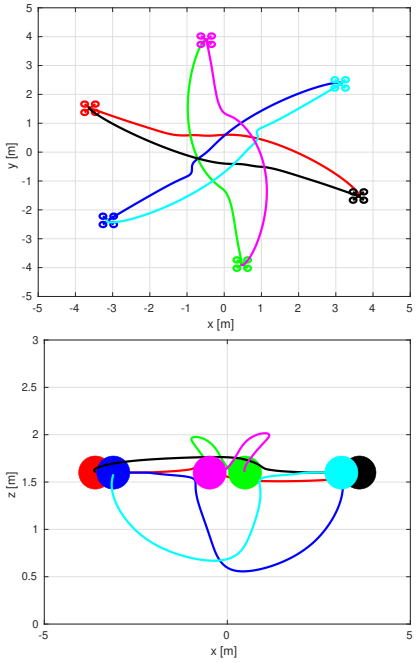


Figure 5-8: Resulting paths for one of the symmetric swap scenarios with the decentralized MPC-based planner with a CVM.

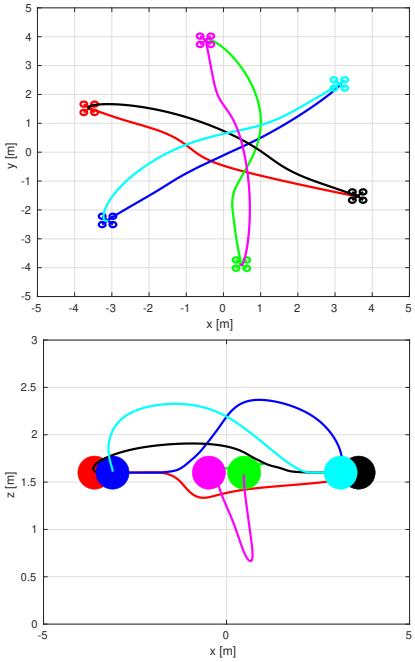


Figure 5-9: Resulting paths for one of the symmetric swap scenarios with the decentralized MPC-based planner with our RNN-based model.

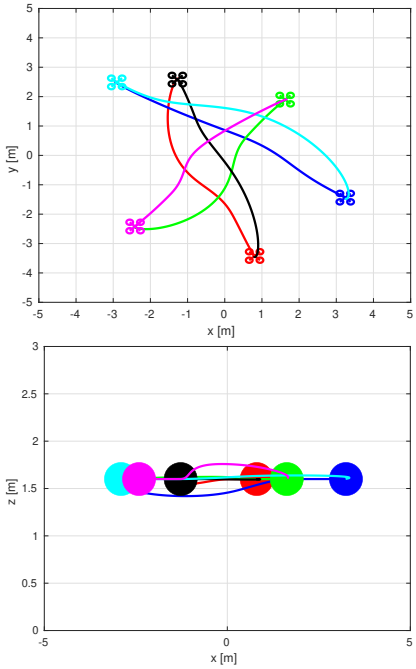


Figure 5-10: Resulting paths for one of the asymmetric swap scenarios with the centralized sequential MPC-based planner.

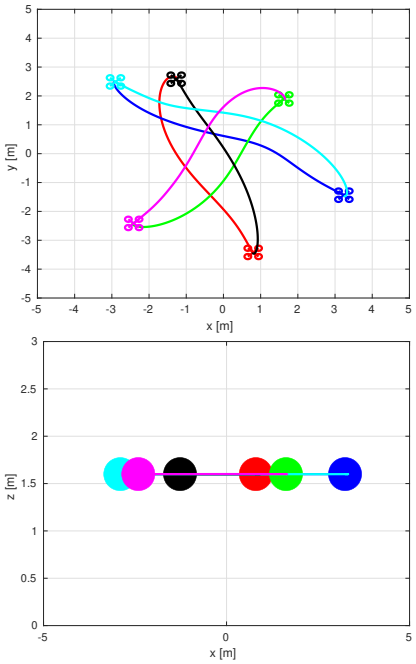


Figure 5-11: Resulting paths for one of the asymmetric swap scenarios with the distributed MPC-based planner.

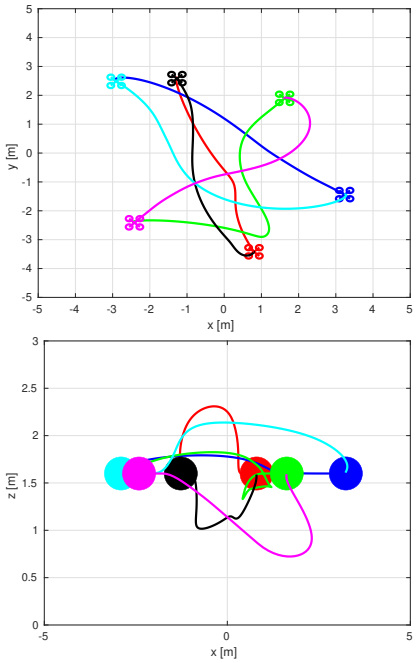


Figure 5-12: Resulting paths for one of the asymmetric swap scenarios with the decentralized MPC-based planner with a CVM.

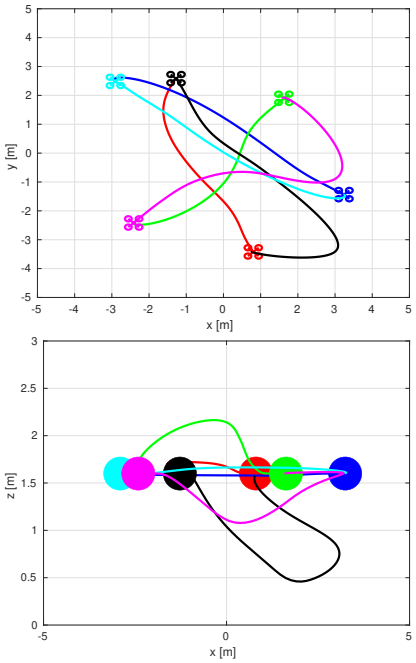


Figure 5-13: Resulting paths for one of the asymmetric swap scenarios with the decentralized MPC-based planner with our RNN-based model.

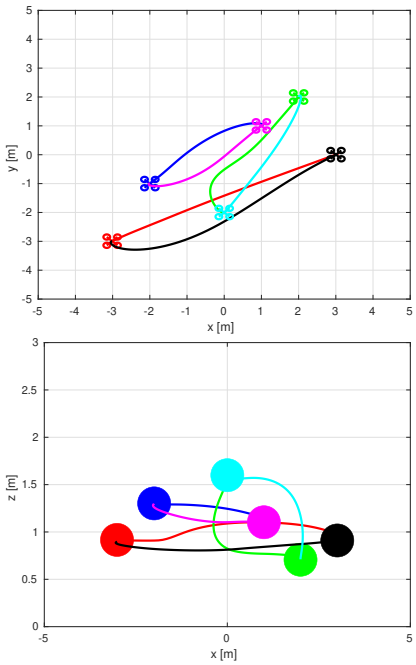


Figure 5-14: Resulting paths for one of the pair-wise swap scenarios with the centralized sequential MPC-based planner.

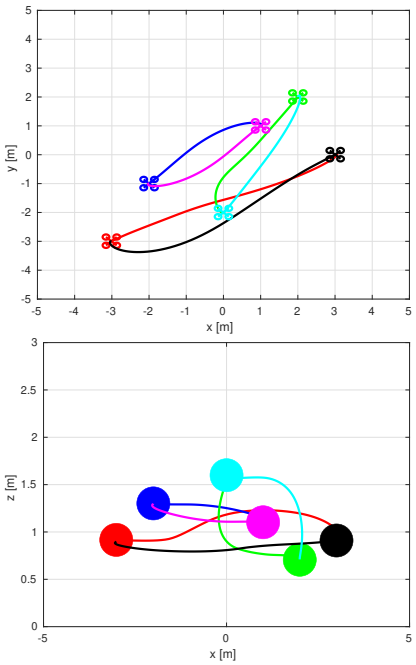


Figure 5-15: Resulting paths for one of the pair-wise swap scenarios with the distributed MPC-based planner.

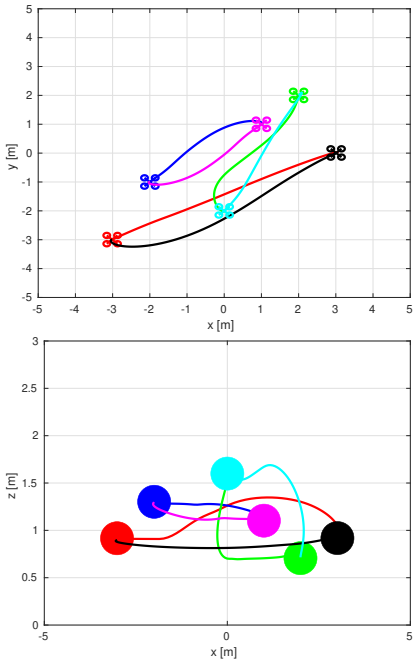


Figure 5-16: Resulting paths for one of the pair-wise swap scenarios with the decentralized MPC-based planner with a CVM.

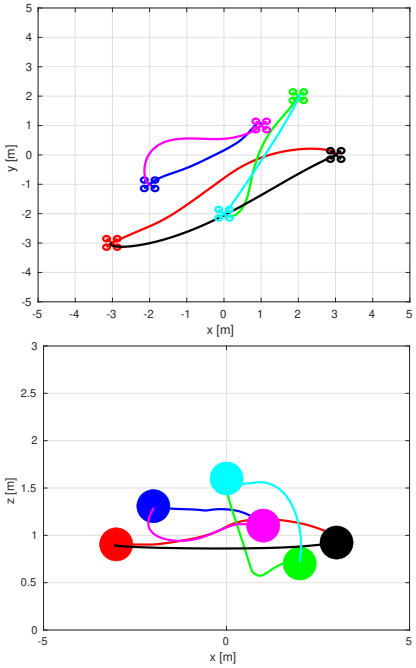


Figure 5-17: Resulting paths for one of the pair-wise swap scenarios with the decentralized MPC-based planner with our RNN-based model.

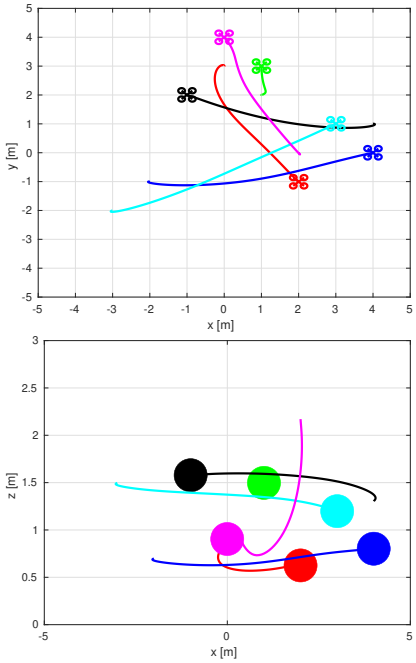


Figure 5-18: Resulting paths for one of the random scenarios with the centralized sequential MPC-based planner.

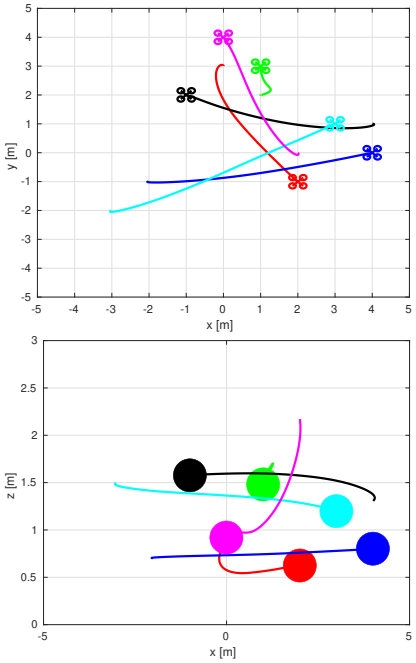


Figure 5-19: Resulting paths for one of the random scenarios with the distributed MPC-based planner.

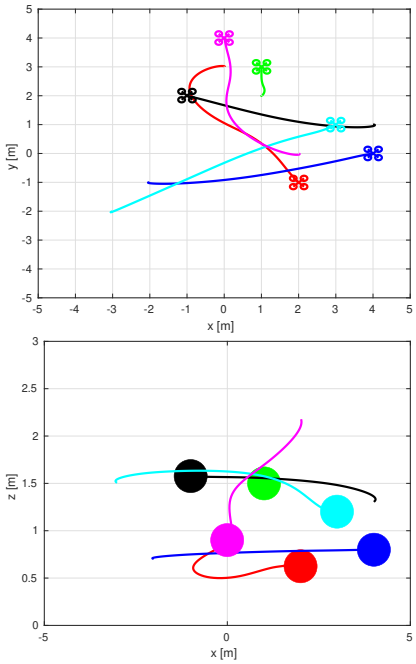


Figure 5-20: Resulting paths for one of the random scenarios with the decentralized MPC-based planner with a CVM.

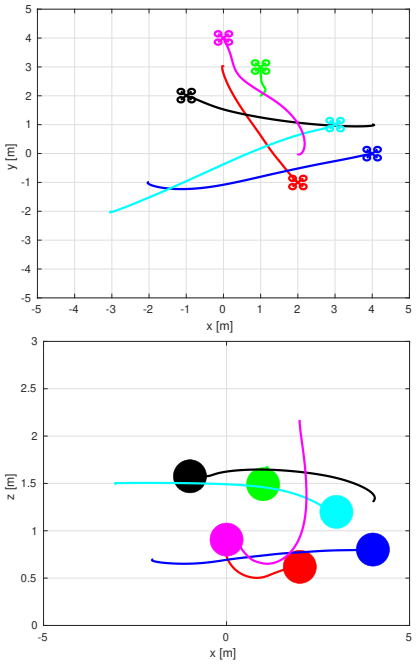


Figure 5-21: Resulting paths for one of the random scenarios with the decentralized MPC-based planner with our RNN-based model.

Chapter 6

Conclusions and Future Work

6-1 Conclusions

In this thesis I have proposed interaction-aware communication-free decentralized motion planner for multi-robot systems in dynamic three-dimensional environments. To achieve this, I have first designed a recurrent neural network (RNN) architecture that is able to learn how a robot in a team will behave in the future based on its observed past behavior and the observed past behaviors of other robots and of the dynamic obstacles in the scene. In our tests, our trained network produced accurate predictions in a variety of scenarios, being able to deal with an arbitrary number of robots and obstacles through the use of a global max pooling layer. However, I did observe a slight degree of overfitting towards the obstacle density used when generating the training dataset, becoming slightly less accurate in very sparsely populated environments.

To generate the data for training, I have used a centralized model predictive control (MPC)-based planner that considers dynamic obstacles and robots other than the one we are planning for as constraints on its future positions. The positions that dynamic obstacles will occupy in the future are predicted through a constant velocity model (CVM), while the ones of other robots are determined by the solutions to their respective MPC problems. Under this framework, it was straightforward to decentralize the path planner and use the predictions made by the neural network instead of the MPC-planned optimal trajectories to forecast robots' future positions over the prediction horizon. In my experiments, the observed performance degradation from my method with respect to the centralized and distributed versions of the MPC-based planner was minimal. Furthermore, I demonstrated that a more accurate prediction model for other robots leads to a better planning performance when I compared the decentralized MPC-based planner using my RNN-based model to the same planner using a CVM instead.

When compared to other learning-based decentralized motion planning strategies such as reinforcement learning (RL), my approach has the main advantage that it provides a collision avoidance guarantee through the use of an MPC-based policy for planning. Furthermore, the

proposed approach is able to deal with problems that are notoriously hard for traditional RL algorithms, such as stabilizing an unstable plant with complex dynamics, and handling three-dimensional workspaces. Decoupling the motion planning problem into behavior prediction and the path planning itself also makes it easier to debug my method when compared to end-to-end machine learning-based approaches. Additionally, even though these other methods may be able to theoretically find the globally optimal control policy, ours allows for a greater error tolerance from the machine learning model, as it is only used for prediction, instead of directly outputting the control actions for the robot. Thus, even if predictions are not perfectly accurate, the ego robot should still be able to avoid collisions thanks to the MPC. Another shortcoming from RL methods when compared to my mixed approach is that they are usually short-sighted, as they tend to only plan for the next time step based on the current state of the system. Due to the nature of MPC, the proposed method avoids collisions over a finite number of steps into the future.

6-2 Future Work

During my tests I have trained and tested the neural network on a variety of datasets. This process has lead us to notice how challenging it is to generate a set of data that includes all possible scenarios that robots navigating in three-dimensional space may encounter. Even though my network can deal with an arbitrary amount of agents and obstacles through the use of global pooling layers, TensorFlow does not natively allow changing the number of layers during training once the network has been initialized, even if the have shared weights, thus forcing the training dataset to only include data with a certain number of robots and obstacles. As already mentioned, I have noticed a slight overfitting on datasets with obstacle densities similar to those of the one used for training. I believe it would be interesting to explore options to make it possible to deal with a varying number of robots and obstacles during training other than initializing the network with a high enough number of inputs and padding the unused ones. I believe that if I were able to solve this problem, the generalizability of my network could be improved further.

Another issue which I have noticed during simulations of the proposed motion planner is that the uncertainty in my model's predictions becomes quite large when robots are close together, making the MPC-planned trajectories sometimes rapidly shift around the general direction that the robot will end up following. A possible solution to this issue could be to train a variational model and estimate the uncertainty of the predictions, so that the distribution of possible predictions may change more smoothly from one time step to the next. I think that solving this issue is a worthwhile effort, as it could lead to a more reliable motion planner.

A completely different extension which would also be interesting is making the network able to deal with a higher variety of obstacle shapes and sizes. A first step would be to consider ellipsoidal obstacles of arbitrary semi-axes dimensions, instead of the fixed ones which I considered for my method. I have done some preliminary tests in this direction by parameterizing the obstacles' inputs in such a way that they contain the dimensions of the ellipsoids, but I did not see a significant improvement in prediction performance over the current network implementation so I did not explore this option any further. Another direct extension to my architecture would be to include the planned trajectory of the ego robot or its goal position as an input to the network. This type of information is available

during inference time without need of communication, and it could potentially lead to better network predictions for the other robots, as I could explicitly account for how the ego robot's intentions will affect the paths of other robots. Finally, another option to improve prediction performance would be to consider a different network architecture altogether. Given the nature of the problem, I could model the relations among the robots and of the robots with the environments with a spatio-temporal graph, and use a graph neural network to learn the underlying dynamics.

On the motion planning side, a significant limitation of the chosen MPC path planning problem formulation comes from the fact that I consider other robots as dynamic obstacles which need to be avoided in the future. This simplification of other robots' behavior works well in practice when the sampling rate is fast enough, but it leads to trajectories that are far from globally optimal, as robots do not cooperate with each other. If the robots tried to optimize for a joint team objective that accounted for others' trajectories as well as one's own, paths that are more efficient overall may be planned. However, solving this problem is not trivial, especially in a communication-less decentralized manner, so more work in this direction is needed.

Finally, in my tests I have assumed the positions of other robots and obstacles to be known without error. In practice, these will have to be estimated from sensor data , which will add some inaccuracies. It should be explored how these inaccuracies affect overall performance, to verify whether the robustness of my method needs to be improved in order for it to be applied to a team of robots in the real world.

Appendix A

Paper

In this appendix I include a draft version of the research paper that has resulted from this thesis' work, which we intend to submit to the IEEE Robotics and Automation Letters (RA-L) with the option of a presentation at the 2021 IEEE International Conference on Robotics and Automation (ICRA).

Learning Interaction-aware Trajectory Predictions for Multi-Robot Motion Planning in Dynamic Environments

Francisco Martínez Claramunt, Hai Zhu, Bruno Brito and Javier Alonso-Mora

***Abstract*— This paper presents a data-driven decentralized approach for multi-robot motion planning in dynamic environments. When navigating in a shared space, each robot needs accurate motion predictions of neighboring robots to achieve predictive collision avoidance. These motion predictions can be obtained among robots by sharing their future planned trajectories with each other via communication. However, such communication may not be available nor reliable in practice. In this paper, we introduce a new model based on recurrent neural networks (RNNs) that is able to learn multi-robot motion behaviors from demonstrated trajectories generated using a centralized planner. The learned model can run efficiently online for each robot and provide interaction-aware trajectory predictions of its neighbors based on observations of their historical states. We then incorporate the trajectory prediction model into a decentralized model predictive control (MPC) framework for multi-robot collision-avoidance and compare it with a centralized planner. Simulation results show that our decentralized approach can achieve a level of performance comparable to the centralized planner while being communication-free and scalable to a large number of robots.**

I. Introduction

Historically, the use of robots has been mainly restricted to industrial areas under controlled conditions due to their high cost and associated safety hazard. Navigating in unknown and dynamic environments entails different challenges to those found in industrial settings, problems which were accentuated in the past by limited onboard computing power and battery technology. However, advancements in electronics and other engineering disciplines such as control theory, motion planning and artificial intelligence have opened new possibilities for commercial applications of mobile robots that were previously not possible. Well-known examples of these are autonomous cars, delivery drones and surveillance robots.

These novel applications will require the robot to move among other decision-making agents, such as humans or other robots. This circumstance makes the collision avoidance problem much more complicated than navigating in static environments, since in addition to avoiding obstacles, agents must also avoid colliding with each other. In the simplest case where all agents are robots under our control and in a known environment, it may be possible to implement a centralized approach and let a central coordinator find the best paths for all of them. However, such approaches are usually not implemented in practice because it is more computationally efficient for large teams of robots to distribute the path planning problem among the onboard computers. This approach is also more fault tolerant, as it removes the single point of failure in the form of the central coordinator. Even though the theoretical achievable performance of distributed approaches is similar to that of centralized ones, in practice communication often becomes unreliable. This leads to significant performance degradation with respect to the ideal scenario, and may even result in collisions among the robots. Furthermore, there are also applications for which communication is simply not possible because one or several agents are not under our control, as is the case when navigating among humans. These circumstances motivate the development of fully decentralized approaches, which rely solely on local sensor data in order to plan collision-free trajectories instead of using communication.

In this paper we propose a communication-free decentralized multi-robot local path planning approach with a collision-avoidance guarantee over a finite future horizon and a performance that is similar to its centralized counterpart. This method relies on a learned interaction model in order to predict how robots other than the controlled one will behave in the future and plan accordingly so as to avoid them. Our method is suitable for three-dimensional environments with an arbitrary number of robots with the same (possibly unstable and nonlinear) dynamics and an arbitrary number of obstacles, which may be static or dynamic.

The main contributions of our work are:

- A RNN-based interaction-aware trajectory prediction model that is capable of coping with an arbitrary number of dynamically complex agents and an arbitrary number of static or dynamic obstacles in three-dimensional spaces.
- A decentralized nonlinear MPC-based multi-robot motion planner that is able to account for interactions among the robots through the use of the previous trajectory prediction model.

We train the prediction neural network on data obtained with a centralized sequential version of the MPC-based motion planner, in which the planned trajectories for each robot are used as predictions. We evaluate our method on a team of quadrotors in a number of simulated scenarios. We compare the performance of our approach to three different versions of the used motion planner: centralized sequential, distributed, and decentralized using a constant velocity model model (CVM) for predictions. We show how

This work is supported by NWO Veni grant 15916.

The authors are with the Department of Cognitive Robotics, Delft University of Technology, 2628 CD, Delft, The Netherlands f.martinezclaramunt@student.tudelft.nl, {h.zhu;bruno.debrito;j.alonsomora}@tudelft.nl

our decentralized approach achieves a collision avoidance performance that is similar to its centralized counterpart.

II. Related Work

A. Multi-Robot Collision Avoidance

Multi-robot collision avoidance is a mature research topic that is still very actively researched. Traditional single robot rule-based collision avoidance methods such as the dynamic window approach [1] and artificial potential fields [2], [3] have been adapted in the past to work in the multi-robot case. These can be implemented in a decentralized manner, but they are extremely short-sighted due to their reactive nature and may lead to deadlocks. VO were also extended to work for decentralized multi-robot systems in [4] under the name of RVO. Further improvements to this framework led to the well known ORCA algorithm from [5], and its extension to non-holonomic robots in [6]. However, these methods ignore the robots' dynamics and only plan for one step ahead into the future.

There have also been a number of receding horizon optimization approaches that have been tried in the literature, which can be classified in centralized [7], [8], [9], [10], distributed [11], [12], [10] and decentralized (communication-free) [13], [10]. Centralized methods do not scale well to bigger teams of robots and the requirement of a central planner makes them unsuitable for most remote outdoor environments. As it has been already introduced, distributed methods often do not work as well in practice as they do in theory because of the tendency of wireless communication to be unreliable. Regarding the cited communication-free decentralized MPC-based methods, they use a CVM to predict the trajectories of robots other than the controlled one, which leads to a significant performance degradation with respect to their centralized MPC counterparts due to the poor prediction accuracy of the model. In our work we close this performance gap between the centralized and decentralized approaches by developing a much more accurate RNN-based prediction model which is trained on simulations using a centralized sequential MPC.

There are also many learning-based techniques that have been used to achieve decentralized multi-robot collision avoidance, such as interacting Gaussian processes [14], [15], [16] and reinforcement learning [17], [18], [19], [20]. However, they have only considered environments of up to two dimensions and they do not provide any collision avoidance guarantees. [21] presents a reinforcement learning-based method that works in three dimensions, but they only consider single robot static environments, and the trained policy does not guarantee collision avoidance. Our method, on the other hand, offers a collision avoidance guarantee over a finite future horizon through the use of an MPC algorithm for planning.

B. Motion Prediction

The topic of motion prediction has raised significant interest in recent years due to the need to account for human future intentions to achieve safe robot navigation in urban environments. One of the first motion prediction approaches that was introduced in the literature is the Social Force model [22]. This rule-based method models pedestrian behavior through the use of attractive and repulsive potentials. Even though it is commonly used as a baseline due to its simplicity, it requires manual parameter-tuning and its accuracy is far from that of the current state-of-the-art.

There have also been several notable attempts to use game theory to predict behavior of decision-making agents [23], [24], [25]. A more sophisticated approach is presented in [26], where game theory is combined with a tool from social psychology called SVO in order to quantify agents' degree of selfishness or altruism before predicting their behavior. However, these methods restrict themselves to very specific scenarios and to the best of our knowledge have only been applied to two dimensional spaces.

The class of approaches that have produced the best results so far are based on machine learning. Some of these include: dynamic Bayesian networks [27], random forests [28], inverse reinforcement learning [29], [30], [31], RNN [32], [33], variational autoencoders [34] and GAN [35], [36]. To the best of our knowledge, no prior paper has explored the topic of trajectory prediction with robots. As a consequence, none of these methods has been developed for 3D environments, which is necessary when dealing with aerial robots. Out of the cited methods, those which account for obstacles do so only for static ones through the use of 2D grid maps of the environment, which is processed by a **CNN!** (**CNN!**), an approach which cannot be straightforwardly extended to 3D. Another main difference between the pedestrian and robot prediction problems is that pedestrians tend to follow quasi-constant speed trajectories, except when they interact with other elements of the road, in which case they usually start and stop suddenly. Robots, on the other hand, tend to show smoother changes in velocity when they reach their goals or start to move towards new ones. Their behavior variability is also much smaller than with humans, since (usually) they all follow exactly the same policy and have theoretically equal dynamics. For these reasons, we develop a new RNN-based architecture that can better leverage the robots' dynamics and which can deal with an arbitrary number of other robots and static or dynamic 3D obstacles.

III. Preliminaries

A. Multi-Robot Collision Avoidance

We consider a team of n robots moving in a shared workspace $\mathcal{W} \subseteq \mathbb{R}^3$, where each robot $i \in \mathcal{I} = \{1, 2, \dots, n\} \subset \mathbb{N}$ is modeled as an enclosing sphere with radius r. The dynamics of all robots are the same, and they are described by a discrete-time equation as follows,

$$\mathbf{x}_i^{k+1} = \mathbf{f}(\mathbf{x}_i^k, \mathbf{u}_i^k), \quad \mathbf{x}_i^0 = \mathbf{x}_i(0), \tag{1}$$

where $\mathbf{x}_i^k \in \mathbb{X} \subset \mathbb{R}^{n_x}$ denotes the state of the robot, typically including its position $\mathbf{p}_i^k$ and velocity $\mathbf{v}_i^k$, and $\mathbf{u}_i^k \in \mathbb{U} \subset \mathbb{R}^{n_u}$ the control inputs at time k. $\mathbb{X}$ and $\mathbb{U}$ are the admissible state space and control space, respectively. $\mathbf{x}_i(0)$ is the initial state of robot i. Any pair of robots i and j from the group

are mutually collision-free if $\left\|\mathbf{p}_i^k - \mathbf{p}_j^k\right\| \geq 2r, \forall i \neq j \in \mathcal{I}, \forall k = 0, 1, \ldots$. Each robot has a given goal location $\mathbf{g}_i$, which generally comes from some high-level path planner or is specified by the user.

The objective of multi-robot collision avoidance is to compute a local motion $\mathbf{u}_i^k$ for each robot in the group, that respects its dynamic constraints, makes progress towards its goal location $\mathbf{g}_i$ and is collision-free with other robots in the team for a short time horizon.

B. Obstacle Model

For each obstacle $o \in \mathcal{I}_o = \{1, 2, \ldots, n_o\} \subset \mathbb{N}$ at position $\mathbf{p}_o \in \mathbb{R}^3$, we model it as an upright non-rotating enclosing *ellipsoid* centered at $\mathbf{p}_o$ with semi-principal axes (a, b, c). These are assumed to either be static or follow a constant velocity model (CVM), as in [10]. In practice, they may be used to model stationary objects or other agents for which an accurate prediction model is not available. For the remainder of this work, we will assume all of the obstacles to be of the same size, as they will represent humans moving among the robots.

C. Collision Condition

The collision condition of robot i with respect to robot j at time k is defined as

$$C_{ij}^k \coloneqq \{\mathbf{x}_i^k \,|\, \left\|\mathbf{p}_i^k - \mathbf{p}_j^k\right\| \leq 2r\} \,. \tag{2}$$

Collision checking between a robot i and an obstacle o requires calculating the minimum distance between a sphere and an ellipsoid, which cannot be performed in closed form [37]. To this end, we approximate the obstacle with an enlarged ellipsoid and check if the robot's position is inside it. The collision condition thus becomes

$$C_{io}^k \coloneqq \{\mathbf{x}_i^k \,|\, \left\|\mathbf{p}_i^k - \mathbf{p}_o^k\right\|_\Omega \leq 1\} \,, \tag{3}$$

where $\Omega = \text{diag}(1/(a+r+\delta)^2, 1/(b+r+\delta)^2, 1/(c+r+\delta)^2)$, and δ is the minimum enlarging coefficient as defined in [38].

D. Centralized Sequential Model Predictive Control

The multi-robot collision avoidance problem is formulated as a receding horizon constrained optimization problem. For each robot $i \in \mathcal{I}$, a discrete-time constrained optimization formulation with N time steps and planning horizon $\tau = N\Delta t$, where Δt is the sampling time, is derived as follows,

$$\begin{aligned}
\min_{\mathbf{x}_i^{1:N}, \mathbf{u}_i^{0:N-1}} \quad & \sum_{k=0}^{N-1} J_i^k(\mathbf{x}_i^k, \mathbf{u}_i^k) + J_i^N(\mathbf{x}_i^N, \mathbf{g}_i) \\
\text{s.t.} \quad & \mathbf{x}_i^0 = \mathbf{x}_i(0), \\
& \mathbf{x}_i^k = \mathbf{f}(\mathbf{x}_i^{k-1}, \mathbf{u}_i^{k-1}), \\
& \left\|\mathbf{p}_i^k - \mathbf{p}_j^k\right\| \geq 2r, \\
& \left\|\mathbf{p}_i^k - \mathbf{p}_o^k\right\|_\Omega \geq 1, \\
& \mathbf{u}_i^{k-1} \in \mathbb{U}, \quad \mathbf{x}_i^k \in \mathbb{X}, \\
& \forall j \neq i \in \mathcal{I};\, \forall o \in \mathcal{I}_o;\, \forall k \in \{1, \ldots, N\}.
\end{aligned} \tag{4}$$

At every time step, the problem for each robot $i \in \mathcal{I}$ will be solved sequentially by a central planner in a receding-horizon fashion. Each other robot $j \neq i \in \mathcal{I}$ will be considered as a dynamic obstacle the N future positions of which will be determined by the latest available solution to its respective MPC problem. If the latest available optimal trajectory for a robot does not include a position for the last time step in the planning horizon N, a CVM will be used to estimate it. Once all n sub-problems have been solved, the first step of each robot's optimal control action sequence is executed.

The cost function of the MPC problem has three different terms, which are:

- a goal navigation cost:

$$J_i^N(\mathbf{x}_i^N, \mathbf{g}_i) = \left\|\mathbf{p}_i^N - \mathbf{g}_i\right\|_{Q_\mathbf{g}}, \tag{5}$$

  where $\mathbf{g}_i$ is the goal position of robot i and $Q_\mathbf{g}$ is the goal navigation weighting matrix.
- a control input cost:

$$J_i^k(\mathbf{u}_i^k) = \left\|\mathbf{u}_i^k\right\|_{Q_\mathbf{u}}, \tag{6}$$

  where $Q_\mathbf{u}$ is the control input weighting matrix.
- a collision potential cost based on the logistic function:

$$J_i^k(\mathbf{x}_i^k) = \frac{Q_c}{1 + \exp(\lambda_c(d_{ij}^k - d_{ij}^{\text{safe}}))}, \tag{7}$$

  where d_{ij}^k is the distance between robot i and robot or obstacle j at time k, Q_c is the collision potential weighting coefficient, λ_c is a parameter defining the smoothness of the cost function and d_{ij}^{safe} is a safety distance. It must be noted that the distance d_{ij}^k is computed differently depending on whether it is between two robots or between a robot and an obstacle:

$$d_{ij}^k = \begin{cases} \left\|\mathbf{p}_i^k - \mathbf{p}_j^k\right\|, & \text{if } j \in \{\mathcal{I} \setminus i\} \\ \left\|\mathbf{p}_i^k - \mathbf{p}_j^k\right\|_\Omega, & \text{if } j \in \mathcal{I}_o \end{cases}. \tag{8}$$

E. Distributed Model Predictive Control

The previous centralized sequential formulation can be easily distributed among the robots' onboard computers by letting them share the solutions to their own MPC problems with the rest of the team. These solutions will then be used as predictions of the other robots' future positions.

F. Decentralized Model Predictive Control

Extending the approach to work in a communication-free decentralized setting requires the usage of an alternative method for each robot $i \in \mathcal{I}$ to predict the future positions of each robot $j \neq i \in \mathcal{I}$ for the planning horizon N. A simple CVM model may be used as in [13], [10], but we will propose a machine learning-based model for better accuracy.

IV. Interaction-Aware Trajectory Prediction

Our trajectory prediction problem involves estimating the future positions of each robot $q \in \{\mathcal{I} \setminus c\}$ over a future horizon of N_{pred} steps, where robot q is known as the query robot and robot c is the controlled one (also known as the ego robot). We will use the following notation to denote the future sequence of positions for robot q starting at time $t+1$:

$$p_{q,\text{pred}}^{t+1:t+N_{\text{pred}}} \coloneqq \left(p_{q,\text{pred}}^{t+1}, p_{q,\text{pred}}^{t+2}, ..., p_{q,\text{pred}}^{t+N_{\text{pred}}}\right), \tag{9}$$

with the ground truth being identified as $p_{q,\text{target}}^{t+1:t+N_{\text{pred}}}$. However, in order to avoid overfitting based on the robots' absolute positions when training our neural network, we will instead work with sequences of velocity predictions $v_{q,\text{pred}}^{t+1:t+N_{\text{pred}}}$, and will numerically integrate them afterwards starting from the current position $p_{q,\text{pred}}^{t}$.

A. Formulation

We formulate the trajectory prediction problem as finding a mapping

$$\left(v_q^{t-N_{\text{past}}+1:t}, \mathcal{X}_{r,q}^{t-N_{\text{past}}+1:t}, \mathcal{X}_{o,q}^{t}\right) \mapsto v_{q,\text{pred}}^{t+1:t+N_{\text{pred}}}, \quad (10)$$

where:

- $v_q^{t-N_{\text{past}}+1:t}$ is the past sequence of N_{past} observed velocities of robot q,
- $\mathcal{X}_{r,q}^{t-N_{\text{past}}+1:t}$ is the set of sequences of past relative states of the other robots with respect to robot q:

$$\mathcal{X}_{r,q}^{t-N_{\text{past}}+1:t} \coloneqq \left\{ x_{iq}^{t-N_{\text{past}}+1:t} \mid i \in \{\mathcal{I} \setminus q\} \right\}, \quad (11)$$

with

$$x_{iq}^{t-N_{\text{past}}+1:t} = \left(x_{iq}^{t-N_{\text{past}}+1}, x_{iq}^{t-N_{\text{past}}+2}, ..., x_{iq}^{t}\right) \quad (12)$$

and

$$x_{iq}^{t} \coloneqq x_i^t - x_q^t = \begin{bmatrix} p_i^t - p_q^t \\ v_i^t - v_q^t \end{bmatrix}, \quad (13)$$

- and $\mathcal{X}_{o,q}^{t}$ is the set of relative current states of the obstacles with respect to robot q:

$$\mathcal{X}_{o,q}^{t} \coloneqq \left\{ x_{iq}^{t} \mid i \in \mathcal{I}_o \right\}. \quad (14)$$

This mapping will be found such that we minimize the MSE[1] between the predicted future horizon of velocities $v_{q,\text{pred}}^{t+1:t+N_{\text{pred}}}$ and the actual one $v_{q,\text{target}}^{t+1:t+N_{\text{pred}}}$:

$$\text{MSE}\left(\left\{ v_{q,\text{pred}}^{t+1:t+N_{\text{pred}}} \,\middle|\, t \in \mathcal{T} \right\}\right) = \frac{1}{|\mathcal{T}| \cdot N_{\text{pred}} \cdot \dim(\mathcal{W})} \sum_{t \in \mathcal{T}} \sum_{k=1}^{N_{\text{pred}}} \left\| (v_{q,\text{target}}^{t+k} - v_{q,\text{pred}}^{t+k}) \right\|^2, \quad (15)$$

where $\mathcal{W}$ is the workspace the robot lives in and $\mathcal{T}$ is the set of time steps for which a trajectory prediction has been obtained.

B. Network Architecture

We propose the network architecture shown in Figure 1 to approximate the optimal mapping in Equation (10). It consists of three different modules: the query robot state encoder, the environment encoder and the decoder. All layers in the network feature $L2$ regularization for the weights and biases and hyperbolic tangent activation functions, except for the output one which is linear. Besides the recurrent decoder, which has 128 neurons, and the output layer, which has as many as the dimension of the workspace, all layers include 64 neurons. The recurrent layers are of the LSTM type, since these prevent the vanishing gradient problem present in regular RNN units [39], [40], and are able to learn time dependencies over longer periods of time. Thanks to the use of recurrent layers, the network can consider an arbitrary number of input and output sequence lengths, which do not need to be equal. Next, we will discuss the details of each of the three architecture modules.

1) Query robot state encoder: It consists of a recurrent layer that produces a flat encoding z_q^t from the history of velocities of the query robot $v_q^{(t-N_{\text{past}}+1:t)}$. This layer learns a dynamical model of the query robot, so that the network can leverage it to obtain better predictions.

2) Environment encoder module: It includes $n-1$ parallel recurrent layers with shared weights to encode the set of sequences of past relative states of the other robots with respect to the query robot $\mathcal{X}_{r,q}^{t-N_{\text{past}}+1:t}$ into set $\mathcal{Z}_{r,q}^{t}$, and n_o parallel dense layers with shared weights which encode the set of relative current states of the obstacles with respect to the query robot $\mathcal{X}_{o,q}^{t}$ into set $\mathcal{Z}_{o,q}^{t}$. The encodings from both of these sets, which are all made to have the same length, are stacked together and then a global max pooling operation is executed along the new data axis. Thus, this module can capture the interaction of the query robot with an arbitrary number of other robots and obstacles in the environment and encode it into a single flat vector z_e^t. This framework also makes it possible to account for different types of agents and obstacles by training their own set of encoders and stacking them with the rest of intermediate encodings.

3) Decoder module: It takes in the concatenation of z_q^t with z_e^t and passes it through a recurrent decoder which will generate a sequence of length equal to the desired prediction horizon. Then, a fully connected layer is included to increase the overall model complexity of the network. Finally, a fully connected layer is included with as many neurons as the dimension of the workspace (3 in this case), in order to generate an output of the desired target size.

C. Dataset

To generate the data to train our network and later validate our proposed path planner, we use the MATLAB-based simulator from [41], which considers the dynamics of a Parrot Bebop 2 quadrotor and uses Forces Pro [42] as the MPC optimization solver. For the training data, we simulate 10^5 steps with 10 quadrotors navigating among 10 dynamic obstacles, with a planning horizon of 20 and a sampling rate of 50 ms. Robots' enclosing spheres have a radius of 0.3 m and the obstacles' ellipsoids have semi-axes lengths of (0.4, 0.4, 0.9) m, so that they approximate the shape of a human. In the simulation, each robot will move towards its own randomly generated goal, which will change dynamically once it gets close enough according to some randomly distributed distance and velocity thresholds, to create a variety of approach behaviors. Obstacles will move at constant speeds with a direction vector that is larger in the X and Y directions than in the Z one and their positions will be randomly reset within the considered $10 \times 10 \times 3$ m

[1]In practice the cost function will also include regularization terms for the neural network's weights and biases, but these are not written here for conciseness.

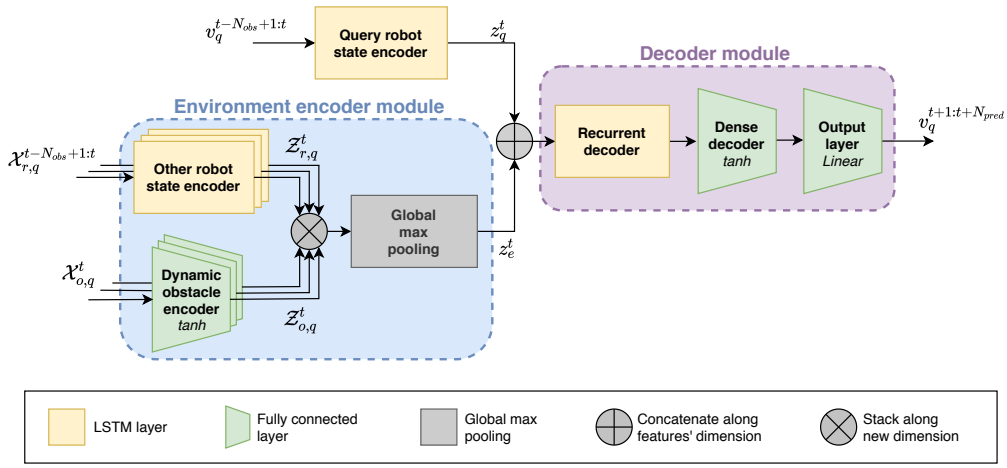


Fig. 1: Network architecture

environment once they get out of bounds. Two separate smaller datasets have also been generated for validation and testing.

D. Evaluation Metrics

Although we can use the cost function defined in Equation (15) to compare the performance of different prediction models, this metric is not informative enough about the trajectory prediction accuracy. This is because, in addition to being based on velocities instead of positions, it adds up the error over the whole prediction horizon. Therefore, we will instead integrate the sequence of predicted velocities in order to obtain position predictions and will then compute the average displacement error (based on the euclidean distance or $L2$ norm) at each of the future steps in the horizon.

E. Implementation Details

We have implemented the described network architecture using TensorFlow 2 and trained it on a laptop with an Intel Core i7 7850H processor, 16 GB of RAM and an NVIDIA GTX 1060 GPU with 6 GB of VRAM, taking an hour of training on the GPU, but already producing accurate predictions at around 30 minutes. On this same computer it takes around 30 ms to perform inference with 12 robots and 12 obstacles. The network has been trained with past and prediction horizons of length 20, with the data from the simulator having a sampling rate of 50 ms. To improve learning, both input and output data have been scaled to be in the [-1,1] range.

V. Results

A. Trajectory prediction

We test our prediction method on datasets collected with the centralized version of the MPC-based planner which have not been used for training nor validation. The scenarios that will be considered are an open environment with 4 and 10 quadrotors, and the same environments but with a number of dynamic obstacles equal to the number of robots. The baselines to which we compare our RNN-based model are a CVM which assumes that robots keep the speed measured at the last available step, and the planned trajectories computed with the centralized sequential MPC that was used to generate the data.

A summary of the performance results is shown in Figure 2, and two screenshots of sample predictions using our model in the setting with 10 robots and 10 obstacles are displayed in Figure 3. We can see how the performance of the three methods stays consistent across the four scenarios, with the only meaningful differences being our RNN-based model performing slightly worse in the least dense environment, and all methods showing larger than usual variance in the 10 quadrotors plus 10 obstacles. The second phenomenon is expected due to the higher environment clutter, but the first one can be explained by our network overrelying on the environment encoder module explained in Section IV-B. We can also observe how our method is on average almost as accurate as the optimal trajectories planned by the centralized sequential MPC, even though we do not feed the neural network the robots' current goal positions, unlike the MPC. When compared with our model, it is clear that the CVM is not an accurate predictor of the robots' future trajectories.

B. Motion planning

We compare the performance of our decentralized path planning method to the centralized sequential MPC-based planner that was used to generate the training data. as well as the distributed version explained in Section III-E. Furthermore, we will also use a decentralized version of the path planner which uses a CVM as the prediction model, to analyze whether the more accurate trajectory forecasts of our RNN-based model lead to significantly better planning performance or not. For our tests, we will focus exclusively on obstacle-free scenarios, since the way in which obstacles are accounted for is equivalent within all considered versions of the MPC: they are assumed to keep a constant speed, which is indeed how they behave.

We will analyze 4 different types of test scenarios, each of them with 50 unique sets of pairs of start and goal positions

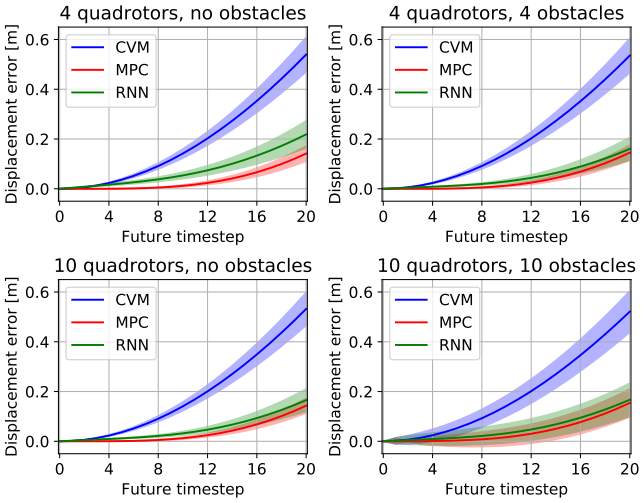


Fig. 2: Plot showing the mean $L2$ prediction error as a function of the prediction step in four different settings with a constant velocity model (CVM), the MPC planner that was used to generate the dataset, and our proposed RNN-based model. The solid lines represent the average errors and the filled patches around them are 30% of the standard deviation. The sampling period is 50 ms, so the whole prediction horizon of 20 steps is equivalent to 1 second.

for six quadrotors. The first of these consists of a *symmetric swap*, in which the robots start from the vertices of a virtual regular hexagon that is parallel to the ground plane and they need to exchange positions with the robot in the opposing vertex. Each of the different cases within this group will feature different hexagon sizes and rotation along the Z axis. The second class of tests consists of *asymmetric swaps*, which differ from the previous ones in that hexagons are no longer regular, thus leading to more challenging collision-avoidance problems. All the start and goal positions are still placed at the same height, and the shape and size of the hexagon changes for the different sets. Next, we will require the robots to perform *pair-wise swaps*, in which the robots are placed in random start positions and assigned a pair with which they will need to exchange positions. This scenario is less challenging because the straight lines that unite start to goal positions are very unlikely to intersect with each other, so robots no longer attempt to go through the center of the environment in order to minimize their trajectory lengths. The last type of test will simply involve the robots moving from *random* starting positions to random goals.

To numerically evaluate the performance of the different motion planners in these experiments we consider a wide range of metrics. We compute the minimum distance between any two quadrotors during the whole simulation, and we also count how many of the 50 sets of trajectories includes at least one collision between a pair of robots. We also calculate the trajectory lengths and durations, and the average linear velocities throughout each scenario. A summary of the obtained results can be seen in Table I. For a qualitative performance analysis, we show sample trajectory trails for a specific set of start to goal trajectories from the asymmetric swap scenario in Figure 4.

From the metrics shown in Table I, we can see how using our RNN-based prediction model significantly enhances performance of the decentralized MPC-based planner when compared to using a CVM, leading to consistently lower trajectory lengths and durations on average, and greatly diminishing the likelihood of collisions. Furthermore, we achieve average trajectory lengths, trajectory durations and speeds comparable to the centralized planner.

VI. Conclusion

In this paper, we have presented a decentralized multi-robot MPC-based path planner which accounts for robots' interactions with obstacles and with other robots in the team through the use of a RNN-based trajectory prediction model that was trained on data collected with a centralized version of the path planner. We have shown how our prediction model is suitable for three-dimensional spaces and generalizes well to simulated scenarios with different numbers of robots and obstacles. We have also demonstrated that the proposed decentralized planning framework using our prediction model achieves a similar performance to the centralized counterpart that was used to train the RNN, massively reducing the number of collisions with respect to using the same framework but considering a CVM for the predictions.

References

[1] J. Bruce and M. Veloso, "Real-Time Multi-Robot Motion Planning with Safe Dynamics," in *Multi-Robot Systems. From Swarms to Intelligent Automata*, vol. 3, 2005, pp. 159–170.

[2] G. Roussos and K. J. Kyriakopoulos, "Completely Decentralised Navigation of Multiple Unicycle Agents with Prioritisation and Fault Tolerance," in *Proceedings of the IEEE Conference on Decision and Control (CDC)*. IEEE, 2010, pp. 1372–1377.

[3] ——, "Decentralized and Prioritized Navigation and Collision Avoidance for Multiple Mobile Robots," in *Distributed Autonomous Robotic Systems*. Springer, 2012, pp. 189–202.

[4] J. van den Berg, M. Lin, and D. Manocha, "Reciprocal Velocity Obstacles for Real-Time Multi-Agent Navigation," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2008, pp. 1928–1935.

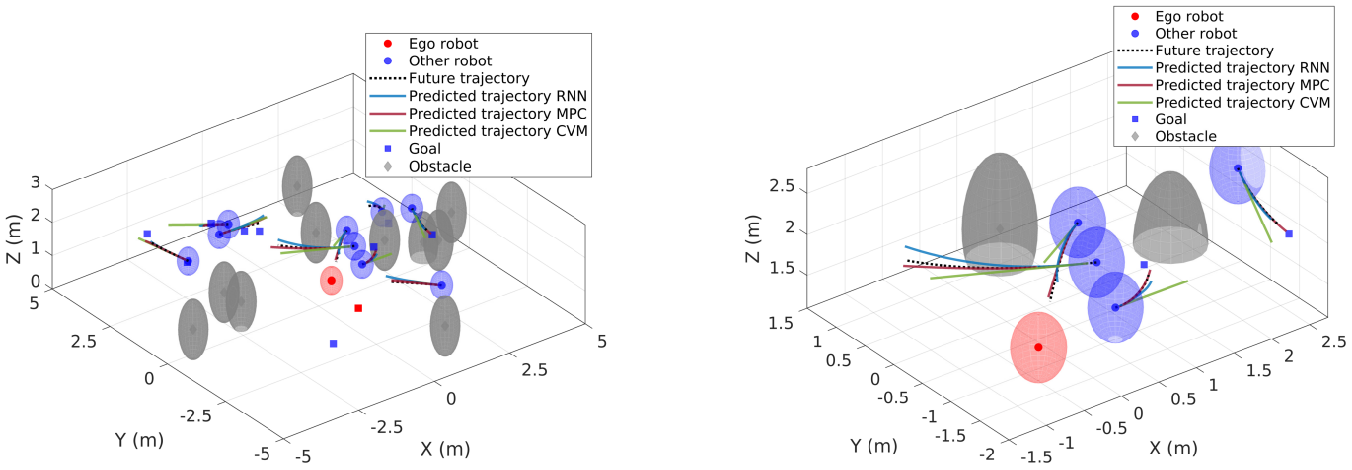
[5] J. van den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-Body Collision Avoidance," in *Proceedings of the International Symposium on Robotics Research (ISRR)*, 2011, pp. 3–19.

[6] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P. Beardsley, and R. Y. Siegwart, "Optimal Reciprocal Collision Avoidance for Multiple Non-Holonomic Robots," in *Distributed Autonomous Robotic Systems*. Springer, 2013, pp. 203–216.

[7] T. Schouwenaars, B. De Moor, E. Feron, and J. How, "Mixed Integer Programming for Multi-Vehicle Path Planning," in *Proceedings of the European Control Conference (ECC)*, no. 3. IEEE, 2001, pp. 2603–2608.

[8] C. Leung, S. Huang, N. Kwok, and G. Dissanayake, "Planning Under Uncertainty Using Model Predictive Control for Information Gathering," *Robotics and Autonomous Systems*, vol. 54, no. 11, pp. 898–910, 2006.

[9] G. P. Roussos, G. Chaloulos, K. J. Kyriakopoulos, and J. Lygeros, "Control of Multiple Non-Holonomic Air Vehicles Under Wind Uncertainty Using Model Predictive Control and Decentralized Navigation Functions," in *Proceedings of the IEEE Conference on Decision and Control (CDC)*. IEEE, 2008, pp. 1225–1230.



(a) Complete view of the environment

(b) Zoomed view of the environment around the ego robot

Fig. 3: Screenshots of sample predictions of our RNN based model compared to predictions from a CVM and the planned trajectories from the centralized sequential MPC in an environment with ten robots and ten dynamic obstacles.

Scenario	Motion planner	Minimum distance (m)	Num. of collisions	Trajectory length (m) Min.	Max.	Average	Trajectory time (s) Min.	Max.	Average	Average speed (m/s)
Symmetric swap	Centralized	0.80	0	5.46	8.96	7.22 ± 0.85	5.15	6.05	5.62 ± 0.24	1.27
	Distributed	0.80	0	5.52	8.73	7.26 ± 0.84	4.75	6.40	5.51 ± 0.35	1.31
	Decen. (CVM)	0.32	4 (!)	5.43	9.55	7.45 ± 0.88	4.95	7.30	6.02 ± 0.55	1.23
	Decen. (RNN)	**0.75**	**0**	**5.41**	**10.60**	$\mathbf{7.35 \pm 0.91}$	**4.75**	**9.7**	$\mathbf{5.59 \pm 0.40}$	**1.30**
Asymmetric swap	Centralized	0.80	0	5.08	9.02	6.77 ± 0.80	4.65	5.85	5.17 ± 0.30	1.30
	Distributed	0.80	0	4.99	8.84	6.82 ± 0.84	4.65	6.25	5.12 ± 0.33	1.32
	Decen. (CVM)	0.16	15 (!)	5.32	18.06	7.76 ± 1.89	5.05	7.30	5.96 ± 0.51	1.28
	Decen. (RNN)	**0.37**	**3 (!)**	**5.16**	**12.60**	$\mathbf{7.14 \pm 1.10}$	**4.80**	**6.85**	$\mathbf{5.48 \pm 0.42}$	**1.29**
Pair-wise swap	Centralized	0.80	0	1.64	9.92	5.10 ± 1.87	3.50	5.85	4.76 ± 0.44	1.06
	Distributed	0.80	0	1.82	9.75	5.10 ± 1.86	3.60	5.85	4.72 ± 0.43	1.07
	Decen. (CVM)	0.34	3 (!)	1.74	17.42	5.54 ± 2.53	3.60	5.75	5.00 ± 0.65	1.06
	Decen. (RNN)	**0.79**	**0**	**1.70**	**9.94**	$\mathbf{4.94 \pm 2.02}$	**3.40**	**5.95**	$\mathbf{4.83 \pm 0.45}$	**1.01**
Random	Centralized	0.80	0	0.39	8.63	4.66 ± 1.94	3.50	5.50	7.72 ± 0.40	0.98
	Distributed	0.8	0	0.39	8.73	4.67 ± 1.93	3.50	5.40	4.70 ± 0.40	0.98
	Decen. (CVM)	0.76	0	0.39	9.53	4.82 ± 2.06	3.60	6.35	4.89 ± 0.57	0.98
	Decen. (RNN)	**0.78**	**0**	**0.39**	**9.19**	$\mathbf{4.36 \pm 2.11}$	**3.85**	**5.95**	$\mathbf{4.76 \pm 0.43}$	**0.91**

TABLE I: Comparison of the performance of the four considered MPC-based motion planner variations (centralized, distributed, decentralized with CVM and decentralized with our RNN-based model) across the four different types of scenarios that have been tested (symmetric swap, asymmetric swap, pair-wise swap and random goals).

[10] H. Zhu and J. Alonso-Mora, "Chance-constrained collision avoidance for mavs in dynamic environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 776–783, 2019.

[11] D. Shim, H. Kim, and S. Sastry, "Decentralized Reflective Model Predictive Control of Multiple Flying Robots in Dynamic Environment," in *Proceedings of the IEEE Conference on Decision and Control (CDC)*, 2003, pp. 3621–3626.

[12] A. Richards and J. How, "Decentralized Model Predictive Control of Cooperating UAVs," in *Proceedings of the IEEE Conference on Decision and Control (CDC)*. IEEE, 2004, pp. 4286–4291.

[13] M. Kamel, J. Alonso-Mora, R. Siegwart, and J. Nieto, "Robust Collision Avoidance for Multiple Micro Aerial Vehicles Using Nonlinear Model Predictive Control," *IEEE International Conference on Intelligent Robots and Systems*, vol. 2017-Septe, pp. 236–243, 2017.

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

[15] P. Trautman, J. Ma, R. M. Murray, and A. Krause, "Robot Navigation in Dense Human Crowds: The Case for Cooperation," in *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 2153–2160.

[16] ——, "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.

[17] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized Non-Communicating Multiagent Collision Avoidance with Deep Reinforcement Learning," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 285–292.

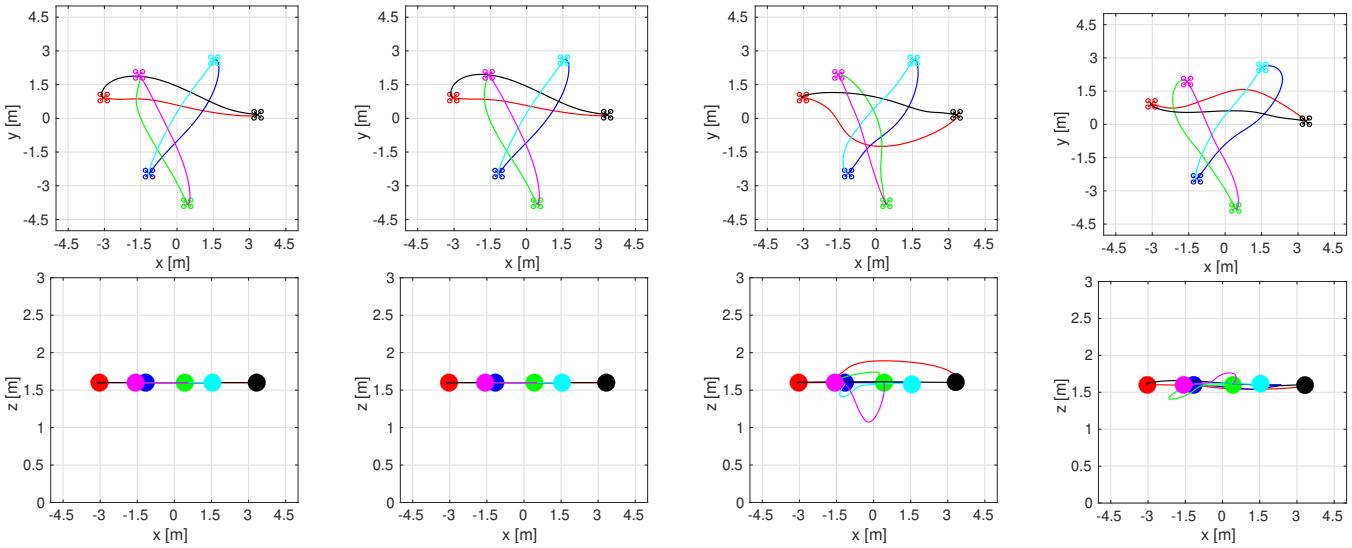
[18] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially Aware Motion Planning with Deep Reinforcement Learning," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 1343–1350.

[19] M. Everett, Y. F. Chen, and J. P. How, "Motion Planning among Dynamic, Decision-Making Agents with Deep Reinforcement Learning," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3052–3059.

[20] P. Long, T. Fanl, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards Optimally Decentralized Multi-Robot Collision Avoidance via Deep Reinforcement Learning," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 6252–6259.

[21] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, "Learning Deep Control Policies for Autonomous Aerial Vehicles with MPC-Guided Policy Search," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 528–535.

[22] D. Helbing and P. Molnár, "Social Force Model for Pedestrian Dynamics," *Physical Review E*, vol. 51, no. 5, pp. 4282–4286, 1995.



(a) Centralized sequential (b) Distributed (c) Decent. with CVM (d) Decent. with RNN model

Fig. 4: Simulation results of six quadrotors exchanging positions in the asymmetric swap scenario. Solid lines represent the trajectories executed by the quadrotors. The upper and lower plots show the top view (X-Y) and side view(X-Z), respectively.

[23] A. Martin, "Interactive Motion Prediction using Game Theory," Ph.D. dissertation, University of Padua, 2013.

[24] A. Turnwald, D. Althoff, D. Wollherr, and M. Buss, "Understanding Human Avoidance Behavior: Interaction-Aware Decision Making Based on Game Theory," *International Journal of Social Robotics*, vol. 8, no. 2, pp. 331–351, 2016.

[25] D. W. Oyler, Y. Yildiz, A. R. Girard, N. I. Li, and I. V. Kolmanovsky, "A Game Theoretical Model of Traffic with Multiple Interacting Drivers for Use in Autonomous Vehicle Development," in *Proceedings of the American Control Conference (ACC)*, 2016, pp. 1705–1710.

[26] W. Schwarting, A. Pierson, J. Alonso-Mora, S. Karaman, and D. Rus, "Social Behavior for Autonomous Vehicles," in *Proceedings of the National Academy of Sciences*, vol. 116, no. 50, 2019, pp. 24 972–24 978.

[27] T. Gindele, S. Brechtel, and R. Dillmann, "Learning Context Sensitive Behavior Models from Observations for Predicting Traffic Situations," in *Proceedings of the International IEEE Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2013, pp. 1764–1771.

[28] L. Leal-Taixé, M. Fenzi, A. Kuznetsova, B. Rosenhahn, and S. Savarese, "Learning an Image-Based Motion Context for Multiple People Tracking," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2014, pp. 3542–3549.

[29] M. Kuderer, S. Gulati, and W. Burgard, "Learning Driving Styles for Autonomous Vehicles from Demonstration," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 2641–2646.

[30] D. Sadigh, S. Sastry, S. A. Seshia, and A. D. Dragan, "Planning for Autonomous Cars that Leverage Effects on Human Actions," in *Robotics: Science and Systems*, vol. 2, 2016.

[31] D. Sadigh, S. S. Sastry, S. A. Seshia, and A. Dragan, "Information Gathering Actions Over Human Internal State," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 66–73.

[32] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social LSTM: Human trajectory prediction in crowded spaces," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 2016, pp. 961–971.

[33] A. Vemula, K. Muelling, and J. Oh, "Social Attention: Modeling Attention in Human Crowds," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 4601–4607.

[34] N. Lee, W. Choi, P. Vernaza, C. B. Choy, P. H. Torr, and M. Chandraker, "DESIRE: Distant Future Prediction in Dynamic Scenes with Interacting Agents," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 2165–2174.

[35] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi, "Social GAN: Socially Acceptable Trajectories with Generative Adversarial Networks," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2018, pp. 2255–2264.

[36] A. Sadeghian, V. Kosaraju, A. Sadeghian, N. Hirose, S. H. Rezatofighi, and S. Savarese, "SoPhie: An Attentive GAN for Predicting Paths Compliant to Social and Physical Constraints," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 1349–1358.

[37] A. Y. Uteshev and M. V. Goncharova, "Point-to-Ellipse and Point-to-Ellipsoid Distance Equation Analysis," *Journal of Computational and Applied Mathematics*, vol. 328, pp. 232–251, 2018.

[38] B. Brito, B. Floor, L. Ferranti, and J. Alonso-Mora, "Model predictive contouring control for collision avoidance in unstructured dynamic environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4459–4466, 2019.

[39] S. Hochreiter and J. Urgen Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[40] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to Forget: Continual Prediction with LSTM," *Neural Computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[41] H. Zhu and J. Alonso-Mora, "Chance-Constrained Collision Avoidance for MAVs in Dynamic Environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 776–783, 2019.

[42] A. Domahidi and J. Jerez, "Forces professional," *embotech GmbH (http://embotech. com/FORCES-Pro)*, 2014.

Bibliography

[1] J. Bruce and M. Veloso, "Real-Time Multi-Robot Motion Planning with Safe Dynamics," in *Multi-Robot Systems. From Swarms to Intelligent Automata*, vol. 3, pp. 159–170, 2005.

[2] G. Roussos and K. J. Kyriakopoulos, "Completely Decentralised Navigation of Multiple Unicycle Agents with Prioritisation and Fault Tolerance," in *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pp. 1372–1377, IEEE, 2010.

[3] G. Roussos and K. J. Kyriakopoulos, "Decentralized and Prioritized Navigation and Collision Avoidance for Multiple Mobile Robots," in *Distributed Autonomous Robotic Systems*, pp. 189–202, Springer, 2012.

[4] J. van den Berg, M. Lin, and D. Manocha, "Reciprocal Velocity Obstacles for Real-Time Multi-Agent Navigation," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1928–1935, IEEE, 2008.

[5] J. van den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-Body Collision Avoidance," in *Proceedings of the International Symposium on Robotics Research (ISRR)*, pp. 3–19, 2011.

[6] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P. Beardsley, and R. Y. Siegwart, "Optimal Reciprocal Collision Avoidance for Multiple Non-Holonomic Robots," in *Distributed Autonomous Robotic Systems*, pp. 203–216, Springer, 2013.

[7] T. Schouwenaars, B. De Moor, E. Feron, and J. How, "Mixed Integer Programming for Multi-Vehicle Path Planning," in *Proceedings of the European Control Conference (ECC)*, no. 3, pp. 2603–2608, IEEE, 2001.

[8] C. Leung, S. Huang, N. Kwok, and G. Dissanayake, "Planning Under Uncertainty Using Model Predictive Control for Information Gathering," *Robotics and Autonomous Systems*, vol. 54, no. 11, pp. 898–910, 2006.

[9] G. P. Roussos, G. Chaloulos, K. J. Kyriakopoulos, and J. Lygeros, "Control of Multiple Non-Holonomic Air Vehicles Under Wind Uncertainty Using Model Predictive Control

and Decentralized Navigation Functions," in *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pp. 1225–1230, IEEE, 2008.

[10] H. Zhu and J. Alonso-Mora, "Chance-Constrained Collision Avoidance for MAVs in Dynamic Environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 776–783, 2019.

[11] D. Shim, H. Kim, and S. Sastry, "Decentralized Reflective Model Predictive Control of Multiple Flying Robots in Dynamic Environment," in *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pp. 3621–3626, 2003.

[12] A. Richards and J. How, "Decentralized Model Predictive Control of Cooperating UAVs," in *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pp. 4286–4291, IEEE, 2004.

[13] M. Kamel, J. Alonso-Mora, R. Siegwart, and J. Nieto, "Robust Collision Avoidance for Multiple Micro Aerial Vehicles Using Nonlinear Model Predictive Control," *IEEE International Conference on Intelligent Robots and Systems*, vol. 2017-Septe, pp. 236–243, 2017.

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

[15] P. Trautman, J. Ma, R. M. Murray, and A. Krause, "Robot Navigation in Dense Human Crowds: The Case for Cooperation," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2153–2160, IEEE, 2013.

[16] 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.

[17] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized Non-Communicating Multiagent Collision Avoidance with Deep Reinforcement Learning," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 285–292, 2017.

[18] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially Aware Motion Planning with Deep Reinforcement Learning," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1343–1350, IEEE, 2017.

[19] M. Everett, Y. F. Chen, and J. P. How, "Motion Planning among Dynamic, Decision-Making Agents with Deep Reinforcement Learning," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 3052–3059, IEEE, 2018.

[20] P. Long, T. Fanl, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards Optimally Decentralized Multi-Robot Collision Avoidance via Deep Reinforcement Learning," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6252–6259, IEEE, 2018.

[21] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, "Learning Deep Control Policies for Autonomous Aerial Vehicles with MPC-Guided Policy Search," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 528–535, 2016.

[22] D. Helbing and P. Molnár, "Social Force Model for Pedestrian Dynamics," *Physical Review E*, vol. 51, no. 5, pp. 4282–4286, 1995.

[23] A. Martin, *Interactive Motion Prediction using Game Theory*. PhD thesis, University of Padua, 2013.

[24] A. Turnwald, D. Althoff, D. Wollherr, and M. Buss, "Understanding Human Avoidance Behavior: Interaction-Aware Decision Making Based on Game Theory," *International Journal of Social Robotics*, vol. 8, no. 2, pp. 331–351, 2016.

[25] D. W. Oyler, Y. Yildiz, A. R. Girard, N. I. Li, and I. V. Kolmanovsky, "A Game Theoretical Model of Traffic with Multiple Interacting Drivers for Use in Autonomous Vehicle Development," in *Proceedings of the American Control Conference (ACC)*, pp. 1705–1710, 2016.

[26] W. Schwarting, A. Pierson, J. Alonso-Mora, S. Karaman, and D. Rus, "Social Behavior for Autonomous Vehicles," in *Proceedings of the National Academy of Sciences*, vol. 116, pp. 24972–24978, 2019.

[27] T. Gindele, S. Brechtel, and R. Dillmann, "Learning Context Sensitive Behavior Models from Observations for Predicting Traffic Situations," in *Proceedings of the International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pp. 1764–1771, IEEE, 2013.

[28] L. Leal-Taixé, M. Fenzi, A. Kuznetsova, B. Rosenhahn, and S. Savarese, "Learning an Image-Based Motion Context for Multiple People Tracking," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3542–3549, IEEE, 2014.

[29] M. Kuderer, S. Gulati, and W. Burgard, "Learning Driving Styles for Autonomous Vehicles from Demonstration," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2641–2646, IEEE, 2015.

[30] D. Sadigh, S. Sastry, S. A. Seshia, and A. D. Dragan, "Planning for Autonomous Cars that Leverage Effects on Human Actions," in *Robotics: Science and Systems*, vol. 2, 2016.

[31] D. Sadigh, S. S. Sastry, S. A. Seshia, and A. Dragan, "Information Gathering Actions Over Human Internal State," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 66–73, IEEE, 2016.

[32] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social LSTM: Human trajectory prediction in crowded spaces," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 961–971, IEEE, 2016.

[33] A. Vemula, K. Muelling, and J. Oh, "Social Attention: Modeling Attention in Human Crowds," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4601–4607, IEEE, 2018.

[34] N. Lee, W. Choi, P. Vernaza, C. B. Choy, P. H. Torr, and M. Chandraker, "DESIRE: Distant Future Prediction in Dynamic Scenes with Interacting Agents," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2165–2174, IEEE, 2017.

[35] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi, "Social GAN: Socially Acceptable Trajectories with Generative Adversarial Networks," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2255–2264, IEEE, 2018.

[36] A. Sadeghian, V. Kosaraju, A. Sadeghian, N. Hirose, S. H. Rezatofighi, and S. Savarese, "SoPhie: An Attentive GAN for Predicting Paths Compliant to Social and Physical Constraints," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1349–1358, 2019.

[37] S. M. LaValle, "Planning algorithms," *Planning Algorithms*, vol. 9780521862, pp. 1–826, 2006.

[38] E. F. Camacho and C. Bordons, *Model Predictive Control*. Advanced Textbooks in Control and Signal Processing, Springer, 1999.

[39] J. B. Rawlings and D. Q. Mayne, *Model Predictive Control: Theory and Design*. Nob Hill Publishing, 2009.

[40] L. Grüne and J. Pannek, *Nonlinear Model Predictive Control*. Springer, 2017.

[41] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[42] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient Flow in Recurrent Neural Nets : The Difficulty of Learning Long-Term Dependencies," in *A Field Guide to Dynamical Recurrent Neural Network*, Wiley-IEEE Press, 2001.

[43] S. Hochreiter and J. Urgen Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[44] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to Forget: Continual Prediction with LSTM," *Neural Computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[45] A. Y. Uteshev and M. V. Goncharova, "Point-to-Ellipse and Point-to-Ellipsoid Distance Equation Analysis," *Journal of Computational and Applied Mathematics*, vol. 328, pp. 232–251, 2018.

[46] A. Domahidi and J. Jerez, "Forces professional," *embotech GmbH (http://embotech.com/FORCES-Pro)*, 2014.

Glossary

List of Acronyms

SVO	social value orientation
ORCA	optimal reciprocal collision avoidance
VO	velocity obstacle
RVO	reciprocal velocity obstacle
RL	reinforcement learning
MPC	model predictive control
GAN	generative adversarial network
RNN	recurrent neural network
CNN	convolutional neural network
LSTM	long short-term memory
MSE	mean squared error
CVM	constant velocity model
RA-L	IEEE Robotics and Automation Letters
ICRA	IEEE International Conference on Robotics and Automation

List of Symbols

Abbreviations

Δt	sampling period
$\mathcal{I}$	set of robots
$\mathcal{I}_o$	set of obstacles
$\mathcal{W}$	workspace
$\mathcal{X}_{o,q}^k$	set of the dynamic obstacles' relative states at time k with respect to the query robot q
$\mathcal{X}_{r,q}^{k_1:k_2}$	set of the other robots' relative past state horizon from time k_1 to time k_2 with respect to the query robot q
n	number of robots
N_{past}	observation horizon length
N_{pred}	prediction horizon length
n_o	number of obstacles
p_i^k	position of robot/obstacle i at time k
p_{ij}^k	relative position of robot/obstacle i at time k with respect to robot/obstacle j
v_i^k	velocity of robot/obstacle i at time k
v_{ij}^k	relative velocity of robot/obstacle i at time k with respect to robot/obstacle j
x_i^k	state of robot/obstacle i at time k
x_{ij}^k	relative state of robot/obstacle i at time k with respect to robot/obstacle j