Learning a guidance policy to navigate among dynamic agents in constrained environments with continual reinforcement learning

J.S. Brinkman



TU Delft

Master of Science Thesis

Learning a guidance policy to navigate among dynamic agents in constrained environments with continual reinforcement learning

by

J.S. Brinkman

to obtain the degree of Master of Science in Mechanical Engineering
at the Delft University of Technology,
to be defended on Thursday July 22, 2021 at 15:00.

Student number: 4395557
Project duration: September 1, 2020 – July 22, 2021
Thesis committee:

Dr. J. Alonso-Mora,	TU Delft, supervisor
B. Brito,	TU Delft, daily supervisor
Dr. J. Kooij,	TU Delft

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Acknowledgements

This thesis research is conducted at the Cognitive Robotics department of the Technical University of Delft (TU Delft) to obtain a Master degree in Mechanical Engineering. I am thankful to everyone in the department that I got the opportunity to work in the fascinating field of robot navigation and machine learning. Although the university was closed for most of the time due to Covid-19, I gained a lot of knowledge and programming skills, which I would never have wanted to miss.

I would first like to thank my supervisor, Javier Alonso-Mora, for the guidance and insightful feedback in all our meetings. Furthermore, I would like to thank my daily supervisor, Bruno Brito, for all the excellent guidance and educational support throughout my thesis. Without your help, I would never have gained so much knowledge on machine learning and robot navigation, and I am thankful for that. Our discussions and meetings were always inspiring and motivated me to push my work to a higher level.

I would also like to thank Réka Bérci-Hajnovics, Ludo Stellingwerff, and Snehal Jauhri from the robotics company Dobots for their insightful feedback in our monthly meetings. Their great expertise in robotics gave me a different perspective on my research, which was invaluable. Moreover, the weekly workshops on new software and programming languages in the first few months of my thesis were fun and educational.

J.S. Brinkman
Delft, July 2021

Contents

1 Introduction **1**
1.1 Research question . . . 3

2 Paper **5**

3 Discussion **25**
3.1 Static obstacle representations . . . 25
3.2 Continual learning . . . 26
3.3 Real-world experiments . . . 26

4 Conclusion **27**
4.1 Conclusions . . . 27
4.2 Recommendations . . . 28
4.2.1 Recommendations to extend the network architecture . . . 28
4.2.2 Recommendations for continual reinforcement learning . . . 28

A Network training **31**
A.1 Comparing static obstacle representations . . . 31
A.1.1 Training setup . . . 31
A.1.2 Training results . . . 31
A.2 Continual reinforcement learning . . . 32

B Convolutional Neural Network (CNN) **33**
B.1 Network Architecture . . . 33
B.2 CNN training . . . 34

C Additional simulation experiments **35**
C.1 Experiments in the door scenario . . . 35
C.2 Continual learning experiments . . . 36
C.2.1 Additional results on learning an increasing number of agents . . . 36
C.2.2 Additional experiments on a sequence of task with static obstacles . . . 37

D Real-world experiments **41**
D.1 Robot description . . . 41
D.2 Real-world setup . . . 41
D.3 Additional real-world experiments . . . 42

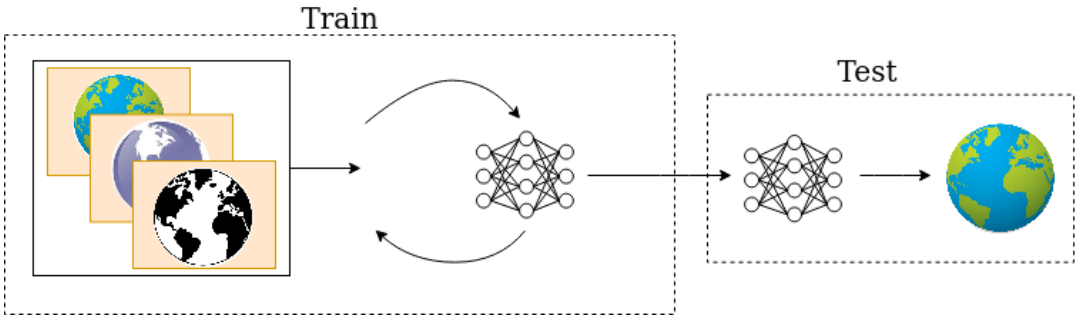
1 Introduction

In 1960, the first autonomous robot (Unimate) was introduced at a General Motors plant to extract hot metal parts from a casting machine [21]. By easing the workload of humans and increasing safety, the popular robot had shown the benefits of an autonomous system. Since then, autonomous mobile robots became an interesting research topic [18]. Most of these robots are designed to be used in structured and controlled industrial environments, like manufacturing plants or warehouses [12]. For instance, the supermarket Picnic is a new player in the European e-grocery market, designing highly automated distribution centres consisting of many robots to sort and distribute groceries for customers [14]. The deployment of robots operating in these structured environments is relatively easy compared to unstructured (human-populated) environments since dynamic obstacles are limited and special features for the robots can be designed. However, a new field of robotics is emerging where the applications of mobile robots are moving from structured industrial environments towards unstructured dynamic environments, e.g. hospitals, schools, offices, construction sites, or other human-populated environments. In these environments, autonomous robotic systems can be applied to various services, like delivery, cleaning, inspection, providing customer support, etc. For instance, the company iRobot focuses on robots that perform practical tasks to make daily life easy [13]. The company already sold 30 million house robots worldwide, including Roombas (robot vacuums), robot mops, floor scrubbing robots, etc., demonstrating the strong demand for robots in human-populated environments.

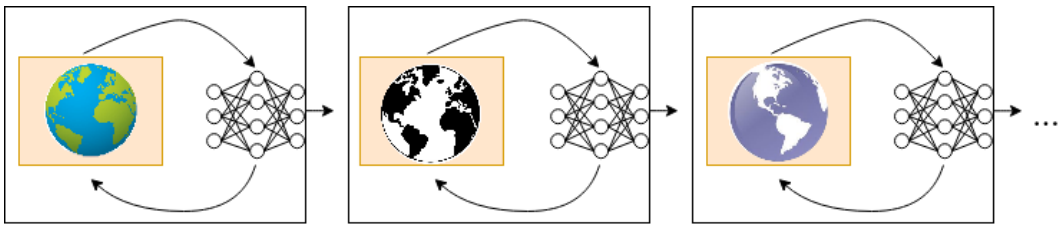
While advancing into unstructured dynamic environments, several challenges are introduced. One key challenge is how to navigate in close vicinity of humans. The behavior of the robot has to be carefully designed to be accepted by the surrounding humans. Not only should the behavior of the robot be safe and collision-free, but it should also be socially compliant, that is, having human-like behavior and respecting the social rules and norms [16]. However, multiple aspects provoke the challenge of robot navigation. One aspect is that human behavior is stochastic and only partially observable by the robot, making it difficult to predict their actions. Another aspect that makes robot navigation among humans difficult is the interaction between humans and the robot. Humans have the ability to 'read' other humans' intentions and plan accordingly. To achieve such socially compliant behavior for the robot, the robot should know how its actions can affect the human's actions and plan accordingly. Thus, the robot should take interactions among humans into account while navigating.

There are many methods for robot navigation developed since the introduction of mobile robots in human environments [17]. Early methods capture the interactions among surrounding agents by relying on geometry [2], physics [11], or handcrafted functions [27]. Another body of work uses the developments of deep neural networks to make the robot's behavior safer and socially compliant. These data-driven approaches learn a policy that can account for human-human and human-robot interactions to generate safe motion plans [4][8][9]. Particularly, deep reinforcement learning is often used to generate a socially compliant robot. For instance, [5] accounts for social norms when learning a collision-avoidance policy. [8] focuses on releasing the assumption that surrounding agents follow particular behavior rules. Although the learning-based methods are able to perform well in dynamic scenarios, they do not guarantee collision avoidance or feasibility with respect to the vehicle dynamics.

Another limitation of data-driven approaches is that they are restrictive towards new and unseen environments. The learning methods clearly distinguish two phases: a training phase and a testing



(a) Learning with train and test phase



(b) Continual learning

Figure 1.1: Schematic view of two different learning approaches. (a) Two phases: training phase and testing phase. In the training phase, a model is learned offline from data. The learned model is then deployed during testing. (b) Learning from data in a sequence, one after the other. Here, the model adapts continually to new data, i.e. it never stops learning.

phase (Fig. 1.1a). In the training phase, a model is trained offline in static and controlled conditions. After training, the learned model is frozen, i.e. model parameters will not change anymore. Then, the frozen model is deployed in test environments, assuming that the data distribution during training is similar to during testing. However, unstructured, dynamic environments are constantly changing and shifts in the distribution of data, also known as *concept drift*, are highly likely to occur. To accommodate concept drift, typically, the model has to be retrained entirely to avoid performance degradation.

On the other hand, a more natural way of learning is to release the distinction between the training and testing phase and learn incrementally from new incoming data [24][25] (Fig. 1.1b). It can be seen as a more human-like way of learning: humans accumulate knowledge throughout their lives by learning from new, incoming data. This way of learning allows a model to adapt to ever-changing environments during deployment. The model can gradually acquire new knowledge over a lifetime and better adjust to the current scenarios. Yet, training the model with new data introduces a bias towards the new data, which can interfere with old knowledge, a problem also known as *catastrophic forgetting*.

Continually learning from new data without forgetting the previously learned tasks, i.e. mitigating catastrophic forgetting, is a steadily growing topic of interest [19]. This research paradigm of continual learning (CL) focuses on improving the adaptation and generalization capability of machine learning models. The goal is to learn multiple tasks in a sequence, one after the other, resulting in a single model that can perform well on all tasks. Most research on CL focuses on supervised learning settings, specifically classification tasks [6]. For example, a model learns to discriminate between two consecutive digits from 0 to 10 [7][23][28]. However, the research on continual reinforcement learning is still in its infancy. In this thesis, we explore the use of continual learning with reinforcement learning in the field of robot navigation in social contexts. A robot navigating in highly dynamic environments is often subject to a changing environment. To perform well in these changing environments, we aim to develop an algorithm that can adapt a learned model continually to new environments while preserving previously learned knowledge. Therefore, our research can overcome the restrictive way of learning of typical learning-based methods and constitute a step forward in developing continual reinforcement learning strategies.

1.1. Research question

The extensive research on robot navigation in human environments has brought already much progress in this field. Specifically, the use of deep reinforcement learning (DRL) has led to promising algorithms performing well at robot navigation tasks among humans [8][9]. Still, there is room for improvements for these algorithms in terms of neural network architecture and learning of the policy. With this in mind, we pose the following research questions:

1. How can we build on available algorithms using deep reinforcement learning to achieve improvements in the navigation performance in realistic unstructured, dynamic environments?
2. Can continual reinforcement learning be successfully applied to learning a robot navigation policy in human environments, and can it constitute an improved generalized model?

2

Paper

Learning a guidance policy to navigate among dynamic agents in constrained environments with continual reinforcement learning

Sant Brinkman
Department of Cognitive Robotics
Delft University of Technology, Netherlands

***Abstract*— Mobile robots that operate in human environments require the ability to safely navigate among humans and other obstacles. Existing approaches use Deep Reinforcement Learning (DRL) to obtain safe robot behavior in such environments, but do not ensure collision avoidance or trajectory feasibility. This issue is solved by methods combining DRL with model predictive control (MPC). However, they do not account for static obstacle avoidance. Moreover, DRL-based approaches train their network in multiple environments with increasing difficulty to speed up training and improve generalization ability. By sequentially training the model on new (more complex) environments, new knowledge could interfere with old knowledge, a problem known as *catastrophic forgetting*. Despite performing well on the most challenging scenarios, performance on more simple scenarios is diminished. This paper introduces a continual reinforcement learning (CRL) strategy that can sequentially learn multiple navigation tasks while retaining performance on all previously learned tasks. For robot navigation, we utilize a combination of MPC with DRL, to develop an algorithm that can avoid not only dynamic interacting agents but also static obstacles. Our approach is shown to be able to safely navigate to a goal position in multiple environments. In addition, by sequentially learning multiple tasks, we improve navigation performance in terms of successful events and travel times and distances compared to other DRL approaches.**



Fig. 1: Our mobile robot operating in the corridor of the university, showing the MPC plan (blue markers) to a recommended subgoal (red marker).

I. Introduction

In recent years, there is a fast development from mobile robots operating in structured industrial environments towards unstructured, dynamic scenarios in which the robot acts in the vicinity of humans (e.g. offices, houses, streets). Yet, safe robot navigation in human environments remains a challenge due to the stochasticity of human behavior and the partial observability of their decisions. A robot in human environments requires a decentralized collision avoidance algorithm that has the ability to handle interactions among all agents (other humans) and the different behaviors that these agents exhibit.

Several approaches [1][2] have used optimization-based methods, e.g., model predictive control (MPC), for safe robot navigation to satisfy kinodynamic and collision avoidance constraints. While these approaches can quickly generate smooth trajectories in simple environments, they are difficult to scale to environments with a high number of agents. Another limitation of optimization-based methods is that they plan collision-free trajectories within a limited planning horizon leading to locally optimal solutions prone to deadlocks. To solve this issue, local planners are often guided by a cost-to-go heuristic [3] (e.g., Euclidean distance), which provides a subgoal in the local environment. While this is practical in static environments, the cost-to-go heuristic is less trivial in crowded, dynamic scenarios. Methods using deep reinforcement learning (DRL) are introduced to provide a subgoal in these dynamic scenarios that can learn to estimate a cost-to-go while taking complex interactions among agents into account [4][5][6]. The DRL methods can learn the computationally expensive operations of providing a subgoal in an offline training phase and then quickly query the learned policy. Some methods learn a policy directly from raw sensor data [7], whereas others first pre-process the raw sensor data before it is fed into the network [5][6]. Although these approaches are able to provide a subgoal in dynamic environments, a major concern in DRL approaches is the lack of guarantee for collision-free trajectories or feasible control actions to satisfy realistic dynamic constraints.

Combining DRL with optimization-based methods overcomes the limitations on both approaches [8]. In particular, DRL can learn an interaction-aware policy to provide global guidance to a local trajectory planner, which in its turn guarantees kino-dynamic feasibility and safety in highly dynamic environments [9]. However, these methods have only guaranteed collision avoidance among dynamic agents but not yet among static obstacles. In reality, static obstacles

are almost always present, for example, walls, furniture, etc. Thus, it is essential to incorporate static obstacles into the network to generate safe motion plans in realistic human environments (Fig. 1).

Since static obstacles can occur in many shapes and locations, it is important to generalize the trained model to multiple environmental contexts. Generally, DRL approaches improve the generalization ability of their model by training on a set of different environments. For instance, the training is often decomposed into several subtasks with increasing difficulty to enhance performance in complex environments [9][6][7]. However, training a model to convergence on a new subtask introduces a bias towards the new data, interfering with the previously learned tasks. The interference leads to performance degradation on the previously learned tasks, which is a problem known as *catastrophic forgetting* [10].

Simple solutions to this issue are to jointly learn on all tasks (domain randomization)[11], or to store experiences and replay them while learning (Experience Replay (ER))[12]. However, these solutions are less practical in real-world settings, where we do not have access to all data all at once and are limited by computational and memory constraints. In order to train a model in real-world settings, we have to continually learn from data that arrives in a sequence without forgetting what we have learned. The recently emerging paradigm of continual learning (CL) addresses this problem [13]. Since most of the CL research has focused on supervised learning settings [14], there is limited research on continual learning in reinforcement learning settings.

In this paper, we employ continual learning to learn sequentially multiple navigation tasks while satisfying a realistic setting. For robot navigation, we build upon the approach of [9] and expand it to also account for environmental constraints (static obstacles). Using DRL, a policy is trained to recommend a subgoal to the MPC planner. The MPC planner guarantees to respect collision avoidance constraints for static and dynamic obstacles when a feasible solution is found. Our collision avoidance policy is trained and tested in multiple scenarios in simulation.

The main contributions of this work are:

- A method that integrates static obstacle representations for robot navigation in dynamic environments utilizing the combination of DRL with MPC, in which a policy is learned to recommend a subgoal to the local planner satisfying kinodynamic and collision avoidance constraints (if a feasible solution to the optimization problem is found),
- A continual reinforcement learning strategy that trains a model sequentially on multiple tasks with increasing difficulty while consolidating knowledge from previous tasks.

We present simulation results showing that our approach improves over several baselines when continually learning a sequence of tasks. Finally, we demonstrate our approach in real-world environments.

II. Related Work

This section reviews work on mobile robot navigation in unstructured dynamic environments and continual learning methods.

A. Robot navigation

In the field of robot navigation, a large variety of algorithms have been developed to obtain safe robot behavior among static and dynamic obstacles [15]. Some early methods use a straightforward approach, in which the robot navigates to the goal while avoiding obstacles on the way, also called reaction-based approaches. An example of these reaction-based methods for multi-agent systems is Reciprocal Velocity Obstacles (RVO) [16] where all agents, that use the same policy, select a collision avoiding velocity to other agents resulting in collaborative collision avoidance. Optimal Reciprocal Collision Avoidance (ORCA) [17] is one of the many extensions of RVO, solving the optimization problem with linear programming. Although these reactive-based approaches are computationally efficient and ensure physical safety of the nearby agents, these approaches can have highly reactive behaviors due to the use of one-step look-ahead predictions. Predictive motion planners, e.g., model predictive control (MPC), achieve less reactive but more complex and predictive behavior by planning over a prediction horizon [2]. However, predictive planners are computationally more expensive.

More recent work uses learning-based methods, e.g., deep reinforcement learning (DRL), to model social interactions and cooperation [5][6][7]. DRL uses neural networks (NNs) to learn mapping functions, which map input sensor data to output actions. Some DRL approaches use end-to-end training, where a neural network directly processes raw sensor data to output an action probability [7][18]. The raw sensor measurements can be provided by a single sensor [7] or by multiple sensors [18] to be more robust in highly dense scenarios. In contrast to using raw sensor data, other work proposes to first extract information about the surrounding agents from perception pipelines before feeding it to the neural network [19][6][4][5]. For instance, [19] learned to select collision-free actions from the surrounding agent's states and [4] incorporated social norms to achieve more socially compliant behavior. To avoid dependency on a fixed number of agents, [6] extended previous work by accounting for a variable number of agents. Subsequent work added a attention module to better assign the relative importance of nearby agents in the crowd [5][20][21].

While data-driven approaches can incorporate social interactions among dynamic agents, they cannot guarantee to respect kinodynamic and collision avoidance constraints. Therefore, recent approaches combine DRL with local optimization techniques to benefit from both classes of methods. For example, [22] collects diverse training samples from the environment by using the exploration strength of a reinforcement learning (RL) policy. Then, MPC is used to improve the sampled trajectories at test time. In the field of robot navigation, [9] uses DRL to learn a policy recommending

a subgoal based on observations in the environment. Then, an MPC planner computes a trajectory towards the recommended subgoal, guaranteeing to respect collision avoidance and kinodynamic constraints (if a feasible solution is found). The latter approach incorporates social interactions among dynamic agents, but does not include static obstacles in the network. Yet, static obstacles can highly influence the other agent's interactions and motion plans. For instance, [23] shows how other agent's motion predictions significantly improves by accounting for static obstacles. In this work, we expand the approach of [9] to account for static obstacles as we believe it is highly essential.

B. Continual Learning

Continual learning (CL), also known as lifelong learning, has a growing interest in the machine learning field to overcome the problem of catastrophic forgetting [13]. CL is inspired by human-like learning, in which a model keeps on learning sequentially from a continuous stream of data containing multiple tasks. The goal is to create a model that performs well on all tasks by retaining the learned ability on previous tasks. The existing approaches for continual learning can be divided into three categories: architecture-based, memory-based methods and regularization-based approaches [10].

Architecture. These approaches expand the neural network model with new layers, nodes, or modules to separate parameters from previous and new information. After expansion, the network is retrained with an increased number of neurons while keeping the model for the previously learned tasks unchanged. Progressive networks [24] create a new model for every newly learned task and connect it to the base architecture. [25] introduced Reinforced Continual Learning (RCL), in which RL is implemented as a controller to determine the optimal number of nodes/filters to add to the network for a new task. Although the architecture-based approaches can allocate specific neurons to tasks, the neural network model grows with each new task making it very large and complex. Therefore, architecture-based approaches can suffer from scalability issues.

Memory. Memory-based approaches maintain a neural network model that is fixed in size and stores data or pseudodata from previous tasks in a buffer for the purpose of rehearsal. The data stored in the buffer can be randomly chosen or carefully selected. The latter is used in iCaRL [26], where a core set is kept with the most representative data samples to help the model retain essential information. While this method requires task boundaries, some methods can work in a task-free setting [27][28]. [27] selects the samples for the replay buffer based on the diversity of the samples to represent all previous data as good as possible. [28] proposes gradient updates on the samples in the replay buffer to make them more likely to be forgotten (they assume that samples that are more 'forgettable' help to retain test performance). This method can be applied on top of other replay methods, like GSS or experience replay (often used in RL). However, since memory-based approaches require a separate memory to store data, the memory can become large with an increasing number of tasks. Moreover, memory-based approaches are prone to overfit due to the previous data only being partially stored and replayed.

Regularization. In contrast to architecture-based and memory-based approaches, regularization-based approaches do not require much computational and memory resources. These approaches use a regularization term to prevent significant changes in parameters important to previous tasks. The regularization-based approaches mainly differ in the computation of the important parameters. EWC [29] computes the important parameters by an approximation of the diagonal of the Fisher Information Matrix (FIM). [30] modified EWC to an online setting by keeping a running sum of the previous tasks' Fisher Information Matrices instead of keeping one FIM for every previous task. Keeping only one FIM reduces the computational complexity, making online EWC scalable to a large number of tasks. Synaptic Intelligence (SI) [31] computes the important parameters in an online manner by measuring the contribution of parameters to the loss function: the importance of the parameters is represented in the loss. Another method that maintains an online setting is Memory-aware Synapses (MAS) [32], which computes the important parameters by accounting for the influence of changes in the parameters on the prediction of the model.

Currently, most CL approaches in research are focused on supervised learning settings [33]. Only a small subset of research has been focused on combining continual learning with RL, also known as continual reinforcement learning (CRL). Most existing CRL approaches focus on games [34][29], but to the extent of our knowledge, CRL is not explored in the context of robot navigation in human environments. Here, we employ continual learning in the domain of robot navigation to allow learning in an online setting. This new training approach leads to a more generalized policy that overcomes catastrophic forgetting during training with CL.

III. Preliminaries

Throughout this article, vectors are denoted in bold lowercase letters, $\mathbf{x}$, matrices in capital, M, and sets in calligraphic uppercase, $\mathcal{S}$. $\|\mathbf{x}\|$ denotes the Euclidean norm of $\mathbf{x}$ and $\|\mathbf{x}\|_Q = \mathbf{x}^T Q \mathbf{x}$ denotes the weighted squared norm. The sub index $\cdot_t$ indicates the value at time-step t. The super index $\cdot^i$ and $\cdot^o$ indicates agent i and obstacle o, respectively. No super index refers to the ego-agent. In the RL formulation, $\{\mathbf{s}, \mathbf{a}\}$ denote the state and action, respectively. In the optimization problem, $\{\mathbf{x}, \mathbf{u}\}$ denote the control state and actions commands, respectively.

A. Problem formulation

Consider a mobile robot that has to complete several navigation tasks in multiple environments while interacting with n_{obs} static obstacles and n humans, hereinafter referred to as agents. Let the ego-agent denote the mobile robot on the plane $\mathcal{W} = \mathbb{R}^2$, where $\mathbf{p}$ and $\mathbf{v}$ denote the ego-agent's position and velocity, respectively. For each other

agent $i \in \{1, \dots, n\}$ in the environment, r^i denotes the radius, $\mathbf{p}^i \in \mathbb{R}^2$ denotes the position, and $\mathbf{v}^i \in \mathbb{R}^2$ the velocity, which is assumed to be constant. For each static obstacle $o \in \{1, \dots, n_{obs}\}$, $\mathbf{p}^o \in \mathbb{R}^2$ denotes the position. The static obstacles occupy an area denoted as $\mathcal{O}^{static} \in \mathcal{W}$. The area occupied by the ego-agent is denoted as $\mathcal{O}$ and the area by the other agents as $\mathcal{O}^i$.

At each time step t, the ego-agent observes its own state $\mathbf{s}_t$, the i-th other agent's states $\mathbf{S}_t = \bigcup_{i \in \{1,\dots,n\}} \mathbf{s}_t^i$ and a static obstacle representation of the environment $\mathbf{O}_t^{static}$ (more details in Sec. IV-B). After the observation of the states, the ego-agent takes action $\mathbf{a}_t$ and receives reward $R(\mathbf{s}_t, \mathbf{a}_t)$ and the states $\mathbf{s}_{t+1} = h(\mathbf{s}_t, \mathbf{a}_t)$ at the next time step $t+1$, under the transition model h. We consider a partially observable setting, in which only the position and velocity of the other agent's is available, but information about their goal position is assumed unknown.

The aim is to find a policy π maximizing the cumulative rewards over time to navigate from an initial position $\mathbf{p}_0$ to the final goal $\mathbf{g}$ while satisfying the ego-agent's dynamics and the static and dynamic collision avoidance constraints. The policy can be defined as the following optimization problem:

$$
\begin{aligned}
\pi^* = \underset{\pi}{\operatorname{argmax}} \quad & \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R(\mathbf{s}_t, \pi(\mathbf{s}_t, \mathbf{S}_t, \mathbf{O}_t^{static}))\right] \\
\text{s.t.} \quad & \mathbf{x}_{t+1} = f(\mathbf{x}_t, \boldsymbol{u}_t), && \text{(1a)} \\
& \boldsymbol{s}_T = \mathbf{g}, && \text{(1b)} \\
& \mathcal{O}_t(\mathbf{x}_t) \cap (\mathcal{O}_t^{static} \cup \mathcal{O}_t^i) = \emptyset && \text{(1c)} \\
& \mathbf{u}_t \in \mathcal{U},\ \mathbf{s}_t \in \mathcal{S},\ \mathbf{x}_t \in \mathcal{X}, && \text{(1d)} \\
& \forall t \in [0, T], \quad \forall i \in \{1, \dots, n\}
\end{aligned}
$$

where Eq. 1a represents the transition dynamic constraints under the dynamic model f, Eq. 1b the terminal constraints, and Eq. 1c formalizes the collision avoidance constraints. We denote $\mathcal{U}$, $\mathcal{S}$ and $\mathcal{X}$ (1d) as the corresponding set of admissible control inputs, states, and control states, respectively.

B. Dynamic obstacles

The other agents in the environment are modeled with different behaviors to create scenarios similar to the real world. We assume that each other agent's behavior can either be cooperative or non-cooperative. For the cooperative policy, Reciprocal Velocity Obstacle (RVO)[16] is used. In RVO, all other agents contribute to joint collision avoidance with a cooperation coefficient $c^i \sim \mathcal{N}(0.1, 1)$, a higher cooperation coefficient results in more shared collision avoidance. For the non-cooperative policy, a constant velocity (CV) policy is used, where the agents navigate in a straight line to the goal.

C. Static obstacles

An occupancy grid map encodes all static obstacles in the environment. The global occupancy grid map is built a priori with predefined coordinates (for simulation) or from sensor readings (in reality). Note that the occupancy grid map only includes static obstacles and no dynamic obstacles.

D. Robot Description

We employ a second-order unicycle model for the ego-agent [35]:

$$
\begin{aligned}
\dot{x} &= v \cos\psi \quad \dot{v} = u_a \\
\dot{y} &= v \sin\psi \quad \dot{\omega} = u_\alpha \\
\dot{\psi} &= \omega
\end{aligned} \tag{2}
$$

where x and y describe the ego-agent's position coordinates, ϕ describes the heading in the global frame, v and w are the linear and angular velocity, respectively. u_a and u_α are the linear and angular acceleration, respectively.

E. Task settings

In continual learning methods, multiple tasks are learned in a sequential setting. Different tasks could vary in many different ways, such as different objects, different people, different objectives, different environments, etc. In our case, a task is defined as an environment, so a change in the environment is a change in task. We assume there are z different tasks $t \in \{1, \dots, z\}$. The learning policy can only access the tasks sequentially, in the order $t_1, t_2, \dots, t_z$. We aim to learn policy π (Eq. 1a) such that it performs well on all tasks.

IV. Approach

A limitation of approaches combining DRL with MPC is that they do not account for static obstacles in the navigation strategy. We extend the approach of [9], called GO-MPC, to account for environmental constraints. Moreover, we employ continual learning in our framework to allow training on multiple tasks. Our complete framework consists of deep reinforcement learning (DRL) with MPC to generate safe motion plans in realistic environments. Figure 2 shows an overview of our complete framework. In every task, reinforcement learning is used to learn a sequence of intermediate subgoals that guide the ego-agent towards the final goal. Then, MPC generates safe motion commands towards the provided subgoal while satisfying collision avoidance constraints if a feasible solution is found.

In this section, we first explain the static obstacle representations as input to our neural network to account for environmental constraints (Sec. IV-A). Then, we describe our RL framework (Section IV-B) and the neural network model (Section IV-B.2). Section IV-C describes the local planner (MPC) and section IV-D describes continual learning. Finally, in section IV-E, we present our complete training algorithm.

A. Static obstacle representations

Environmental constraints always bound real-world scenarios; think about walls, furniture, objects, etc. When navigating in these cluttered environments, humans have the remarkable ability to plan their motions accordingly. However, humans act differently to static obstacles than other humans. For instance, a human is more comfortable getting

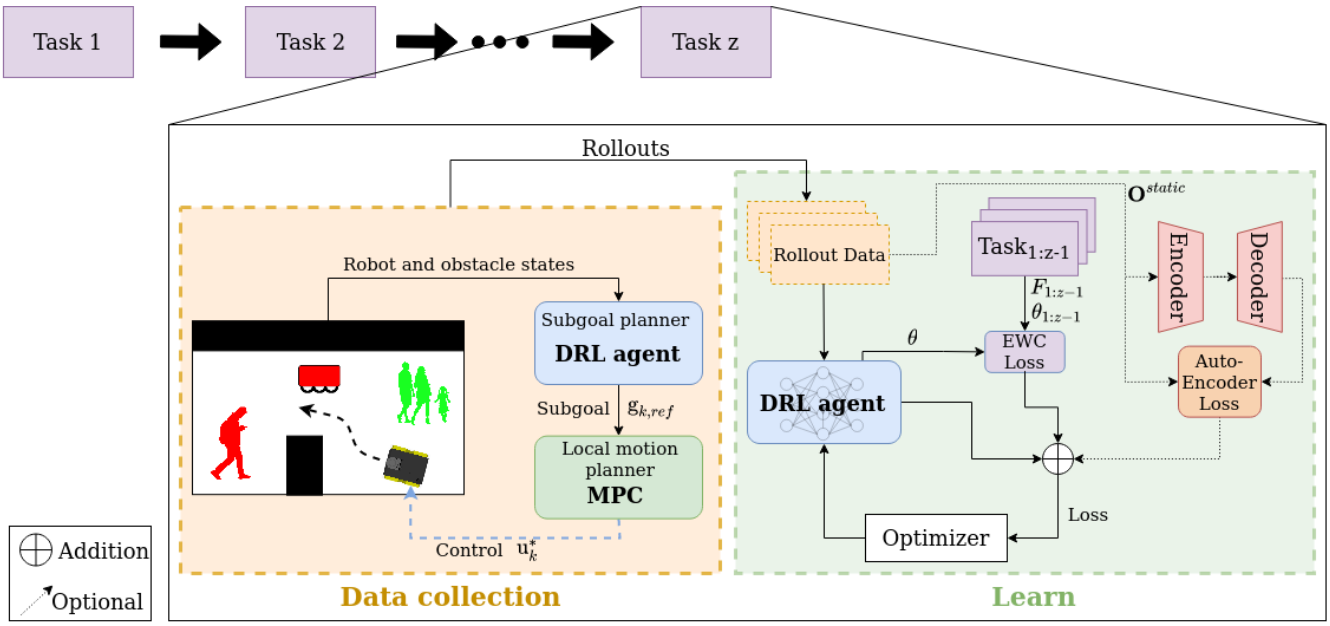


Fig. 2: High-level overview of the proposed navigation strategy. We train a policy sequentially on different environments (tasks). In each task, the subgoal planner observes an environment with static and dynamic obstacles and recommends a subgoal to the MPC planner, which on its turn computes control commands for the robot. The policy used by the subgoal planner is optimized based on rollouts (performed episodes), previous tasks and a CNN (optional). The EWC loss mitigates catastrophic forgetting on previously learned tasks.

close to a static object than to another human. Hence, we believe it is essential to account for both static and dynamic obstacles to obtain socially compliant robot behavior. To achieve this, we add a static obstacle representation as an input channel to our neural network (Section IV-B.2).

Here, we introduce two static obstacle representations: a local occupancy grid and an angular map. The occupancy grid is a common way to encode static obstacles in the environment. Still, the dimensionality of the occupancy grid is high, which could cause the overall training time to be large. To reduce dimensionality, [23] introduced the angular map. The angular map in [23] encodes information about other agents, but we use it to encode static obstacles. Hence, we compare both representations to evaluate their performance.

1) Occupancy grid: The local occupancy grid is an extract of the environment's global map, centered and aligned with the robot's position and heading, respectively (Figure 3). In the fixed-sized grid, cells containing an obstacle have a value of 1 and cells that are free (no obstacle) have a value of 0. It is important to note that the occupancy grid has to be provided beforehand to the agent, making it less flexible for real robot applications.

To reduce the high dimensionality of the occupancy grid and hence the overall training time, a convolution neural network (CNN) is used, as in [23]. The CNN compresses the occupancy grid into a representation in which the spatial context is maintained. The CNN is trained with a convolutional auto-encoder (AE) during training of the complete network. The AE consists of two parts, the encoding and decoding part. In the encoding part, the occupancy grid output is reduced to a compressed representation. The decoding part decodes the compressed representation to represent the output of the occupancy grid again. The training error of the AE can be defined as [23]:

$$\mathcal{L}(g_{\text{in}}, g_{\text{out}}) = \sum_{i=1}^{D_x} \sum_{j=1}^{D_y} (g_{\text{in}}(i,j) - g_{\text{out}}(i,j))^2 \qquad (3)$$

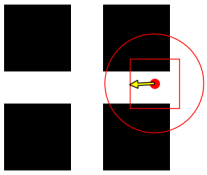
where g_{in} is the input occupancy grid, g_{out} is the decoded occupancy grid, D_x and D_y are the dimensions of the grid, and i and j are the grid indices along respectively the x- and y-axis. To obtain the AE parameters, $\mathcal{L}$ should be minimized.

2) Angular Map: The angular map is another way to represent static obstacles. The advantage of the angular map is that the dimensionality is lower than the occupancy grid, which reduces the overall training time.

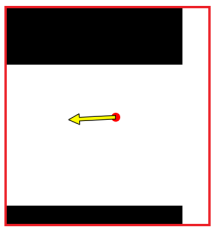
The angular map encodes the information of obstacles in the surroundings of the robot by using angular ranges. The output of the angular map is a one dimensional vector, $\mathbf{f}$, in which each element defines the minimum distance between the agent and the obstacles that lie within an angular range, as shown in Figure 3. Similar to the local occupancy grid, the angular map is centered at the agent's position and aligned with its heading. The angular map vector consists of W elements, where W is the number of uniform angular cells. Each element in the vector, f_w, with $w \in \{1, ..., W\}$ can be described as:

$$f_w = min(r_{\text{max}}, \{\rho^o | o \in \{1, ..., N_{\text{obs}}\}\}) \qquad (4)$$

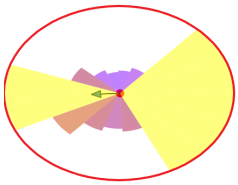
where r_{max} is the maximum range around the robot to observe obstacles, N_{obs} is the total number of obstacles in



(a) Global occupancy grid



(b) Local occupancy grid



(c) Angular map

Fig. 3: Depiction of the static obstacle representations, including the ego-agent (red dot), it's heading (arrow) and the maximum range (red contours). The local occupancy grid (b) and the angular map (c) correspond to the global occupancy grid (a). In the angular map, each slice encodes the distance to the closest obstacle within that slice. The darker the color, the smaller the distance.

the range of the agent, $\rho^o = \|\mathbf{p}, \mathbf{p}^o\|$ describes the distance between the ego-agent and the closest obstacle o. The angular grid ranges from α_{min} to α_{max}, where each opening angle γ_k can be described as:

$$\gamma_w = (w-1)\frac{\alpha_{\text{max}} - \alpha_{\text{min}}}{W} \tag{5}$$

The angular map works similarly to a laser scan sensor, in which every range of the laser scan can detect an object in the environment. Therefore, the advantage of the angular grid is that it can be used in real robot experiments without the need for a global map beforehand. Additionally, obstacles closer to the ego-agent are captured by more angular cells, resulting in an angular map vector inferring higher importance to closer obstacles without the need of an additional attention model [5]. Closer obstacles are more important to the decisions of the agent. Hence, a more detailed representation of closer obstacles is advantageous.

For neural network purposes, the values in the angular map vector are normalized as:

$$f_w = \frac{1 - f_w}{f_{\text{max}}} \tag{6}$$

B. Global planning: learning to guide

A robot navigating among dynamic decision-making agents and static obstacles can be formulated as a sequential decision making problem, which can be solved with a reinforcement learning framework. In our approach, reinforcement learning is used to learn a sequence of intermediate subgoals that guide the ego-agent towards the final goal while minimizing the cost-to-go.

1) Reinforcement learning: During the RL training process, we seek the optimal policy $\pi : (\mathbf{s}_t, \mathbf{S}_t, \mathbf{O}_t^{\text{static}}) \rightarrow \mathbf{a}_t$, which maps the ego-agent's observations of the environment to a probability distribution of actions. The ego-agent's observations are defined as the joint state $[\mathbf{s}, \mathbf{s}^i, \mathbf{O}^{\text{static}}]$, which is divided in three sources of information: information about the ego-agent, information about the other agents in the environment, and information about static obstacles. The first input is the state of the ego-agent ($\mathbf{s}_t$) at time t, which can be described as follows:

$$\mathbf{s}_t = [d_g, \mathbf{g}_{\text{rel}}, v_{\text{ref}}, \psi, r] \tag{7}$$

where $d_g = \|\mathbf{p}_t - \mathbf{g}\|$ is the ego-agent's distance to the goal and $\mathbf{g}_{\text{rel}} = \mathbf{p}_t - \mathbf{g}$ is the relative goal with respect to the ego-agent's position. The second input includes the states of the surrounding agents:

$$\mathbf{s}_t^i = [\mathbf{p}_t^i, \mathbf{v}_t^i, r^i, d_t^i, r^i + r] \;\; \forall i \in \{1, n\}, \tag{8}$$

where $d_t^i = \left\|\mathbf{p}_t - \mathbf{p}_t^i\right\|$ is the distance to the i-th agent. The third input includes the static obstacle representations (Sec. IV-A).

The action space is a set of permissible position increments in continuous space $\mathcal{A} \subset \mathbb{R}^2$. The position increments are denoted as $\boldsymbol{\delta}_t = [\boldsymbol{\delta}_{t,x}, \boldsymbol{\delta}_{t,y}]$, where $\boldsymbol{\delta}_{t,x}$ and $\boldsymbol{\delta}_{t,y}$ are the position increments (x, y) at time step t, respectively. The actions are bounded by the the planning horizon N and the maximum linear velocity v_{max} as follows: $\|\boldsymbol{\delta}_t\| \leq N v_{\text{max}}$.

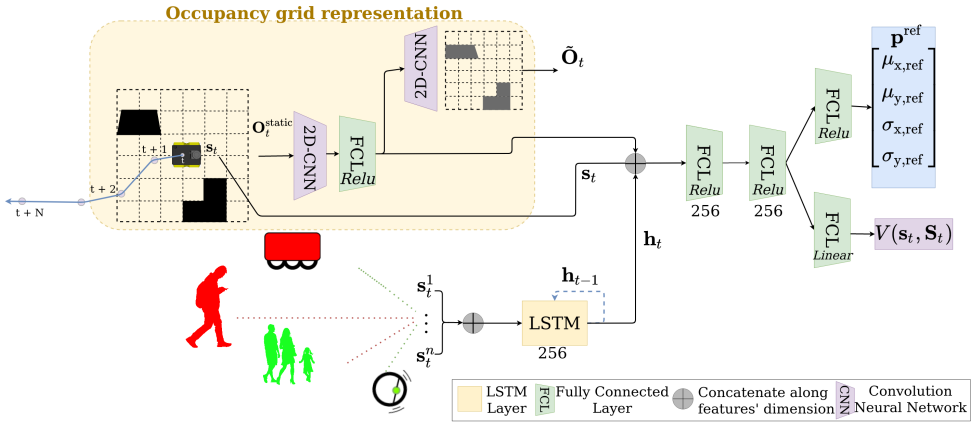
To enable the network to learn to reach the goal position, a reward function is formulated:

$$R(s,a) \equiv \begin{cases} r_{\text{goal}}, & \text{if } \mathbf{p}_t = \mathbf{g} \\ r_{\text{collision}}, & \text{if } d_{\text{min}} < r + r^i \;\; \forall i \in 1, n \\ r_{\text{collision}}, & \text{if } d_{\text{min}}^o < r \\ r_{\text{near}}, & 0 < d_{\text{min}} < d_{\text{near}} \\ r_t, & \text{otherwise} \end{cases} \tag{9}$$

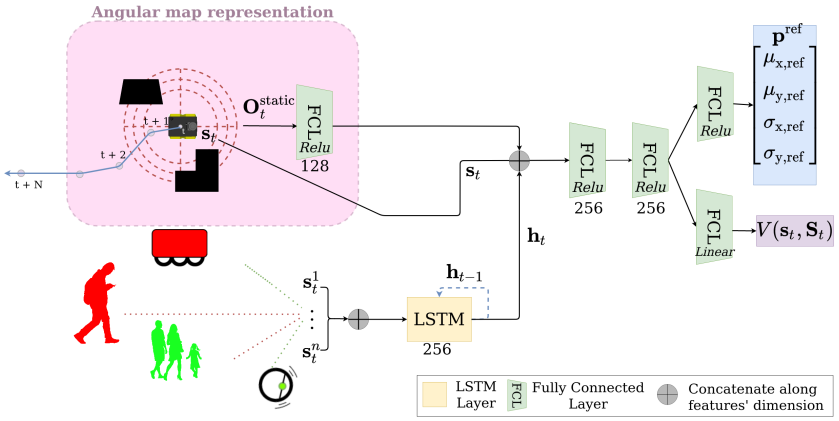
where $d_{\text{min}} = \min_i ||\mathbf{p}_t - \mathbf{p}_t^i||$ is the distance to the closest agent and $d_{\text{min}}^{\text{o}}$ is the distance to the closest static obstacle. The reward function tells the agent how well it performed; it awards the agent when reaching the goal (r_{goal}) and it gives a penalty when the agent collides with other agents or static obstacles ($r_{\text{collision}}$). Also, the robot is penalized when it comes too close to other agents, i.e. within a distance of d_{near}. In addition, r_t is a *time* reward to encourage the speed of the agent.

2) Neural network model: We design two neural network architectures to account for static obstacles, as depicted in Fig. 4. Note that the only difference between the networks is the static obstacle input channel (colored boxes).

There are three input channels to the neural network: 1) the ego-agent's state, 2) the other agent's states, and 3) a static obstacle representation. An LSTM layer processes the other agent's states to account for a variable number of surrounding agents states [6] and to model existing time-dependencies [9]. The input channel for static obstacles is either the occupancy



(a) Network architecture with occupancy grid representation



(b) Network architecture with angular map representation

Fig. 4: The proposed network architectures incorporating static obstacles. Three sources of information are concatenated to serve as input to the neural network: the ego-agent's state $\mathbf{s}_t$, the other agent's states $\mathbf{s}_t^i$, and a representation of the static obstacles in the environment, which can be (a) an occupancy grid representation, or (b) an angular map representation. The output of the model is a policy and a scalar value function.

grid or the angular map (Sec. IV-A). The output of the occupancy grid is handled by a convolutional neural network (CNN), of which the weights are optimized during training. A fully connected layer processes the output of the angular map, like in [23].

After processing the three input channels, their outputs are concatenated to a joint state representation, which is fed into the neural network consisting of two fully connected layers (FCL). The neural network produces two output layers. The first output is a scalar value function, and the second output is a policy represented as a continuous probability distribution over actions.

C. Local Collision Avoidance: Model Predictive Control

To provide guarantees on collision avoidance or feasibility with respect to the robot dynamics, MPC is used to generate control commands for the robot. In this section, we explain the MPC formulation considering the ego-agent's dynamics model (Sec. III-D) and the collision avoidance constraints. The subscript $\cdot_t$ (current time-step) is left out to simplify the notations.

1) State and Control Inputs: The control input vector of the ego-agent is $\mathbf{u} = [u_a, u_\alpha]$. The control state is $\mathbf{x} = [x, y, \psi, v, w] \in \mathbb{R}^5$ following the dynamics model in III-D.

2) Dynamic Collision Avoidance: As proposed in [9], collision-free motions around dynamic obstacles (humans) can be ensured by defining a set of nonlinear constraints to impose that the radius of the other agents r^i does not intersect with the ego-agent's radius r:

$$c_k^i(\mathbf{x}_k, \mathbf{x}_k^i) = \left\|\mathbf{p}_k, \mathbf{p}_k^i\right\| \geq r + r^i, \tag{10}$$

where k is the planning step, $\mathbf{x}$ and $\mathbf{x}^i$ are the ego-agent and other agent's control states, respectively.

3) Static collision avoidance: To guarantee safety around static obstacles (if a feasible solution exists), we define a set of linear constraints for obstacles in the environment. The advantage of using linear constraints compared

to computing a free-space polygon is that the environment is less constrained. Algorithm 1 describes all the steps to compute the linear constraints from a global occupancy map. First, we extract the contours of all static obstacles on the global occupancy map. Extracting contours is done by first creating a binary image of the map, and then the function $findContours$ from OpenCV [36] can detect white objects from a black background and compute their contours (vertices of the obstacle). The contours could define non-convex obstacles. Hence, it is important to convexify all n_{obs} obstacles. Convex obstacles are obtained by computing the convex hull from a set of points (provided by the contours). As in [37], the convex hull can be computed by a function denoted as $convhull$.

Next, for every obstacle o, we impose a linear constraint to guarantee $\mathcal{O}_t(\mathbf{x}_t) \cap \mathcal{O}_t^{static} = \emptyset$ as

$$c_k^o = \mathbf{a}^T \mathbf{p}_k \leq b - r, \tag{11}$$

where $\mathbf{a} = \mathbf{n}_k^o$ is the normal vector and $b = -\mathbf{a}^T \mathbf{p}_k^o$. Figure 5 visualizes all notations used in a linear constraint computation.

We limit the problem complexity and increase the chance of finding a solution by considering a set of the m closest linear constraints to the ego-agent, with $m \leq n_{obs}$. In order to obtain the set of closest linear constraints, the distance between the robot's position $\mathbf{p}_k = [p_{k,x}, p_{k,y}]$ and the linear constraint, defined by line $L : ax + by + c = 0$, is computed as:

$$\|\mathbf{p}_k, c_k^o\| = \frac{|ap_x + bp_y + c|}{\sqrt{a^2 + b^2}} \tag{12}$$

The linear constraints can be ordered by distance to the ego-agent, and the set of m closest constraints is defined as:

***Definition* 1.** *A set of $\mathcal{C}^m \subseteq \mathcal{C}^{n_{obs}}$ is the set of m-th closest linear constraints if the distance $\forall c_k^p \in \mathcal{C}^m, \forall c_k^q \in \mathcal{C}^{n_{obs}} \backslash \mathcal{C}^m : \|\mathbf{p}_k, c_k^p\| \leq \|\mathbf{p}_k, c_k^q\|$.*

The proposed linear constraints ensure that the robot can avoid the m closest obstacles. However, one drawback of using linear constraints is that if there are more than m obstacles close to the robot (e.g. heavily cluttered environments), the robot does not take all necessary linear constraints into account. Therefore, in heavily cluttered environments, the robot cannot always guarantee collision avoidance with static obstacles.

4) Cost function: The DRL network provides a reference position $\mathbf{p}_0^{\text{ref}}$ (subgoal), which guides the ego-agent to the final goal $\mathbf{g}$ and minimizes the cost-to-go while accounting for static and dynamic obstacles. Following [9], we define a terminal cost to enable the ego-agent to reach the provided subgoal:

$$J_N(\mathbf{p}_N, \pi(\mathbf{x}, \mathbf{X})) = \left\| \frac{\mathbf{p}_N - \mathbf{p}_0^{\text{ref}}}{\mathbf{p}_0 - \mathbf{p}_0^{\text{ref}}} \right\|_{Q_N}, \tag{13}$$

where $\mathbf{p}_N$ is the ego-agent's terminal position (at planning step N) and Q_N is the weight coefficient.

Since we aim for socially compliant behavior, the ego-agent should generate smooth trajectories. To obtain smooth

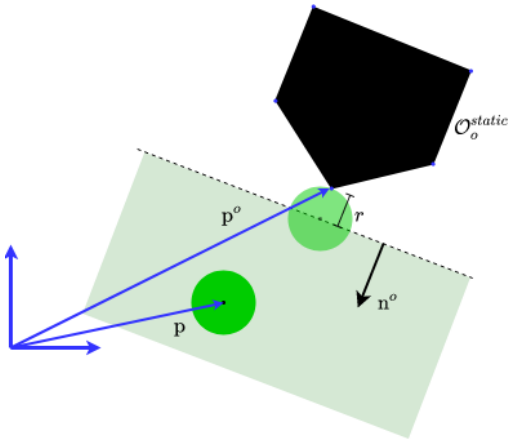


Fig. 5: Depiction of linear constraint between robot (green dot) and obstacle (black). The green region represents the robot's free space resulting from the linear constraint. Note that the linear constraint is displaced by the robot's radius r to ensure collision avoidance.

Algorithm 1 Linear constraints

Input: Global occupancy grid O^{static}
for $t = 0, \ldots,$ time **do**
  Detect contours from global map with contours $= findContours(O^{static})$.
  for contours $= 0, \ldots, n_{\text{obs}}$ **do**
    Convex hull $= convhull$(contours)
    Compute linear constraint c_k^o (Eq. 11).
    $C^{n_{obs}} \leftarrow c_k^o$
    Compute distance between constraint and ego-agent (Eq. 12).
  end for
  Order all $C^{n_{obs}}$ by distance (ascending order).
  Keep the set of m closest constraints C^m.
end for
return C^m

trajectories, we use a quadratic penalty on the control commands of the ego-agent as the stage cost:

$$J_k^{\mathbf{u}}(\mathbf{u}_k) = \|\mathbf{u}_k\|_{Q_u}, \quad k = \{0, 1, \ldots, N-1\}, \tag{14}$$

where Q_u is the weight coefficient.

5) MPC Formulation: At every time step k, a non-convex optimization problem is solved with planning horizon N conditioned on the following constraints:

$$\begin{aligned} \min_{\mathbf{x}_{1:N}, \mathbf{u}_{0:N-1}} \quad & J_N(\mathbf{x}_N, \mathbf{p}_0^{\text{ref}}) + \sum_{k=0}^{N-1} J_k^u(\mathbf{u}_k) \\ \text{s.t.} \quad & \mathbf{x}_0 = \mathbf{x}(0), \quad \text{(1d), (2)}, \\ & c_k^i(\mathbf{x}_k, \mathbf{x}_k^i) > r + r^i, \\ & c_k^o \leq b - r, \\ & \mathbf{u}_k \in \mathcal{U}, \quad \mathbf{x}_k \in \mathcal{S}, \\ & \forall i \in \{1, \ldots, n\}; \forall k \in \{0, \ldots, N-1\}. \end{aligned} \tag{15}$$

D. Continual learning

Current approaches using DRL have a clear distinction between training and testing. After a model is trained, it is deployed in the real world and the model's parameters are never adapted again. However, since real-world environments are often non-stationary and subject to frequent changes, the need for a trained model that can generalize well in real-world settings is high. The current approaches using DRL try to achieve a generalized model by training on multiple tasks. Yet, learning a new task could interfere with previously learned tasks, resulting in performance degradation on the previous tasks. Continual learning approaches address this issue by mitigating catastrophic forgetting.

We employ continual learning in the training of our model to continually learn a sequence of tasks $\{t_1, t_2, \ldots, t_z\}$ while retaining knowledge on previous tasks. In order to consolidate knowledge, we expand the loss function with a regularization term. For regularization, we use the state-of-the-art continual learning approach EWC [29], since it has been shown to work in RL settings. Additionally, we compare EWC to online EWC [30], which has less memory and computational costs.

1) EWC: To properly asses our continual reinforcement learning approach, we split the training into multiple tasks and learn them sequentially without repeating older tasks, that is $t_1 \rightarrow t_2 \rightarrow \cdots \rightarrow t_z$. When learning the first task, the network learns any θ (network's parameters) performing well on task 1: $\theta^* = \theta_1$. When learning task 2, a typical network will adapt θ to perform well on task 2 ($\theta^* = \theta_2$). However, to consolidate knowledge, the network should ideally adapt θ to perform well on task 1 and 2 ($\theta^* = \theta_{1,2}$). EWC aims to achieve $\theta = \theta_{1,2}$ by selectively regularizing θ, i.e., selectively slowing down learning on important parameters. After learning each task, EWC computes the important parameters by using the diagonal elements of the Fisher information matrix F, which is defined as:

$$F_{z,ii} = \frac{1}{|M_z|} \sum_{x \in M_z} \left(\frac{\delta log P_\theta\left(\cdot|x\right)}{\delta \theta_i}\bigg|_{\theta=\theta_z^*} \right)^2 \tag{16}$$

where z represents the task number, M_z is training data containing trajectories from task z, $log P_\theta\left(\cdot|x\right)$ is the predicted output of the network with parameters θ^* given data $x \in M_z$. After the computation of the important parameters for task z, EWC penalizes any change to the network parameters according to their importance when learning a new task. The penalty is comprised by the regularization loss. When learning task z, the regularization loss looks like:

$$\mathcal{L}_{\text{reg}}(\theta) = \sum_{j=1}^{z-1} \left(\frac{\lambda}{2} \sum_i F_{j,ii} (\theta_i - \theta_{j,i}^*)^2 \right) \tag{17}$$

where θ_i represents the current network parameters, with i being the i-th parameter. $\theta_{j,i}^*$ represents the network parameters directly after learning task $z-1$, λ is a constant that describes how important the previous task is compared to the current task, and $F_{j,ii}$ are the diagonal elements of Fisher Information Matrix for task $z-1$.

The complete loss function is as follows:

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{PPO}}(\theta) + \mathcal{L}_{\text{reg}}(\theta) \tag{18}$$

where $\mathcal{L}_{\text{PPO}}(\theta)$ is the standard PPO loss function, as defined in [38]. Note that the regularization term is only added to the loss when the first task t_1 has been learned.

2) Online EWC: For every task learned in EWC, a Fisher Information matrix F and optimal parameters θ^* need to be saved. This makes the quadratic terms in the regularization loss (eq. 17) grow linearly with the number of tasks, which is not practical due to storage and computation issues. Online EWC [30] overcomes this problem by keeping only one Fisher information matrix and one set of parameters while learning multiple tasks. When learning task z, the modified regularization loss is described as:

$$\mathcal{L}_{\text{reg}}(\theta) = \frac{\lambda}{2} \sum_i \tilde{F}_{z-1,i} (\theta_i - \theta_{z-1,i}^*)^2 \tag{19}$$

where $\theta_{z-1,i}^*$ are the optimal parameters directly after learning task $z-1$ and $\tilde{F}_{z-1,i}$ is the overall Fisher, calculated as the running sum of all Fisher Information matrices:

$$\tilde{F}_z = \gamma_{ewc} \tilde{F}_{z-1} + F_z \tag{20}$$

in which $\gamma_{ewc} < 1$ is a discount factor that gradually decays the contribution of older tasks, $\tilde{F}_{z-1}$ is the overall Fisher matrix after presentation of $z-1$ tasks, and F_z is the Fisher of the current task. After learning task $z = 1$, the overall Fisher Information matrix becomes $\tilde{F}_1 = F_1$.

E. Training algorithm

This paper aims to learn a collision-avoidance policy in a sequential setting for navigation in multiple scenarios containing static and dynamic obstacles. To accomplish this, we extend the RL training algorithm of [9], called GO-MPC, with EWC (Sec. IV-D). The training of the guidance policy π_θ^π and the value function $V_{\theta^V}(\mathbf{s})$ with MPC and EWC is described in Algorithm 2. There are two phases used in the training process. The first phase uses supervised learning with MPC as an expert to initialize the network parameters and thus accelerate policy convergence. The second phase uses reinforcement learning to further train our policy with Proximal Policy Optimization (PPO)[38]. As opposed to GO-MPC, we divide the RL phase into multiple tasks to create a continual learning setting. After the policy is trained on task 1, we adapt PPO training with additional regularization terms (Eq. 18), such that knowledge on previous tasks is consolidated when learning new tasks.

V. Results

This section presents experiments and results validating our proposed approach. First, we compare navigation performance between the two static obstacle representations proposed in our network (Sec. V-B). Next, we evaluate our continual reinforcement learning strategy. The evaluation includes an ablation study (Sec. V-F) and a comparison of our proposed approach against two baselines (Sec. V-G-V-H).

Algorithm 2 EWC-GO-MPC

Input: Task sequence $\{t_1, t_2, \cdots, t_z\}$, planning horizon H, value fn. and policy parameters $\{\theta^V, \theta^\pi\}$, number of supervised and RL training episodes $\{n_{MPC}, n_{episodes}\}$, number of agents n, $n_{\text{mini-batch}}$, and reward function $R(\mathbf{s}_t, \mathbf{a}_t, \mathbf{a}_{t+1})$
Initialize states: $\{s_0^0, ..., s_0^n\} \sim S, \{\mathbf{g}^0, ..., \mathbf{g}^n\} \sim S$
while episode $< n_{\text{episodes}}$ **do**
  for task $= t_1, t_2, ..., t_z$ **do**
    for $k = 0, ..., n_{\text{mini-batch}}$ **do**
      if episode $\leq n_{\text{MPC}}$ **then**
        Solve Eq. 15 with $\mathbf{p}^{\text{ref}} = \mathbf{g}$
        Set $\mathbf{a}_t^* = \mathbf{x}_N^*$
      else
        $\mathbf{p}^{\text{ref}} = \pi_\theta(\mathbf{s}_t, \mathbf{S}_t, \mathbf{O}_t^{\text{static}})$
      end if
      Rollout = Step$(\mathbf{s}_t^*, \mathbf{a}_t^*, \mathbf{h}_t)$
      Store rollout and collect trajectories for the current task M_z
    end for
    if episode $\leq n_{\text{MPC}}$ **then**
      Supervised training
    else
      if task > 1 **then**
        PPO training with Eq. 18
      else
        PPO training [39]
      end if
    end if
    Compute Fisher Matrix F_z with M_z (Eq. 16)
    Save the current weights $\theta_z^* = \theta_i$
  end for
end while
return $\{\theta^V, \theta^\pi\}$

A. Implementation details

In this work, we implement the model in Python using TensorFlow. For training with reinforcement learning, we use PPO from Stable-Baselines [39]. With a time-step of 0.2 seconds, we query a policy that provides a subgoal. Using the provided subgoal, the MPC planner solves the optimization problem in Eq. 15 using the Forces Pro solver [40]. We consider a maximum of four linear constraints to compute the collision-free space for the robot. A computer with an Intel Core i7 and 32 GB of RAM is used to train the neural networks. The hyper-parameters are presented in Table I.

B. Performance comparison static obstacle representations

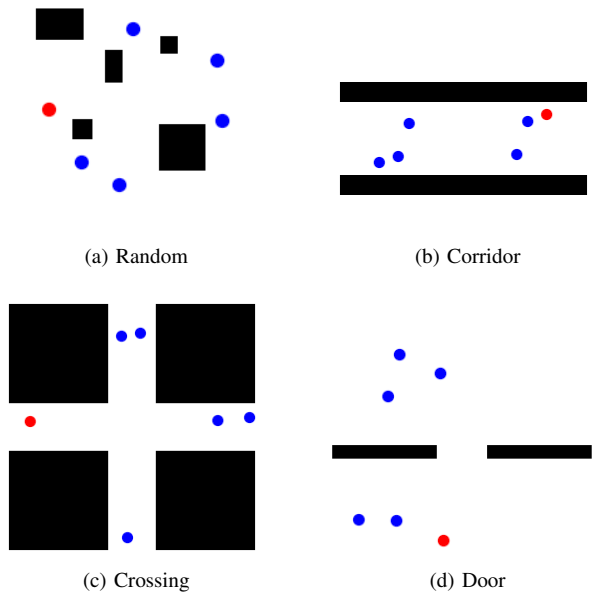
In this section, we compare navigation performance between the two proposed static obstacle representations: the occupancy grid and the angular map. We do not consider continual learning for these experiments since we only want to validate the static obstacle representations.

1) Training setup: In order to obtain a generalized policy that can handle different scenarios, we train on multiple scenarios with static obstacles, as depicted in Fig. 6. For scenarios b, c, and d, the static obstacles (marked as black) are always at a fixed location. This means that reasonable start and goal locations for all agents are defined. In scenario a, the obstacle locations, and therefore the agents' start and goal locations, are randomly generated.

We train the scenarios in the following order: a-b-c-d, but older scenarios are randomly repeated throughout the training process. At the start of the training, we initialize the network in a supervised learning phase, where we only use one random obstacle. Then we increase the number of obstacles (from 1 to 5) and agents (from 2 to 4) in the RL phase (curriculum learning). Next, the policy is trained for 2.5^6 steps on the corridor scenario, crossing scenario and door scenario (in this order) with five other agents. All agents (except the ego agent) use the RVO policy [16] to navigate to their goal.

TABLE I: Hyper-parameters

Parameter	Value	Parameter	Value
Learning rate	10^{-4}	r_t	-0.01
γ	0.99	Size occ. grid	60x60
Clip factor	0.1	Resolution occ. grid	0.1
Planning Horizon N	2 s	Slices ang. map	16
No. mini batches	2048	$[\alpha_{min}, \alpha_{max}]$	$[-\pi, \pi]$
r_{goal}	3	r_{max}	6
$r_{collision}$	-10	EWC batch size	50
r_{near}	-0.1-0.5*d_{min}	γ_{ewc}	0.99



(a) Random (b) Corridor (c) Crossing (d) Door

Fig. 6: The static obstacle scenarios used to train the collision avoidance policy. Obstacles are shown in black, the ego-agent is shown in red and the other agents are shown in blue. (a) Random obstacle scenario, where the locations of the obstacles are randomly generated. (b,c,d) corridor, crossing, and door scenario, respectively. The obstacle locations in these scenarios are fixed.

TABLE II: Comparison of the network trained with the occupancy grid (Occ.) and the angular map (Ang.) in multiple scenarios. Statistics are the average over 100 episodes, where time and distance to goal are computed for successful episodes, and %Failures are the percentage of episodes resulting in either a collision (c) or a deadlock (d).

Scenario	Repr.	Time to goal (mean ± std)[s]	% Failures (%c/%d)	Traveled distance (mean ± std)[m]
Random	Ang.	12.49 ±3.62	6(0/6)	10.65± 4.44
	Occ.	11.06 ± 3.19	4(0/4)	9.22 ± 3.58
Corridor	Ang.	12.83 ± 3.78	0(0/0)	12.41 ±2.30
	Occ.	12.61 ± 3.62	0(0/0)	12.15 ± 2.57
Crossing	Ang.	13.73 ± 3.96	0(0/0)	13.03± 1.73
	Occ.	13.89 ± 4.02	0(0/0)	13.08 ± 2.28
Door	Ang.	22.96 ± 6.84	50(0/50)	11.18 ± 3.92
	Occ.	18.17 ± 5.21	24(0/24)	11.57 ± 3.05

2) Quantitative performance: To test the trained policies, we use the four scenarios in Figure 6. Three metrics measure performance: 1) average time to reach the goal, 2) the percentage of episodes resulting in failures (timeout or collision), and 3) the average distance traveled to the goal.

Quantitative results of both trained networks (angular map and occupancy grid) are shown in Tab. II. The occupancy grid outperforms the angular map in all scenarios, except for the crossing scenario. The angular map performs only slightly better in the crossing scenario, but the difference is not significant. Moreover, the high number of deadlocks in the door scenario of both representations indicates that this scenario is challenging. We attribute this to the small passage available to many agents. In Appendix C, we further elaborate on this scenario.

We will use the occupancy grid representation for further experiments with static obstacles due to the better performance.

3) Qualitative performance: Qualitative results on testing in the four scenarios using the occupancy grid representation are shown in Fig 8. In all scenarios, the ego-agent (blue) is able to safely navigate to its goal while avoiding static obstacles and dynamic agents. In the corridor scenario, the agent is conservative by immediately navigating to the side of the corridor. The agent has learned that this leads to the highest rewards, and therefore it shows conservative behavior.

Next, we present experiments with our proposed continual reinforcement learning strategy.

C. Evaluation details CRL experiments

To evaluate the performance of our policy trained with continual reinforcement learning, we compare four approaches:

- **EWC:** Proposed method, as described in IV-D.1.
- **Online EWC:** Proposed method, as described in IV-D.2.
- **Vanilla baseline:** a model trained similarly as EWC, but without regularization terms.
- **Joint baseline:** a model trained jointly on all tasks: it repeats older tasks while learning new tasks. This approach assumes to have access to all data and therefore it does not satisfy a continual learning setting.

This paper aims to train a model on multiple tasks while maintaining a continual learning setting, where tasks are only sequentially available. We want to maintain this setting to get closer to realistic scenarios, in which non-stationary data only arrives sequentially and re-sampling is impossible. The vanilla baseline and our approaches satisfy this setting, but the joint baseline does not since it has access to all data. Still, we compare against the joint approach and consider it as an upper bound as acknowledged in continual learning [34][41].

D. Training procedure

To properly assess the performance of continual learning approaches, multiple continual learning benchmarks exist [10]. These benchmarks define a set of tasks that can be learned sequentially. The amount of forgetting can easily be estimated by evaluating the model's performance on previous tasks. There are many benchmarks for supervised learning approaches [14], but benchmarks for continual reinforcement learning are less available. Some focus on object-picking tasks [34], where each new task is a different object, others focus on the robot's dynamics [42], where each new task involves a change in the dynamics model. However, continual reinforcement learning is not yet applied for robot navigation in crowded environments. To allow evaluation of the continual learning performance during robot navigation, we consider a set of tasks with an increasing number of agents (Fig. 7):

- **Task 1:** 2 agents in the environment swapping positions.
- **Task 2:** 4 agents in the environment swapping positions with an agent on the opposite side. Each agent's initial and goal position are defined around the origin of the $\mathbb{R}^2$ x-y plane.
- **Task 3:** 6 agents in the environment swapping positions with an agent on the opposite side.

The changes in the task happen at predefined points equidistant in time. The tasks are learned sequentially, that is task 1 $\rightarrow$ task 2 $\rightarrow$ task 3. To accelerate policy convergence, the training is initialized with a supervised learning phase on the first task. We train in the SL phase for 1^5 steps. Thereafter, each task is trained for 1^6 steps. The other agents navigate to their generated goal by either using RVO policy [16] or a non-cooperative policy.

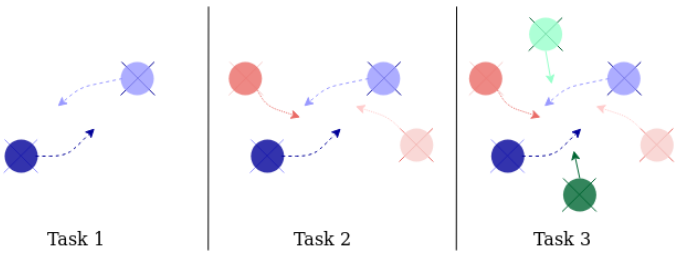
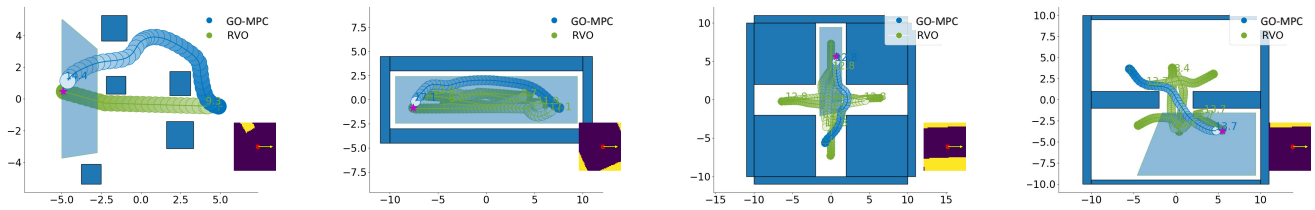


Fig. 7: The continual learning setting consists of three tasks: 2 agents, 4 agents and 6 agents swapping positions.



(a) Random (b) Corridor (c) Crossing (d) Door

Fig. 8: Sample trajectories in four static obstacle scenarios (random, corridor, crossing, door). In all scenarios, the ego-agent using our policy is seen in blue and the other agents using RVO policy are seen in green. The transparent blue area represents the free space computed from the linear constraints. The local occupancy grid is seen on the side, with obstacles in yellow.

E. Metrics

For the navigation performance, the metrics as explained in Section V-B.2 are used (time and distance to goal, and %failures). Continual learning performance is evaluated with Accuracy (ACC) and the Backward Transfer (BWT), as in [43]. ACC is the average cumulative rewards across n tasks. BWT describes the influence of learning new tasks on older tasks. If $BWT < 0$, it directly indicates catastrophic forgetting, $BWT > 0$ reports that learning new tasks helps on improving performance on older tasks. ACC and BWT are computed as follows:

$$\begin{aligned} ACC &= \frac{1}{z}\sum_{i=1}^{z} R_{i,z} \\ BWT &= \frac{1}{z}\sum_{i=1}^{z} R_{i,z} - R_{i,i}, \end{aligned} \tag{21}$$

where $R_{i,z}$ is the total reward on task i after finishing training on task z, $R_{i,i}$ is the total reward on task i after training on task i.

F. Ablation Study

The hyper-parameter λ determines the importance of the previous tasks compared to the new tasks.Here, we study the impact of λ on the continual learning performance. The hyper-parameter search was done with $\lambda = [4, 40, 400]$. Results on accuracy and backward transfer in Table III show that a lower λ leads to higher accuracy in navigation tasks, which is expected since a higher λ finds older tasks more important. For EWC, the backward transfer is lowest with a small λ since important parameters for previous tasks are constrained less. Aiming to reach the highest performance, we select $\lambda = 4$ for the subsequent experiments.

G. Quantitative performance

To compare our approaches against baselines, we test all four approaches on the three tasks used during training (Fig. 7). We perform the tests in two settings: an RVO setting and a mixed setting. In the RVO setting, all other agents employ the RVO policy. In the mixed setting, the other agent's policy can either be RVO policy (80%) or non-cooperative policy (20%). Quantitative performance results across 100 test episodes are summarized in Table IV. Note that the navigation performance results are averaged over all three tasks. We refer the reader to Appendix C.2 for the individual results on each task.

TABLE III: Ablation study on the influence of λ, with $\lambda \in \{4, 40, 400\}$, on the continual learning performance. Results are averaged over statistics on 100 episodes in three scenarios (2, 4, and 6 agents). $\lambda = 4$ leads to highest accuracy in EWC and Online EWC.

	EWC		Online EWC	
	ACC	BWT	ACC	BWT
λ = 4	13.47	-0.012	13.61	0.055
λ = 40	13.38	-0.074	13.44	0.014
λ = 400	13.37	0.073	13.37	0.050

RVO setting: The quantitative results show that our methods significantly outperform the baselines in terms of navigation performance and backward transfer. The low BWT (-3.42) of Vanilla directly indicates catastrophic forgetting. All other approaches achieve a higher value for ACC and BWT, suggesting that these approaches can mitigate forgetting on previous tasks while learning new tasks. Moreover, the navigation performance in terms of time and traveled distance to goal is better for our methods compared to the baselines. The low travel times and distances indicate that the robot's behavior in our methods is less conservative than in the baselines. Remarkably, although our approaches outperform the joint baseline in terms of all navigation metrics, the ACC of the joint baseline is higher. The reward function can explain the latter. A negative reward (r_{near}) is given for getting too close to another agent. Since the ego-agent is less conservative in our methods, it gets close to other agents, leading to more negative rewards and a slightly lower ACC.

Mixed setting: As shown in table IV, the performance of all approaches in terms of the proposed metrics is lower in the mixed setting than in the RVO setting. This decrease in performance is expected since non-cooperative agents do not contribute to collaborative collision avoidance, resulting in more complex scenarios. Our approaches still significantly outperform the vanilla baseline in terms of navigation and continual learning performance in the mixed setting. However, our approaches do not outperform the joint baseline in the number of collisions. Opposite to the RVO setting, the

TABLE IV: Comparison of the four different approaches in a setting with only RVO agents and a mixed setting (cooperative and non-cooperative agents). Results are averaged over 100 episodes on three tasks (2, 4 and 6 agents). ACC is the accuracy in terms of rewards, BWT indicates the amount of forgetting. The time and distance to goal are computed for successful episodes. %Failures are the percentage of episodes resulting in either a collision or a deadlock.

	ACC	BWT	Time to goal (mean ± std)[s]	% Failures (% collisions / % deadlocks)	Traveled distance (mean ± std)[m]
RVO agents					
Vanilla	9.56	-3.42	19.57 ± 5.75	15 (0 / 15)	17.39 ± 7.00
Joint	13.97	-0.14	16.14 ± 4.70	0 (0 / 0)}	15.19 ± 4.05
EWC	13.47	-0.012	**13.82 ± 3.99**	0 (0 / 0)	**12.92 ± 3.01**
Online EWC	13.61	0.055	**13.73 ± 3.97**	0 (0 / 0)	**12.83 ± 2.91**
Mixed agents					
Vanilla	4.01	-4.21	19.19 ± 5.68	20 (5 / 15)	17.04 ± 7.45
Joint	**9.30**	0.46	15.87 ± 4.63	5 (5 / 0)	14.92 ± 4.45
EWC	6.20	0.037	**13.20 ± 3.82**	8 (8 / 0)	**12.30 ± 3.61**
Online EWC	5.50	0.03	**13.14 ± 3.80**	8 (8 / 0)	**12.29 ± 3.51**

less conservative behavior of the robot results in a higher percentage of collisions and hence lower ACC. Thus, the less conservative behavior is not directly advantageous in mixed agent settings.

To give a better insight into the amount of forgetting in all approaches, we evaluate the accuracy in terms of rewards over the course of learning on task 1 (2-agent swapping scenario). For every 1^5 number of steps during training on tasks 1, 2 and 3, the four approaches are tested across 50 episodes on task 1 (see Figure 9). In the first section, the average rewards increase as the models are trained on task 1, confirming that our model is learning. A similar seed causes the performance in all four approaches to be exactly the same, which allows a fair comparison. However, when the models start training sequentially on tasks 2 and 3, the performance of the vanilla baseline (green line) drops drastically. The performance of the joint baseline also drops, but only slightly. The degradation in performance on task 1 while training on new tasks directly indicates catastrophic forgetting. The red (EWC) and the blue line (online EWC) show to be able to retain performance on task 1 throughout learning a sequence of tasks. Therefore, catastrophic forgetting can be alleviated by selectively constraining the network's parameters important for previously learned tasks.

Finally, considering the results of our two methods, there is not a significant difference in the evaluated performance between online EWC and EWC, although the higher memory costs of the latter.

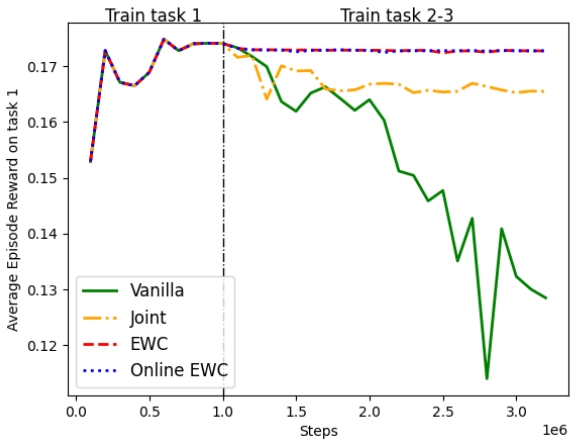


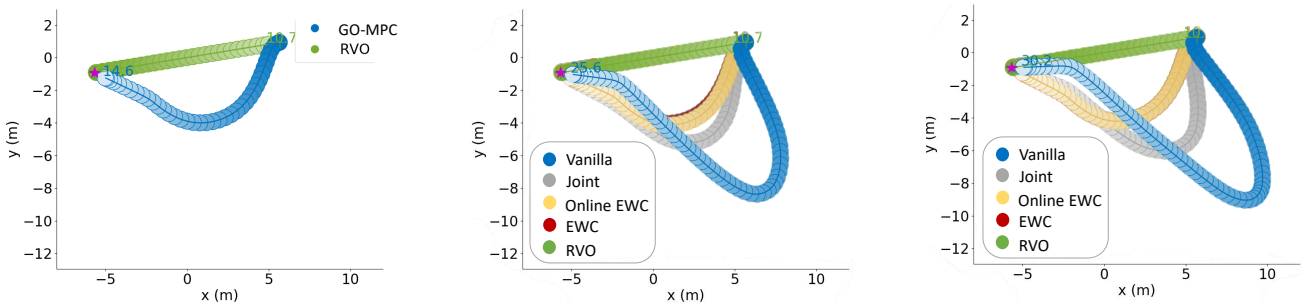
Fig. 9: Average episode reward on task 1 (2-agents swapping positions) across 50 episodes for models trained with vanilla (green), joint (yellow), EWC (red), and Online EWC (blue). The other agents have RVO policy.

H. Qualitative performance

In this section, we compare and analyse the trajectories of the four approaches in scenarios with two agents (task 1) and six agents (task 3) in the RVO setting.

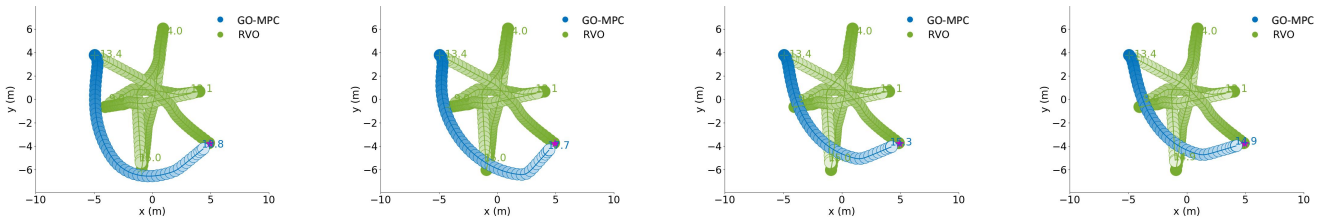
First, we analyse the trajectories on task 1 over the course of learning on other tasks to evaluate the preservation of performance (as we did in Fig. 9). Figure 10 and corresponding Table V show the change in trajectories in a 2-agent scenario (task 1) throughout sequentially training on task 1, 2 and 3. The trajectories of the vanilla baseline (blue markers) and the joint baseline (grey markers) in task 1 increase in length while training the model sequentially on tasks 2 and 3. The reason for this is as follows. Task 2 and 3 contain more agents, causing congestion in the middle. To achieve high rewards, the vanilla and joint approach learn a conservative behavior that avoids congestion by going around. However, by still applying this conservative behavior in task 1, the performance diminishes in this task. The trajectories of EWC (red) and online EWC (yellow) do not increase in length while training on new tasks, i.e., EWC and online EWC have the ability to retain performance on previous tasks.

Qualitative performance results in a 6-agent scenario (task 3) are depicted in Fig. 11. Similar to the quantitative results, the 6-agent setting shows that the trajectories of our methods are shorter than the baselines due to less conservative behavior. The less conservative behavior arises from the first task, in which the agent can swap positions with the other agent without deviating from its trajectory too much. In the consecutive tasks, the ego-agent has to avoid more agents (4 and 6) but retains the behavior learned in the first task, meaning that the agent's trajectory does not deviate significantly.



(a) Trained on task 1 (b) Trained on task 1 and 2 (c) Trained on task 1, 2, and 3

Fig. 10: Comparison of the trajectories of baselines (vanilla and joint) and our proposed approaches (EWC and Online EWC) on task 1 (2 agents swapping positions). (a) Trajectory of all four approaches directly after training on task 1. The trajectory is the same for all approaches due to a similar seed. (b) Trajectories after all four approaches are trained on task 1 and 2 sequentially. The trajectories of the baseline approaches increase in length. (c) Trajectories after training sequentially on all tasks. The trajectories of the baselines have increased in length, indicating catastrophic forgetting. Our approaches can retain performance on previous tasks.



(a) Vanilla (b) Joint (c) EWC (d) Online EWC

Fig. 11: 6 agents swapping positions, where the green agents use RVO policy and the blue agent is the ego-agent. EWC and Online EWC generate less conservative behavior than the baselines, resulting in shorter trajectories.

TABLE V: Time to reach the goal (seconds) in a 2-agent swapping scenario. (a) trained on task 1, (b) trained on task 1 and 2 (c) trained on task 1, 2, and 3. These result correspond to the trajectories in Fig. 10.

	Time to goal [s]			
	a	**b**	**c**	**Total gain [s]**
Vanilla	14.6	25.6	30.2	15.6
Joint	14.6	16.8	19.4	4.8
EWC	14.6	14.6	14.9	0.3
Online EWC	14.6	14.8	14.9	0.3

VI. Real-world experiments

In this section, we present experimental results demonstrating the performance of our proposed network in real-world environments. The generalization from a learned policy in simulation to realistic environments is often challenging due to the sim-to-real gap. This gap is caused by discrepancies between the simulated environments and the real-world environments, e.g., perception errors, different human behaviors, sensor noise. Compared to approaches using only DRL, we already reduce the sim-to-real gap in terms of robot dynamics by the use of an optimizaton-based controller (MPC). However, our learned policy still suffered the sim-to-real gap in a realistic corridor scenario (Appendix D). To bridge this gap, we trained a policy in simulated environments with less discrepancies to the real-world corridor scenario.

We use ROS to integrate our method on a real robot. To validate our ROS integration, we first present results in simulated real-world environments. Then, we present results in real-world environments.

A. Simulated real-world environments

For the purpose of navigation in a real-world corridor scenario, we simulate a corridor with 6 pedestrians. The behavior of the pedestrians is simulated with the pedsim simulator[1], which is an open-source framework based on the social force model. Exact positions and velocities of the other agents are provided, eliminating perception-related errors in these experiments. Fig. 12 shows a qualitative example of the experiment. In Fig. 12a, the subgoal reacts to the surrounding agents when navigating to the final goal (red arrow). MPC is able to generate safe motion plans towards the provided subgoal. Fig. 12b shows that the learned policy tends to recommend a subgoal on the side of the corridor since the policy has learned that this results in high rewards.

These experiments validate the integration of our approach on the real robot.

[1]https://github.com/srl-freiburg/pedsim_ros

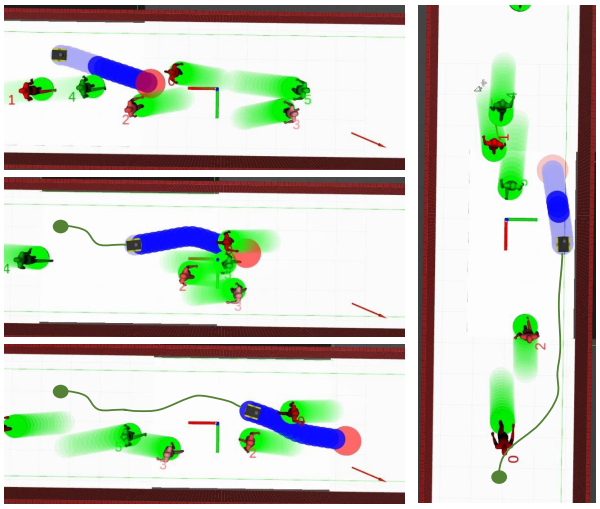


Fig. 12: Snapshots of experiment in simulated real-world corridor scenario. The robot navigates to the final goal (red arrow) by the MPC control commands (blue circles) following the intermediate subgoals (red circle) recommended by the learned policy. The green line indicates the traveled trajectory.

B. Real-world environments

Here, we present experiments of our approach in uncontrolled environments, where the ego-agent is subject to noise and perception errors. The uncontrolled environment is a university corridor at TU Delft. While humans walk through the corridor, our robot has to navigate to a predefined goal position. Other agents in the environment are tracked using the robots detection and tracking pipeline. Ultimately, according to the robots observations, the learned policy provides a subgoal to the MPC planner enhancing the navigation performance and avoiding deadlocks. Moreover, the MPC planner is expected to ensure collision avoidance and kinodynamic constraints (for feasible solutions).

Fig. 13 shows snapshots of the robot operating in the university corridor. Similar to the simulated real-world experiments, a subgoal is provided according to the other agents' states (Fig. 13). However, we observed that the behavior of the robot was not always robust in real-world experiments. In many cases, the subgoal was recommended towards the side of the corridor since the policy achieved highest rewards with this behavior during training. Therefore, the behavior of the robot is dependent on the learned policy.

Although we have shown that in some cases the robot was able to navigate in a real-world corridor, in many other cases the robot did not properly navigate to its goal. For instance, when the robot gets close to a human, it only stops just before collision instead of avoiding the human more in advance. The latter slows down the robot a lot. Several reasons could cause this bad navigation behavior. One reason could be sim-to-real gap in terms of other agents' behavior. In simulation, the behavior was modeled with either RVO policy or a non-cooperative policy. However, in the real world, human behavior is not similar to RVO or non-cooperative policies, e.g., some pedestrians stop in front of the robot. Another reason could be due to perception-related errors of the detection and tracking pipeline of the robot. Errors in the observations of the environment can influence the policy, but also the motion plans generated by MPC.

Finally, to appropriately evaluate performance in real-world environments, improvements to our framework need to be done. Improvements include learning a proper policy for real-world environments, the perception and tracking pipeline and checking the complete integration on the real-robot. We leave this to future work.

VII. Conclusion

This study presents a novel DRL network architecture that incorporates static obstacle representations to navigate in realistic human environments. By combining our proposed network with MPC, we validate that our model can safely navigate in multiple environments with static and dynamic obstacles satisfying collision avoidance constraints. Moreover, we show that our network has the ability to continually learn a sequence of tasks with increasing difficulty without forgetting previously learned tasks by employing EWC in our network. Our CRL framework can significantly reduce the time and distance to the goal in settings with cooperative agents and mixed settings.

References

[1] Brian Paden, Michal Cap, Sze Zheng Yong, et al. "A Survey of Motion Planning and Control Techniques for Self-driving Urban Vehicles". In: (2016). arXiv: 1604.07446 [cs.RO].

[2] Bruno Brito, Boaz Floor, Laura Ferranti, et al. "Model Predictive Contouring Control for Collision Avoidance in Unstructured Dynamic Environments". In: (2020). arXiv: 2010.10190 [cs.RO].

[3] Yu Fan Chen, Shih-Yuan Liu, Miao Liu, et al. "Motion planning with diffusion maps". In: (2016), pp. 1423–1430. DOI: 10.1109/IROS.2016.7759232.

[4] Yu Fan Chen, Michael Everett, Miao Liu, et al. "Socially Aware Motion Planning with Deep Reinforcement Learning". In: (Mar. 2017). URL: http://arxiv.org/abs/1703.08862.

[5] Changan Chen, Yuejiang Liu, Sven Kreiss, et al. "Crowd-Robot Interaction: Crowd-aware Robot Navigation with Attention-based Deep Reinforcement Learning". In: (Sept. 2018). URL: http://arxiv.org/abs/1809.08835.

[6] Michael Everett, Yu Fan Chen, and Jonathan P. How. "Motion Planning Among Dynamic, Decision-Making Agents with Deep Reinforcement Learning". In: (2018). arXiv: 1805.01956 [cs.RO].

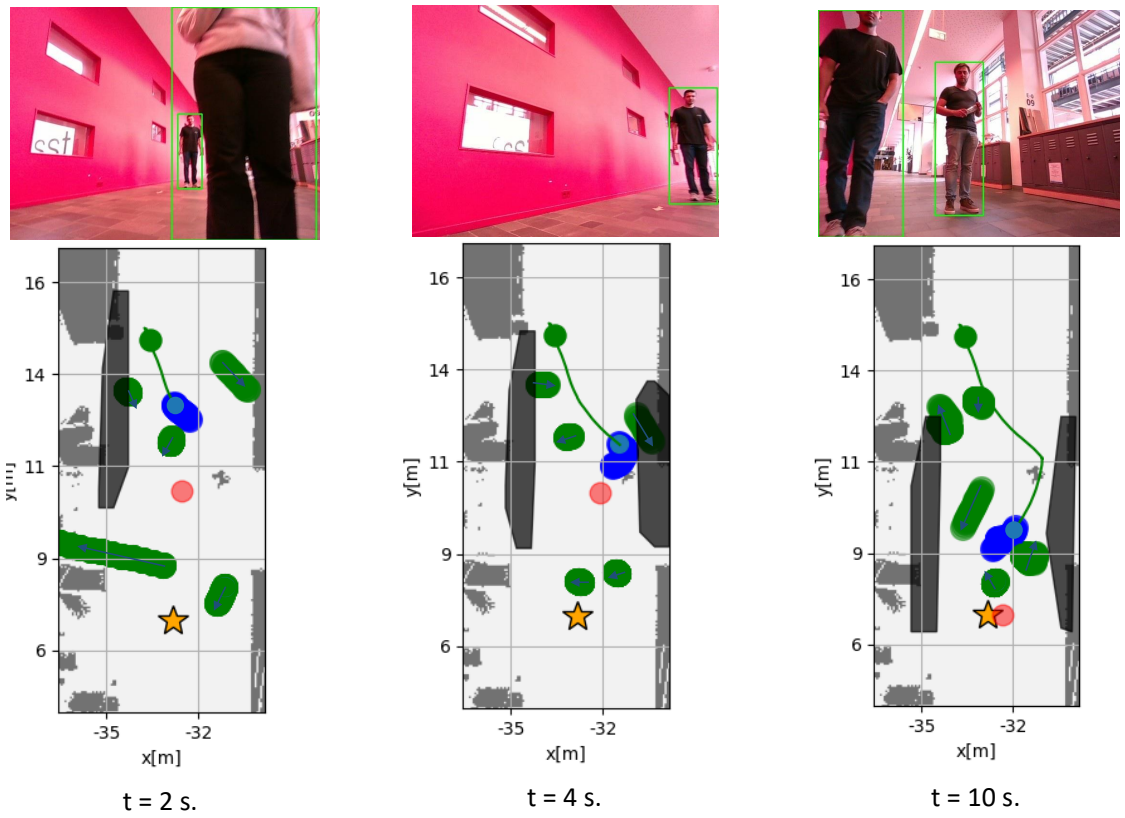


Fig. 13: Snapshots of robot trajectories in a real-world corridor scenario. The top row shows the robot's front camera images including detected pedestrians. The bottom row shows the trajectories at different instances in time. The green line represents the driven trajectory. The green markers are the detected pedestrians and their predicted trajectories, in which the blue arrow indicates the direction. The red marker is the provided subgoal and the yellow star is the final goal. The blue markers represent the MPC generated plan.

[7] Tingxiang Fan, Pinxin Long, Wenxi Liu, et al. "Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios". In: *The International Journal of Robotics Research* 39 (7 2020), pp. 856–892. DOI: 10.1177/0278364920916531. URL: https://sites.google..

[8] Damien Ernst, Mevludin Glavic, Florin Capitanescu, et al. "Reinforcement Learning Versus Model Predictive Control: A Comparison on a Power System Problem". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 39.2 (2009), pp. 517–529. DOI: 10.1109/TSMCB.2008.2007630.

[9] Bruno Brito, Michael Everett, Jonathan P. How, et al. "Where to go next: Learning a Subgoal Recommendation Policy for Navigation in Dynamic Environments". In: (2021).

[10] German I. Parisi, Ronald Kemker, Jose L. Part, et al. "Continual lifelong learning with neural networks: A review". In: *Neural Networks* 113 (May 2019), pp. 54–71. ISSN: 18792782. DOI: 10.1016/j.neunet.2019.01.012.

[11] Josh Tobin, Rachel Fong, Alex Ray, et al. "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World". In: *IEEE International Conference on Intelligent Robots and Systems* 2017-September (Mar. 2017), pp. 23–30. arXiv: 1703.06907. URL: http://arxiv.org/abs/1703.06907.

[12] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, et al. "Asynchronous Methods for Deep Reinforcement Learning". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1928–1937. URL: http://proceedings.mlr.press/v48/mniha16.html.

[13] Timothée Lesort, Vincenzo Lomonaco, Andrei Stoian, et al. "Continual Learning for Robotics: Definition, Framework, Learning Strategies, Opportunities and Challenges". In: (June 2019). URL: http://arxiv.org/abs/1907.00182.

[14] Matthias Delange, Rahaf Aljundi, Marc Masana, et al. "A continual learning survey: Defying forgetting in

classification tasks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), pp. 1–1. DOI: 10.1109/TPAMI.2021.3057446.

[15] Thibault Kruse, Amit Kumar Pandey, Rachid Alami, et al. "Human-aware robot navigation: A survey". In: *Robotics and Autonomous Systems* 61.12 (2013), pp. 1726–1743. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2013.05.007.

[16] J. van den Berg, Ming Lin, and D. Manocha. "Reciprocal Velocity Obstacles for real-time multi-agent navigation". In: *2008 IEEE International Conference on Robotics and Automation*. 2008, pp. 1928–1935. DOI: 10.1109/ROBOT.2008.4543489.

[17] Jur van den Berg, Stephen Guy, Ming Lin, et al. "Reciprocal n-Body Collision Avoidance". In: vol. 70. Apr. 2011, pp. 3–19. ISBN: 978-3-642-19456-6. DOI: 10.1007/978-3-642-19457-3_1.

[18] Jing Liang, Utsav Patel, Adarsh Jagan Sathyamoorthy, et al. "Realtime Collision Avoidance for Mobile Robots in Dense Crowds using Implicit Multi-sensor Fusion and Deep Reinforcement Learning". In: (Apr. 2020). URL: http://arxiv.org/abs/2004.03089.

[19] Yu Fan Chen, Miao Liu, Michael Everett, et al. "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning". In: May 2017, pp. 285–292. DOI: 10.1109/ICRA.2017.7989037.

[20] Shuijing Liu, Peixin Chang, Weihang Liang, et al. "Decentralized Structural-RNN for Robot Crowd Navigation with Deep Reinforcement Learning". In: (2021). arXiv: 2011.04820 [cs.RO].

[21] Yuying Chen, Congcong Liu, Ming Liu, et al. "Robot Navigation in Crowds by Graph Convolutional Networks with Attention Learned from Human Gaze". In: (2019). arXiv: 1909.10400 [cs.RO].

[22] Zhang-Wei Hong, Joni Pajarinen, and Jan Peters. "Model-based Lookahead Reinforcement Learning". In: (2019). arXiv: 1908.06012 [cs.LG].

[23] M. Pfeiffer, G. Paolo, H. Sommer, et al. "A Data-driven Model for Interaction-Aware Pedestrian Motion Prediction in Object Cluttered Environments". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 5921–5928. DOI: 10.1109/ICRA.2018.8461157.

[24] Andrei A. Rusu, Mel Vecerik, Thomas Rothörl, et al. "Sim-to-Real Robot Learning from Pixels with Progressive Nets". In: (Oct. 2016). URL: http://arxiv.org/abs/1610.04286.

[25] Ju Xu and Zhanxing Zhu. "Reinforced Continual Learning". In: (May 2018). URL: http://arxiv.org/abs/1805.12369.

[26] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, et al. "iCaRL: Incremental Classifier and Representation Learning". In: ().

[27] Rahaf Aljundi, Min Lin, Baptiste Goujaud, et al. "Gradient based sample selection for online continual learning". In: (Mar. 2019). URL: http://arxiv.org/abs/1903.08671.

[28] Xisen Jin, Junyi Du, and Xiang Ren. "Gradient Based Memory Editing for Task-Free Continual Learning". In: (2020). URL: https://github.com/INK-USC/GMED.

[29] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, et al. "Overcoming catastrophic forgetting in neural networks". In: (Dec. 2016). URL: http://arxiv.org/abs/1612.00796.

[30] Jonathan Schwarz, Jelena Luketina, Wojciech M. Czarnecki, et al. "Progress Compress: A scalable framework for continual learning". In: (May 2018). URL: http://arxiv.org/abs/1805.06370.

[31] Friedemann Zenke, Ben Poole, and Surya Ganguli. "Continual Learning Through Synaptic Intelligence". In: *Proceedings of the 34th International Conference on Machine Learning*. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, June 2017, pp. 3987–3995. URL: http://proceedings.mlr.press/v70/zenke17a.html.

[32] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, et al. "Memory Aware Synapses: Learning what (not) to forget". In: *In Proceedings of the European Conference on Computer Vision (ECCV)* (Nov. 2018), pp. 139–154.

[33] Raia Hadsell, Dushyant Rao, Andrei A. Rusu, et al. "Embracing Change: Continual Learning in Deep Neural Networks". In: *Trends in Cognitive Sciences* 24.12 (2020), pp. 1028–1040. ISSN: 1364-6613. DOI: https://doi.org/10.1016/j.tics.2020.09.004.

[34] Vincenzo Lomonaco, Karan Desai, Eugenio Culurciello, et al. "Continual Reinforcement Learning in 3D Non-stationary Environments". In: (2020). arXiv: 1905.10112 [cs.LG].

[35] Steven M. LaValle. *Planning algorithms*. Cambridge University Press, 2006. ISBN: 0521862051. DOI: https://doi.org/10.1017/CBO9780511546877. URL: http://planning.cs.uiuc.edu/.

[36] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[37] Javier Alonso-Mora, Eduardo Montijano, Mac Schwager, et al. "Distributed multi-robot formation control among obstacles: A geometric and optimization approach with consensus". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 5356–5363. DOI: 10.1109/ICRA.2016.7487747.

[38] John Schulman, Filip Wolski, Prafulla Dhariwal, et al. "Proximal Policy Optimization Algorithms". In: (2017). arXiv: 1707.06347 [cs.LG].

[39] Ashley Hill, Antonin Raffin, Maximilian Ernestus, et al. *Stable Baselines*. 2018.

[40] Alexander Domahidi and Juan Jerez. *FORCES Professional*. embotech GmbH (http://embotech.com/FORCES-Pro). July 2014.

[41] Rahaf Aljundi, Klaas Kelchtermans, and Tinne Tuytelaars. "Task-Free Continual Learning". In: (Dec. 2018). URL: http://arxiv.org/abs/1812.03596.

[42] Anusha Nagabandi, Ignasi Clavera, Simin Liu, et al. "Learning to Adapt in Dynamic, Real-World Environments Through Meta-Reinforcement Learning". In: (Mar. 2018). URL: http://arxiv.org/abs/1803.11347.

[43] Sayna Ebrahimi, Mohamed Elhoseiny, Trevor Darrell, et al. "Uncertainty-guided Continual Learning with Bayesian Neural Networks". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=HklUCCVKDB.

3

Discussion

We train a neural network on multiple tasks with continual reinforcement learning in the domain of robot navigation. We were the first to apply a CRL strategy in this domain and show that adding regularization terms to the loss function generates the ability to learn new tasks without forgetting previous ones. Therefore, the conducted research helps progress towards an algorithm that can adapt its model during deployment when environmental conditions change. Moreover, by focusing on robot navigation in human environments, we extend an already existing algorithm to account for static obstacles.

In this section, we provide a thorough discussion on our conducted research. This is divided into a discussion on the addition of static obstacle representations, continual learning and real-world experiments.

3.1. Static obstacle representations

We have demonstrated to extend the approach of [3] by accounting for static obstacles in human environments. To incorporate static obstacles, we compared two representations, the occupancy grid and the angular map. The angular map has more benefits than the occupancy grid: the dimensionality is lower, it automatically infers more important obstacles by assigning more angular slices to closer obstacles, and it can be used in real-world experiments without the use of a global map. Despite these many benefits, our results show that representing the static obstacles with the occupancy grid achieves higher navigation performance in terms of travel time and distance and the number of failures. One possibility to improve the performance of the angular map could be to increase the number of angular slices to capture static obstacles more accurately and put more weight on the static obstacle input channel in the neural network. Although the occupancy grid representation has higher dimensionality, a Convolutional Neural Network decreases this significantly. In appendix B, we show that the CNN is able to learn a compressed representation of the provided input. However, we train the CNN during training of our complete network. This means that the static obstacle input channel does not provide helpful information in the initial training phase since the CNN is still being trained. Therefore, we encourage future research to use a pre-trained CNN while training a policy. The pre-trained CNN will provide the network with helpful information from the start, resulting in lower training times.

The drawback of adding static obstacle representations to our network is that the dimensionality increases, i.e., the number of input variables in the training data increases. Therefore, many more training steps are needed to make the training converge to a robust solution. Our algorithm showed the ability to complete navigation tasks in multiple environments constrained by static obstacles despite longer training times. This improvement makes our algorithm more applicable to real-world environments, in which static obstacles are not indispensable.

3.2. Continual learning

We employ continual learning in our training algorithm using the state-of-the-art continual learning methods EWC and online EWC. Both methods selectively penalize changes to the network's parameters according to their importance while learning new tasks. Online EWC has fewer computational and memory constraints, which is beneficial with the aim never to stop learning. When comparing both approaches, our results have shown no significant difference between the two methods, meaning that online EWC is the better choice. We have not only shown a comparison between these two methods, but also against baselines that do not use continual learning. Our results indicate that learning a sequence of tasks with selective regularization of the network's parameters mitigates catastrophic forgetting and leads to better navigation performance in terms of failures and travel times and distances.

Although we prove the continual learning ability in relative simple scenarios, this does not directly indicate similar performance on other scenarios. Therefore, we have also tested our approaches on a different sequence of tasks (Appendix C). We have trained the model on three scenarios with static obstacles similar to the ones used in our paper. The results show that our approaches still outperform the baselines in terms of navigation performance, but with less significance. This indicates that the performance of our approaches is dependent on the sequence of tasks. Moreover, in the experiments with different static obstacle environments, we observed that EWC did not retain performance as much as we showed on the tasks with an increasing number of agents. We attribute this to the value λ, because in literature it has been shown that for a different sequence of tasks, a different value of λ is used [20]. However, we did not change the value of λ for these experiments. Therefore, we encourage future research to investigate different values of λ on different tasks.

Finally, our choice was to employ a regularization-based continual learning approach to mitigate catastrophic forgetting. These approaches do not suffer from great computational and memory constraints compared to architecture-based and memory-based approaches. However, a drawback of regularization-based approaches is that, after learning several tasks, the introduced stability update, which is always directed towards the parameters of the previous network weights, can restrict the model to learn new tasks. In our experiments, in which we learn a sequence of three tasks, the performance of EWC and online EWC on the last task is not inferior to the two baselines, indicating that model saturation does not occur yet. However, when learning on a higher number of tasks, model saturation can occur. If this happens, the model stabilizes, such that it cannot adapt its weights to learn a new task. This issue is often referred to as the *stability-plasticity dilemma* [19]. By combining memory-based approaches with regularization-based approaches, this issue can be partially improved but increases the computational and memory limitations. More recommendations to overcome this issue are discussed in 4.2.

3.3. Real-world experiments

Finally, we implemented our approach in the Robot Operating System to enable mobile robot navigation in real-world environments. Our initial experiments used a learned policy that performed well in simulated environments with static obstacles. However, applying this policy to the real world did not result in good performance (see appendix D). We attribute this to the sim-to-real gap. To achieve good performance in the real world, we reduced the sim-to-real gap by training a policy in an environment more closely related to the real-world environment. For instance, the width of the real-world corridor was adopted in the simulated environments. By applying the newly learned policy, we saw performance improvements. For instance, the recommended subgoal reacted to the surrounding agents in the environment, which was not observed in the initial experiments. Therefore, the behavior of the robot is dependent on the learned policy. However, besides the improved performance of the robot in the real world, the robot's behavior was still not robust. This was not only due to the learned policy but also due to the MPC planner and the detection and tracking pipeline not being robust enough. Further research is required to improve the performance of the robot in the real world.

4

Conclusion

This is the concluding chapter of this thesis. First, we provide conclusions on our research based on the research questions presented in the introduction. Then, we present recommendations for future work.

4.1. Conclusions

RQ 1: How can we build on available algorithms using deep reinforcement learning to achieve improvements in the navigation performance in realistic unstructured, dynamic environments?
The growing interest in mobile robots operating in human environments has led to extensive research on robot navigation in these environments. The difficult challenge of navigation among humans with stochastic behavior has been taken by many algorithms. The recent algorithms combining deep reinforcement learning (DRL) with model predictive control (MPC) are able to obtain high performance in terms of safety and time to reach the goal among agents with different behaviors. The focus of this thesis was to extend these algorithms to be more applicable to realistic settings. We achieved this by expanding GO-MPC [3] with static obstacles. Not only did we add static obstacle representations to the neural network, but also to the MPC controller. In our experiments, we have shown that our algorithm could navigate a robot in four different environments with dynamic agents and static obstacles. Since static obstacles are not indispensable in real-world environments, we conclude that our proposed algorithm improves towards more realistic environments.

RQ 2: Can continual reinforcement learning be successfully applied to learning a robot navigation policy in human environments, and can it constitute an improved generalized model?
Many approaches for robot navigation that learn a policy with deep reinforcement learning increase the generalization ability of their model by training on several tasks with increasing difficulty. Although performing well on the most challenging tasks, the performance on the more simple tasks is diminished. To overcome this issue, we implemented a regularization-based continual learning approach (EWC) in our algorithm. Our experiments showed that we could learn a sequence of tasks with increasing difficulty and retain performance on all tasks by employing our proposed continual reinforcement learning strategy. This increases the generalization ability of our approach since we are not only able to perform well on the most challenging tasks but also on the more simple tasks.

Finally, our conducted research constitutes a step forward in continual reinforcement learning strategies. These strategies can be used to continually adapt the learned model to changes in the environment, resulting in a model with high generalization capabilities.

4.2. Recommendations

In this section, I will describe recommendations for future research. I divided the recommendations into two categories: future directions for static obstacle avoidance and recommendations for continual reinforcement learning.

4.2.1. Recommendations to extend the network architecture

More realistic occupancy grid map

The presented implementation of static obstacles only assumes rectangular or squared shaped obstacles. However, to increase the similarity to the real world, extensions could be made towards a greater variety of shapes, like triangles, circles, non-convex obstacles, etc. Another option would be to use real-world occupancy maps during training. The maps can be made from sensor data of the real robot. To get a wide variety of maps, a Generative Adversarial Network (GAN) [10] can be used, which can generate new plausible maps from a provided dataset. The sim-to-real gap decreases by simulating an environment similar to the real world, potentially leading to better performance in real-world scenarios.

Improve Convolutional Neural Network

The occupancy grid representation is processed by a Convolutional Neural Network (CNN), trained during our policy training. Another (minor) extension is to use a pre-trained CNN to process the occupancy grid that can speed up the training process.

Integration of a social Attention model

Lastly, to improve our network architecture, we could use an attention model for the dynamic agents in the environment, like in [4]. At the moment, the other agents' states are fed into the network by descending order of their distances to the robot. However, the assumption that the closest agent is always the most important agent is not always correct. The heading and velocity are also essential factors to consider when estimating the importance of surrounding agents. A social attentive module can learn the relative importance of surrounding agents to the robot [4].

4.2.2. Recommendations for continual reinforcement learning

Saving the optimal weights of a task

After training on a task, EWC saves the latest network weights to consolidate knowledge on that task. However, in reinforcement learning, the latest weights are not directly the most optimal weights. An improvement would be not to save the latest weights but the weights at the most optimal task performance. Although this might lead to extra memory costs, retaining the most optimal performance of the learned task improves the overall performance.

Weaken the stability bias

Learning a policy through GO-MPC-EWC is different from learning a policy with only GO-MPC. EWC introduces a stability update that is always directed towards the parameters of the previous network weights $\theta^*_{1:k-1}$. The stability update changes the policy and impedes the usual exploration strategy. Future research could focus on relaxing the stability bias created by EWC. One possibility for this is only to perform stability updates every h steps. When h increases, the stability update towards the weights of the previous tasks becomes weaker. This allows exploration every time there is no stability update. Another possibility to weaken the stability bias is considering λ as a variable instead of a constant. λ could be changed (decreased) according to the number of training steps or tasks encountered.

Besides the stability bias introduced by EWC, PPO with clipped gradients (as used in this thesis) also introduces a stability bias to address the high-variance issue of policy gradient methods. By clipping the objective function, PPO does not allow large updates between the old and the new policy. Both stability updates can create a stable learning algorithm but could remove the benefits of the exploration strategy. Therefore, an interesting direction for future research is to explore a variable clip factor and λ during learning the policy.

Changing the continual learning method

The continual learning method EWC [15] is a regularization-based approach often used in continual learning research. However, other methods have outperformed EWC regarding the accuracy and backward transfer, but primarily for classification. Future research can investigate if these methods can also outperform EWC in a reinforcement learning setting.

Moreover, EWC works in a task-based setting, where hard task boundaries are defined. However, in real-world situations, hard task boundaries are often not encountered. Therefore, a promising area for future research is to employ EWC in a task-free setting. To achieve this, an extra detection mechanism would be needed to detect a change in task. Several approaches have already worked on the transition from task-based settings to task-free settings [1][20].

Employing continual learning during deployment in the real-world

As stated in the introduction, to overcome performance degradation during robot deployment in the real world, models need the ability to adapt to the ever-changing environment. To accomplish this, continual learning can be used since it can learn from new data and consolidate knowledge from past data. In this research, we have shown the performance of continual learning in the training process of a model trained with DRL. A promising area for future research is to employ continual learning during robot deployment in the real world, such that the model can be adapted according to the current situation. However, the use of our model in real-world scenarios is limited due to the challenges of real-world reinforcement learning [22]. For example, real-world environments have extremely limited data as the acquisition of real data is slow and costly, which makes learning a challenge and could take a long time to converge. Also, since RL is a process of trial and error, it is difficult to maintain safety in human environments. Therefore, instead of using real-world reinforcement learning, it is interesting to investigate other learning techniques to adapt our trained model in ever-changing environments. For instance, to update the subgoal recommendation policy, supervised learning can be used, in which a detected pedestrian can be the expert for recommending new subgoals.

A

Network training

The DRL model used in this thesis combines pedestrian and static obstacle representations to navigate the robot through cluttered environments. To encode static obstacles, we use an occupancy grid representation and an angular map representation. For this evaluation, we trained the network without continual learning. However, we have also trained our network with continual learning. This Appendix describes the training setup and training results of both learned policies in more detail.

A.1. Comparing static obstacle representations

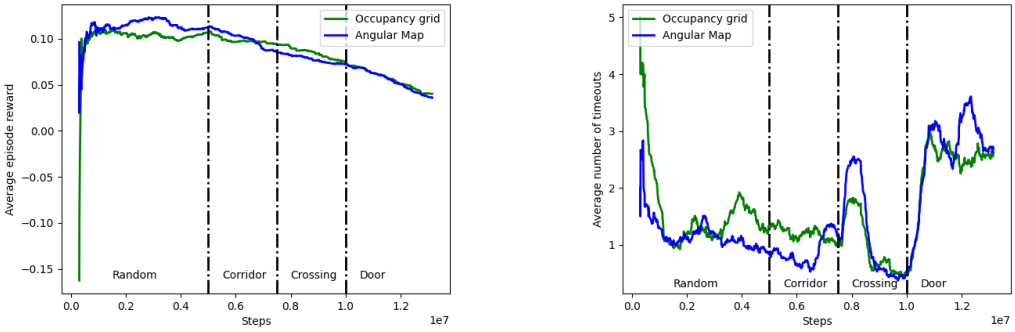
A.1.1. Training setup

Training a network with static obstacle representations requires much more training steps than training for only dynamic agents. The reason for this is that the observation space of the static obstacle representations is high-dimensional.

The network training was already explained in the provided paper. Here, we describe the training in more detail. The training is done in two different phases, a supervised learning phase and a reinforcement learning phase, The network is initialized in the supervised learning phase, in which we train on an environment with one random obstacle and two other agents for 3^5 steps. After that, the RL phase starts. First, environments with random obstacles are used. To speed up the training on multiple random obstacles, we use curriculum learning. For the curriculum learning setup, we start by training on one random obstacle for 1.5^6 steps, then we increase the number of random obstacles to 3 and the number of agents to 4 and train for another 1.5^6 steps. Lastly, we train for 2^6 steps in environments with up to 5 random obstacles of different sizes. After the random obstacles scenarios, the following scenarios are trained for 2.5^6 steps each: corridor scenario, crossing scenario and a door scenario. The total training time is, therefore, 12.5^6 steps.

A.1.2. Training results

In Fig. A.1 both the average episode reward and the number of timeouts per training step is visualized. It can be seen that the results for the occupancy grid and the angular map are almost similar. Moreover, the results in Fig. A.1b show clearly when the tasks changes to the crossing and door scenario. The number of timeouts in the door scenario increase a lot, confirming the high number of deadlocks in the navigation performance as shown in the paper. Also, the number of timeouts for the crossing and door scenarios is higher for the angular map. The latter could be the reason for the occupancy grid outperforming the angular map during testing.



(a) Average episode eward

(b) Number of successful events

Figure A.1: Comparison of moving average episode reward and number of timeouts for training the policy with occupancy grid representation (green) or angular map representation (blue).

A.2. Continual reinforcement learning

This section evaluates the training curves of the four approaches (vanilla, joint, EWC, and Online EWC) when sequentially training on 2, 4, and 6 agents. The training is executed with cooperative and non-cooperative agents, as explained in our paper. Figure A.2 shows the average episode reward during training of the four approaches used in this study. The plots start to diverge when training on tasks 2 and 3, which is expected since the models are exactly similar in task 1. We can see that the joint approach obtains the highest rewards throughout the training process. In the joint approach, older tasks are repeated, and thus rewards of task 1 and 2 are included in the rewards for task 3 (2^6 steps and onwards). Since tasks 1 and 2 are less challenging than task 3, higher rewards are obtained for these tasks. The other three approaches do not repeat older tasks, which could be why the rewards for these approaches are lower.

In our paper, we found that EWC and Online EWC significantly outperformed the vanilla approach in terms of rewards, failures, travel time and distance. However, the latter is not indicated by the training plot, in which we see that the reward for EWC, Online EWC and Vanilla drop similarly. A reason for this could be as follows. The training plot shows the results of the approaches being trained in the indicated task. However, the performance results shown in our paper are averaged over three tasks (1, 2 and 3). EWC and Online EWC retain knowledge on older tasks, meaning that even though the average rewards drop for training in a new task, the model keeps high performance on the previous tasks. Therefore, the results as presented in our paper do correspond to the training plot shown here. Note that this training plot includes cooperative and non-cooperative agents, whereas the results in our paper, in which EWC outperformed the joint approach, only used RVO agents.

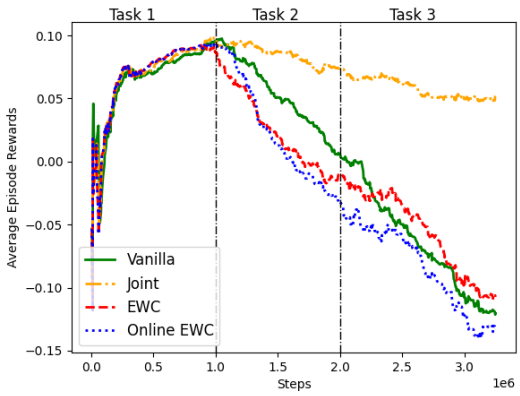


Figure A.2: Moving average of average episode rewards throughout training our CRL framework on 2 (task1), 4 (task2), and 6 (task3) agents. Four approaches are compared: vanilla, joint, EWC (ours) and Online EWC (ours).

B

Convolutional Neural Network (CNN)

During the training of the proposed network in this thesis, a Convolutional Neural Network (CNN) is trained simultaneously. In this appendix, the network architecture is explained in detail, and the training details are provided.

B.1. Network Architecture

The network architecture of the CNN used in this thesis is depicted in Figure B.1. The properties of the convolutional layers are summarized in Table B.1. An encoder is used to create a compressed representation of the input occupancy grid ($\mathbf{g}_{in}$). The encoder consists of three convolutional layers and one fully-connected layer. Using the same layers, a decoder reconstructs the occupancy grid ($\mathbf{g}_{out}$). During training, we aim to minimize the loss ($\mathcal{L}_{\text{reconstruction}}(\mathbf{g}_{in}, \mathbf{g}_{out})$ between the input and output grid.

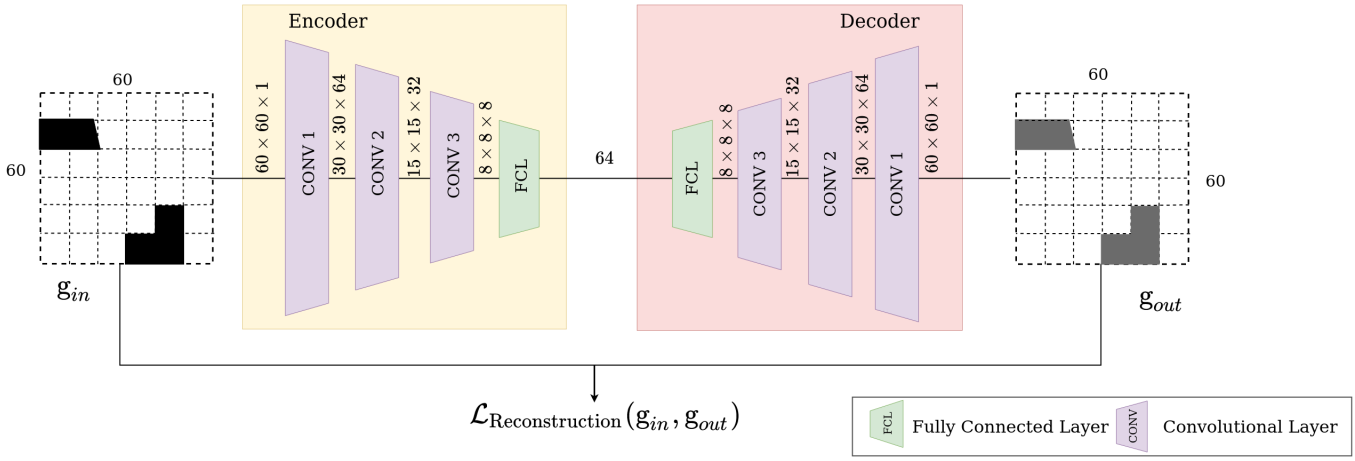


Figure B.1: Convolutional Neural Network (CNN) architecture. An input occupancy grid ($\mathbf{g}_{in}$) is processed by an encoder, consisting of three Convolutional layers (Conv) and one Fully Connected Layer (FCL), to create a compressed representation of the grid. Next, the compressed representation is reconstructed to an output grid ($\mathbf{g}_{out}$) by a decoder, which uses the same layers as the encoder. During training, the CNN tries to get $\mathbf{g}_{out}$ as similar to $\mathbf{g}_{in}$.

Table B.1: Properties of the Convolutional layers in the CNN as shown in Figure B.1.

Layer	Num. filters	kernel	Stride	Activation
Conv1	64	(5,5)	2	ReLU
Conv2	32	(3,3)	2	ReLU
Conv3	8	(3,3)	2	ReLU

B.2. CNN training

Here, we analyze the training process of the CNN during training of the complete network with static obstacles. Note, the training used for this analysis does not include a regularization loss.

The evolution of the auto-encoder loss throughout the entire training process is depicted in Figure B.2. We see that every time a new environment is introduced, the loss of the auto-encoder increases and slowly decreases during training in the particular environment, indicating that the CNN is learning.

Figure B.3 shows the difference between the input and output occupancy grid towards the end of the training process. The spatial context between the input occupancy grid and the output grid is maintained, meaning that the encoder can correctly generate a compressed representation of the input grid. Therefore, it is useful to use the compressed representation as input to the neural network, significantly reducing the training time.

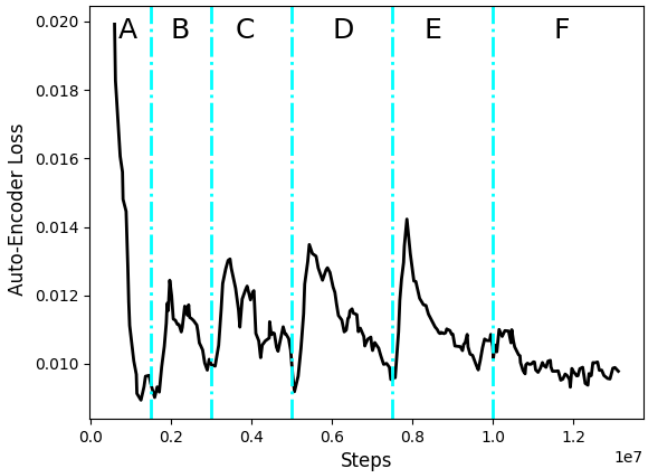


Figure B.2: Auto-encoder loss during training of network with static obstacles. The vertical lines indicate a change in the environment. (A) 1 random obstacle in the environment, (B) 1-3 random obstacles, (C) 2-5 random obstacles, (D) corridor scenario, (E) crossing scenario, (F) door scenario.

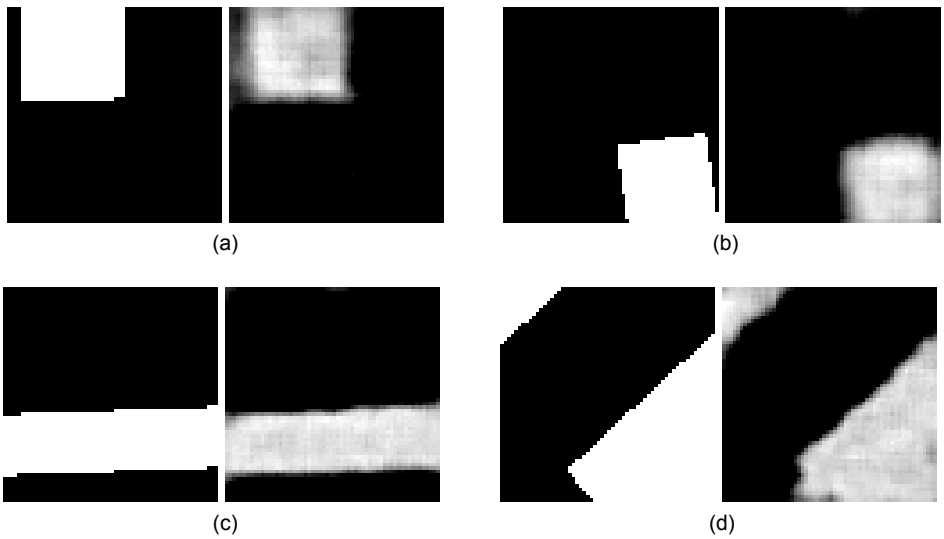


Figure B.3: Four examples of input and output occupancy grid maps created by the CNN. The left image is the input occupancy grid and the right image is the reconstructed occupancy grid.

C

Additional simulation experiments

This appendix describes extra experiments done in simulation. In the first section, additional experiments with static obstacles are provided. In the second section, additional experiments on continual learning are provided.

C.1. Experiments in the door scenario

In this paper, the network training for static obstacle avoidance was done on four different environments: random, corridor, crossing and door. The navigation performance in the door scenario was worse than in the other scenarios. The door is challenging since all agents have to pass through a narrow passage, creating congestion. Here, we investigate if the bad navigation performance in the door scenario is due to the scenario itself or the initialized network.

We train a model only on the door scenario for a total of four million steps. We initialize the network with an SL phase for 3^5 steps. Quantitative results on 100 episodes are shown in Table C.1. The amount of deadlocks is high, even higher than when first trained on all other three environments (as done in our paper). Therefore, we can say that the bad navigation performance in the door scenario is not due to the initialized network with three other environments but that the door scenario is challenging to train. The qualitative results (Fig. C.1) show that deadlocks are caused by the subgoal being on the other side of the wall as the agent. From this situation, the agent is not able to recover.

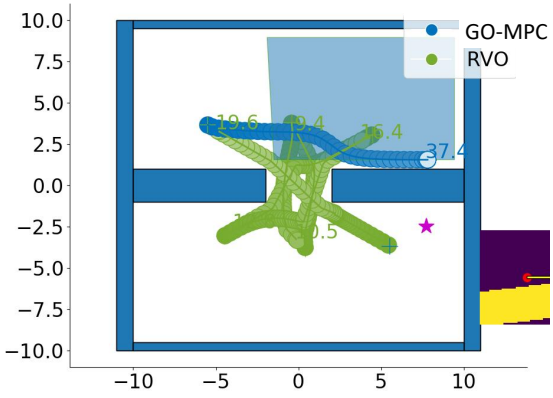


Figure C.1: Deadlock in the door scenario. The subgoal (pink star) is located on the other side of the wall as our agent (blue circle). Other agents in the environment (green) have the RVO policy.

Time (mean ± std)[s]	% Failures (% collisions / % deadlocks)	Distance (mean ± std)[m]
29,43 ± 6,12	34(0/34)	11.29 ± 3.32

Table C.1: Average performance results of 100 episodes on the door scenario. Time and distance to goal are for successful episodes.

C.2. Continual learning experiments

This section presents additional results to the continual learning experiments done in the paper. Moreover, it also presents additional experiments with our proposed continual reinforcement learning framework on a different sequence of tasks.

C.2.1. Additional results on learning an increasing number of agents

In our paper, we present experiments on learning a sequence of three tasks consisting of 2, 4 and 6 agents, respectively. The paper only shows the average results of the learned models on the three tasks. Here, we present more extensive results of the performance on each task separately. Table C.2 shows quantitative results across 100 episodes on each task and their average in terms of time to goal, % failures, and traveled distance to the goal.

Moreover, we evaluate the performance on all three tasks while learning. Figure C.2 shows the average cumulative rewards across 100 episodes on all three tasks for three models: (1) trained only on task 1, (2) trained sequentially on task 1 and 2, (3) trained on all tasks. The cumulative average rewards of all four approaches decrease with an increasing number of agents in the environment. This decrease is caused by the increasing difficulty of the task, resulting in more failures and hence lower rewards. Moreover, it can be seen that the vanilla approach cannot retain knowledge on task 1 and task 2. Lastly, a drawback of regularization-based approaches is that the stability updates towards the previously learned tasks can restrict the model from learning new tasks, i.e., the model saturates. However, we observe that the performance of EWC and online EWC on task 3 is not inferior to the two baselines. Therefore, we can conclude that for these three tasks, model saturation does not occur.

Table C.2: Quantitative results of the four learned models (vanilla, joint, EWC and online EWC) on each learned task (2-agent, 4-agent and 6-agent swapping scenario). Results are averaged across 100 episodes. Time and distance to the goal are computed for successful episodes. % failures are the percentage of episodes ending in a deadlock or collision.

	Time (mean ± std)[s]				% Failures (% collisions / % deadlocks)				Distance (mean ± std)[m]			
Agents	2	4	6	Avg	2	4	6	Avg	2	4	6	Avg
RVO agents												
Vanilla	19.74±5.80	19.50±5.76	19.48±5.69	19.57±5.75	13(0/13)	17(0/17)	15(0/15)	15(0/15)	17.54±7.16	17.38±7.15	17.26±6.69	17.39±7.00
Joint	16.49±4.79	15.99±4.65	15.96±4.67	16.14±4.70	0(0/0)	0 (0 / 0)	0 (0 / 0)	0 (0 / 0)	15.54±4.08	15.08±4.01	14.94±4.07	15.19±4.05
EWC	14.17±4.09	13.58±3.92	13.71±3.97	13.82±3.99	0 (0 / 0)	0 (0 / 0)	0 (0 / 0)	0 (0 / 0)	13.37±3.14	12.74±2.95	12.64±2.95	12.92±3.01
Online EWC	14.10±4.07	13.47±3.89	13.63±3.93	13.73±3.97	0 (0 / 0)	0 (0 / 0)	0 (0 / 0)	0 (0 / 0)	13.28±3.05	12.63±2.85	12.58±2.84	12.83±2.91
Mixed agents												
Vanilla	19.60±5.77	19.00±5.66	18.97±5.60	19.19 ± 5.68	14(1/13)	23(6/17)	15(9/16)	20(5/15)	17.41±7.35	16.92±7.80	16.80±7.18	17.04 ± 7.45
Joint	16.50±4.79	15.78±4.62	15.34±4.49	15.87 ± 4.63	0(0/0)	5(5/0)	10(10/0)	5(5/0)	15.55±4.07	14.87±4.37	14.33±4.93	14.92 ± 4.45
EWC	13.53±3.75	13.02±3.92	13.07±3.73	13.20 ± 3.8	0(0/0)	9(9/0)	15(15/0)	8(8/0)	12.63±3.85	12.15±3.14	12.12±3.84	12.30 ± 3.61
Online EWC	13.39±3.83	13.01±3.71	13.02±3.86	13.14 ± 3.80	0(0/0)	9(9/0)	15(15/0)	8(8/0)	12.55±3.79	12.09±3.47	12.23±3.27	12.29 ± 3.51

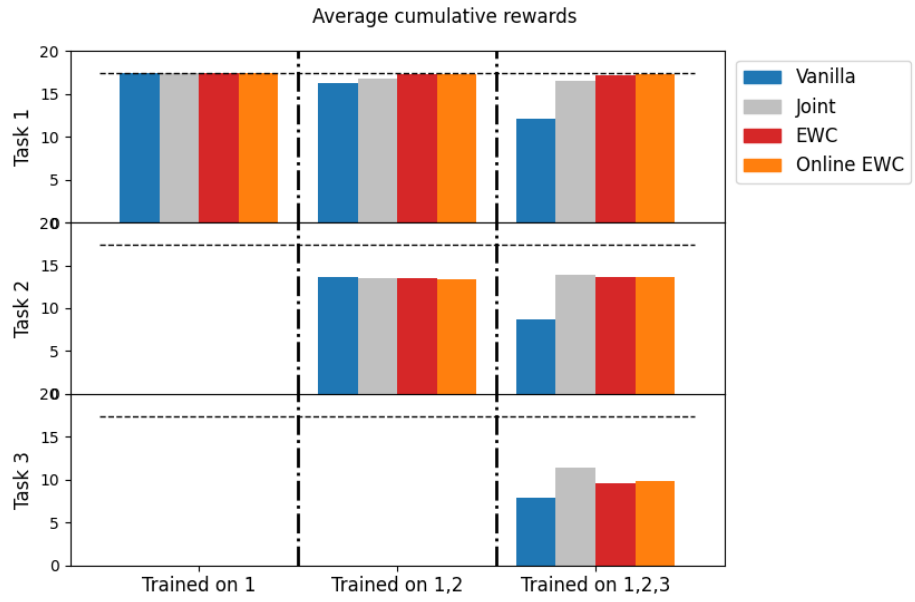


Figure C.2: Average rewards on task 1, 2 and 3 with models trained on task 1, 2 and 3. The dashed black lines indicate the highest rewards obtained. These are test results with only cooperative agents.

C.2.2. Additional experiments on a sequence of task with static obstacles

Using the four approaches presented in the paper (Vanilla and joint baseline, EWC and online EWC), we evaluated continual learning on multiple environments with static obstacles. For training, we consider the following sequence of tasks:

- **Task 1:** One random obstacle in the environment.
- **Task 2:** A corridor scenario.
- **Task 3:** A door scenario.

The changes in the task happen at predefined points equidistant in time. The tasks are learned sequentially, that is $\mathrm{task}\ 1 \rightarrow \mathrm{task}\ 2 \rightarrow \mathrm{task}\ 3$. To accelerate policy convergence, we initialize the training with a supervised learning phase on the first task. We train in the SL phase for 3^5 steps; after that, each task is trained for 3^6 steps. The other agents navigate to their generated goal by using the RVO policy.

Table C.3 shows the average navigation performance of 100 episodes in the three scenarios. The results show that our approaches (EWC and online EWC) outperform the baselines in terms of time and distance to the goal and in % failures. The number of deadlocks of online EWC is surprisingly low since the door scenario normally leads to a high number of deadlocks (see Tab. C.4). This striking result can be caused by using a different way of learning. Moreover, a remarkable result is that the performance of EWC and online EWC is not as similar as shown in our paper. Online EWC performs better in terms of navigation performance. Therefore, we can conclude that it is better to use online EWC. This is not only due to the better performance, but also due to less memory costs compared to EWC.

To get a better insight in the amount of forgetting, we also evaluated all four approaches on task 1 throughout learning the proposed sequence of tasks (Fig. C.3). It can be seen that while training on task 1, the performance on task 1 increases, confirming that our model is learning. When training on tasks 2 and 3, the performance of EWC and the Vanilla baseline drop, indicating catastrophic forgetting. The drop in performance for EWC is not expected. The reason for this drop could be the value λ. From literature we have seen that for different sequences of tasks, a different value of λ is used [20]. Therefore, the value of λ performing well for an increasing number of agents (as in the paper) could not be the correct value for the sequence of tasks we use here. Looking at the results, we assume that a higher value for λ is necessary to retain performance. To confirm this assumption, further experiments

Table C.3: Comparison of the four different approaches in a setting with static obstacles. Results are averaged over 100 episodes on three tasks (random obstacle, corridor, and door). Time and distance to goal are computed for successful episodes. %Failures are the percentage of episodes resulting in either a collision or a deadlock. Note that the high number of deadlocks is caused by the door scenario.

	Time (mean $\pm$ std)[s]	% Failures (% collisions / % deadlocks)	Distance (mean $\pm$ std)[m]
Vanilla	14.38 $\pm$ 4.15	21(5/16)	10.18 $\pm$ 2.97
Joint	15.35 $\pm$ 4.44	23(3/20)	10.83 $\pm$ 3.31
EWC	13.61$\pm$3.95	16(5/11)	9.63 $\pm$ 2.93
Online EWC	12.15$\pm$3.58	9(3/6)	9.99 $\pm$ 2.66

Table C.4: Individual task results corresponding to Table C.3.

	Time (mean$\pm$std)[s]			% Failures (% collisions / % deadlocks)			Distance (mean$\pm$std)[m]		
	Random	Corridor	Door	Random	Corridor	Door	Random	Corridor	Door
Vanilla	9.48$\pm$2.73	13.37$\pm$3.82	20.28$\pm$3.46	6(1/5)	8(3/5)	49(11/38)	8.14$\pm$2.49	11.52$\pm$2.96	10.88$\pm$3.46
Joint	10.04$\pm$2.94	12.08$\pm$3.78	23.92$\pm$6.91	5(0/5)	1(1/0)	63(7/56)	8.98$\pm$3.24	11.75$\pm$2.45	11.75$\pm$4.24
EWC	8.72$\pm$2.53	14.03$\pm$3.97	18.06$\pm$5.35	4(2/2)	11(3/8)	33(9/24)	7.98$\pm$2.23	11.29$\pm$3.02	9.63$\pm$3.54
Online EWC	9.08$\pm$2.59	13.11$\pm$3.76	14.55$\pm$4.37	4(0/4)	7(4/3)	16(5/11)	7.91$\pm$2.07	11.22$\pm$2.79	10.86$\pm$3.12

with different values of λ are necessary on this specific sequence of tasks. However, Fig. C.3 shows that online EWC is able to retain performance on task 1. This also explains the difference between EWC and online EWC observed in Table C.3. Opposed to EWC, the value λ is sufficient for online EWC. The latter indicates that a different value of λ might be necessary when using either EWC or online EWC.

To conclude, this experiment has shown that the performance of our four approaches is dependent on the sequence of tasks. The amount of forgetting on tasks with an increasing number of agents is higher than on the tasks with different static obstacles environments. However, we have shown that for these two sequences of tasks, EWC and online EWC are able to obtain higher performance than the baselines by selectively slowing down the learning on certain parameters.

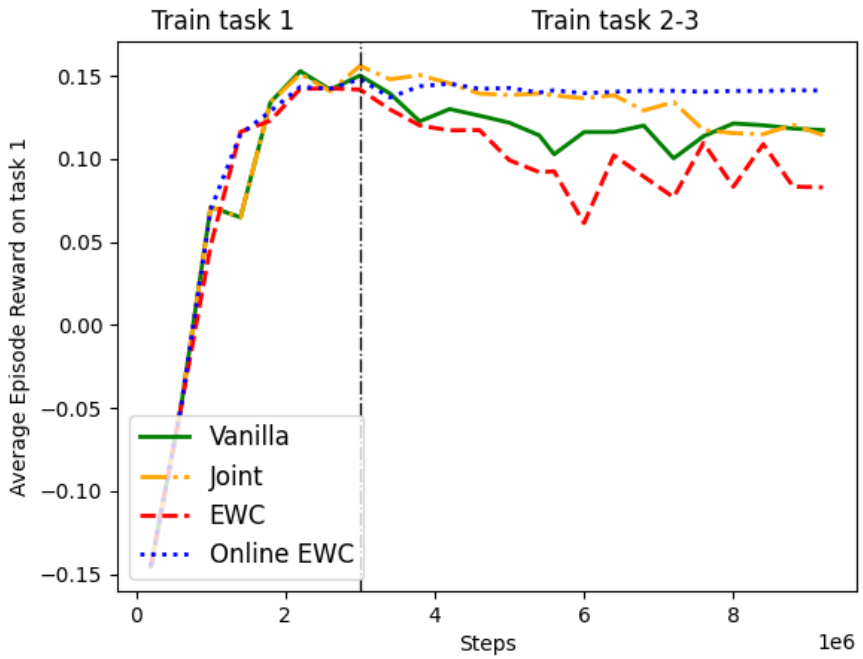


Figure C.3: Average episode reward on task 1 (one obstacle in the middle) across 50 episodes for models trained with vanilla (green), joint (yellow), EWC (red), and Online EWC (blue). The other agents have RVO policy.

D

Real-world experiments

This Appendix provides additional information on the real-world experiments. First, Section D.1 explains the mobile robot in more detail. Section D.2 describes setup for the real-world experiments in more detail. Then, Section D.3 describes a few additional experiments done with the robot in real-world environments.

D.1. Robot description

The mobile robot used for all experiments is the Jackal UAGV (unmanned autonomous ground vehicle), equipped with five intel stereo cameras, an ouster 64-layer 3D LiDAR and built-in imu and GPS (see Figure D.1). The LiDAR measurements reach a distance of 20 m with an angle range from -180 to 180. The angle resolution is 0.01. The sensor update frequency is 20 Hz. The five 2D cameras are mounted in such a way that 360 vision is provided to the robot.

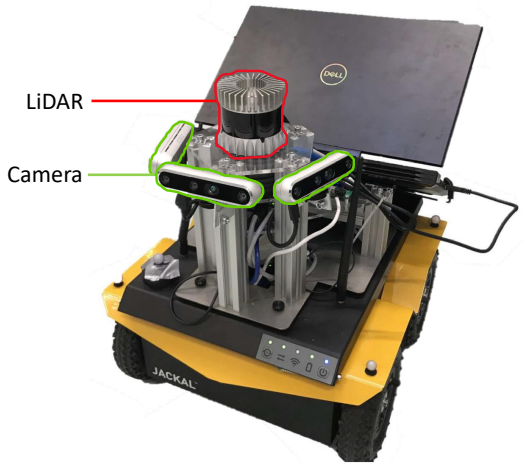


Figure D.1: Jackal Unmanned Autonomous Ground Vehicle (UAGV) equipped with five RGBD camera's and one 3D LiDAR.

D.2. Real-world setup

This section provides a brief description of the implementation of our trained policy on the real robot.

We use the Robotic Operating System (ROS) for integrating of our policy in the real robot. ROS is a flexible framework that provides several tools and libraries to easily create a setup for real-robot applications [26]. Figure D.2 shows an overview of the ROS framework used for our trained policy to work in real-world applications. First, the robot senses the real-world environment with the provided sensors. The sensor data and the pre-defined global map of the environment are provided to the ROS Topics. Then, to obtain all information necessary for the subgoal planner, the sensor data is

processed by multiple tools. For instance, a detection and tracking pipeline detects pedestrians and obtains their positions and velocities. Also, to detect static obstacles, the map is pre-processed by OpenCV, which provides a tool for detecting polygons (obstacles) and providing the polygon's contours in world coordinates. The pre-processed output is fed to the DRL agent, which recommends a subgoal. The MPC-planner generates the necessary actions to reach the subgoal and provides this to the ROS topics. The ROS topics send the necessary information to the robot, which takes the generated actions. The ROS topics run at a speed of 20 Hz. All high-level computations, like obstacle detection, network inference and MPC calculations, are performed on a Dell G7 laptop with Intel Core i7-8th gen and an Nvidia GTX 2070.

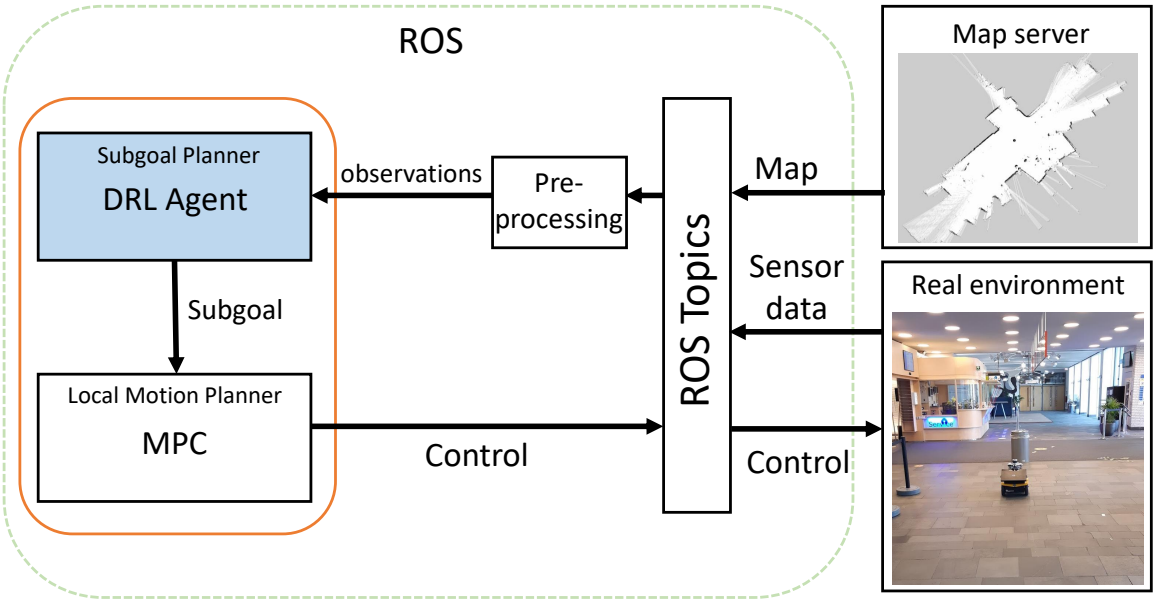


Figure D.2: Architecture used for real-world experiments. The map and sensor data is provided separately to the ROS Topics.

D.3. Additional real-world experiments

To demonstrate our approach in the real world, we started our experiments using the policy trained with the occupancy grid representation on four different environments (as presented in our paper). In the initial experiments, we tested the robot in two unstructured real-world scenarios similar to training scenarios. Note that at the time of both experiments, the pedestrian tracking pipeline on the real robot was not robust enough to provide accurate results, meaning that we could only account for static obstacles. The first scenario is the university hallway with one random obstacle in the middle (Fig. D.3a). A goal position is defined on the other side of the obstacle. The experiment ran four times, and each time the robot successfully reaches the goal. Snapshots of the trajectory in one run is visualized in Fig. D.4. The robot's behavior is conservative and avoids the obstacle in the middle with a large distance between the obstacle and the robot.

The second scenario is a corridor, with and without obstacles (Fig. D.3b). Both experiments are done four times. Experimental results of one run in the empty corridor is shown in Fig. D.5. The top row shows snapshots of camera images during the experiment, and the bottom row shows snapshots of the robot's trajectory, including (sub)goal positions and the MPC plan. The robot does not navigate straight to the goal despite no static or dynamic obstacles in the corridor. The subgoal planner recommends a subgoal outside the corridor. We attribute this error to the sim-to-gap. Despite the wrong positioning of the subgoal, we observe that the robot can still navigate to its goal safely. The latter is due to the integration of the MPC planner, which shows to respect the collision avoidance and kinodynamic constraints. The histogram in Fig. D.7 shows that the distance between the robot and all surrounding obstacles is not below 0.8 m for the entire trajectory, meaning that the robot can safely navigate to

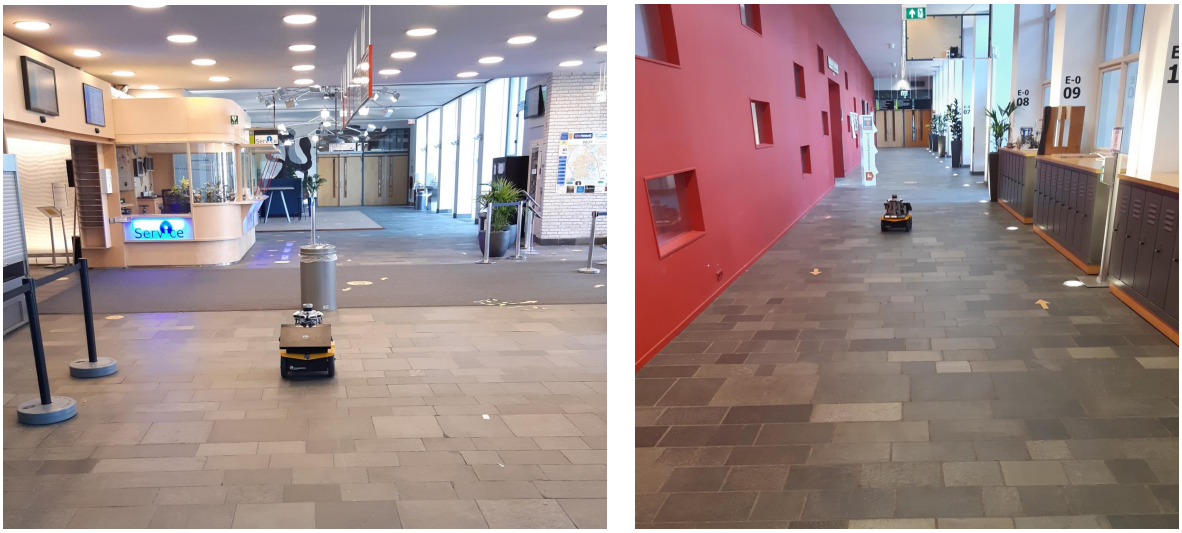
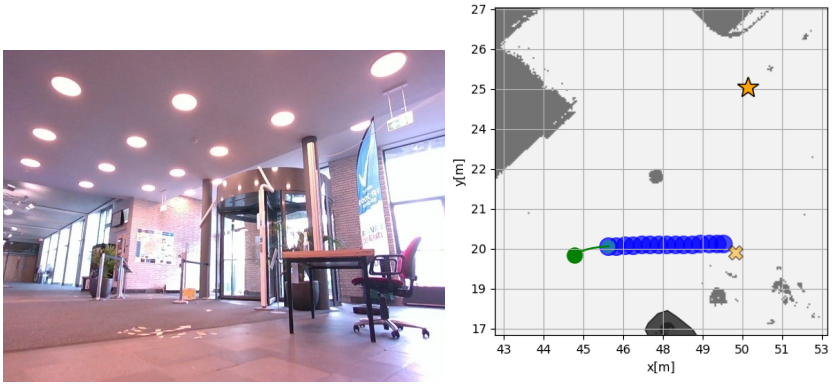
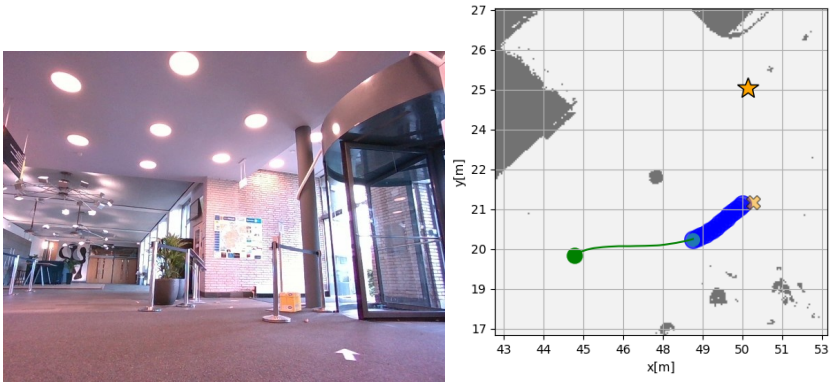


Figure D.3: Real-world scenario used for experiments with the jackal robot. (a) Large hallway at TU Delft with one obstacle placed in the middle. (b) corridor at TU Delft.

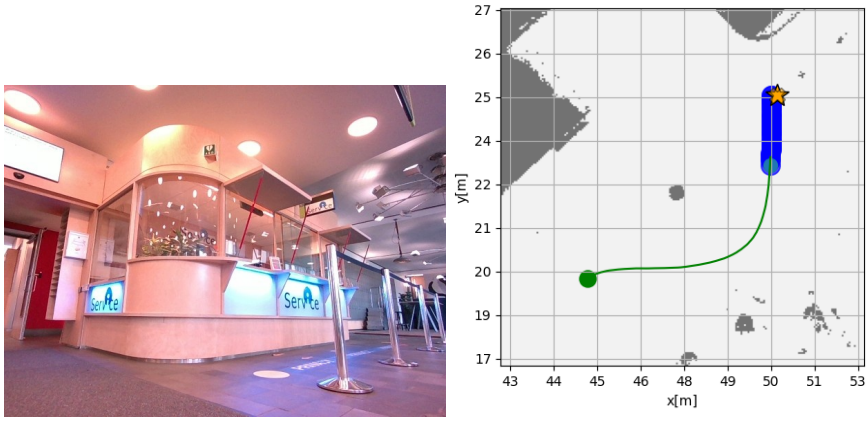
its goal. Experimental results in the corridor with two obstacles are presented in Fig. D.6. Similar to the previous corridor experiment, the subgoal is again provided outside the corridor. Again, the MPC planner succeeds to navigate the robot to the final goal.



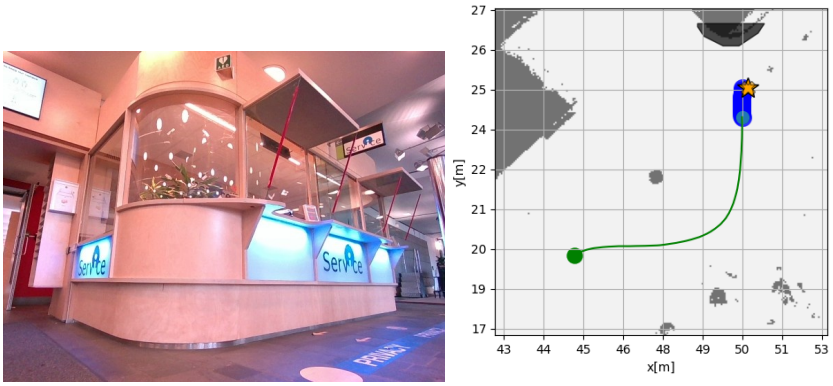
(a) t = 2 s.



(b) t = 6 s.

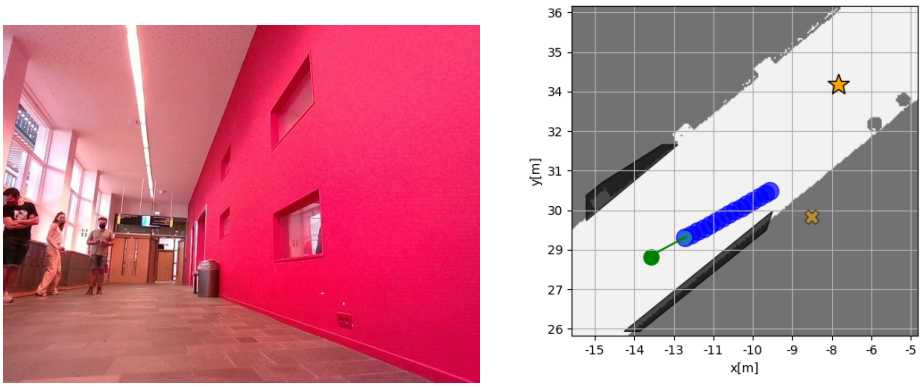


(c) t = 12 s.

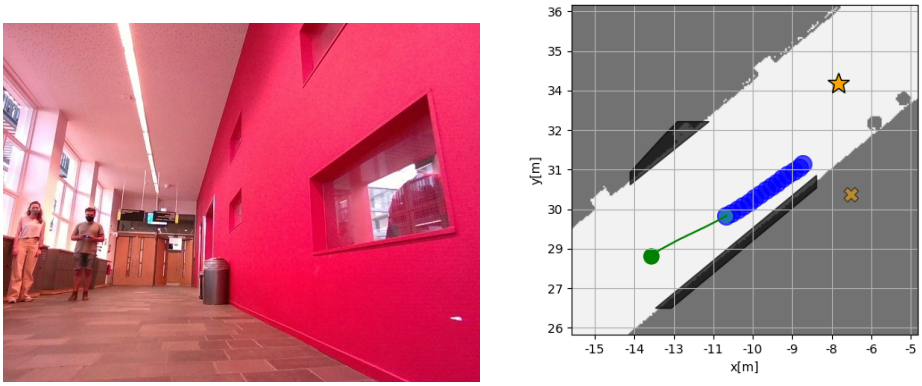


(d) t = 15 s.

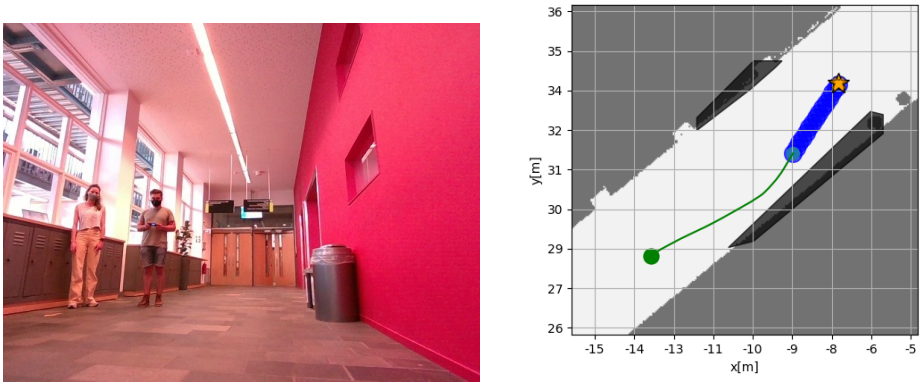
Figure D.4: Collision avoidance with Jackal robot in the university hallway. The robot is required to reach an user-defined goal position. Top row: Camera snapshots of the experiment. Bottom row: Driven trajectory of the robot (green). The MPC plan is marked by the blue circles. Robot initial, subgoal and goal positions are marked in circle disks (green), yellow crosses, and solid stars (yellow), respectively. Black polygons are static obstacles detected by the robot.



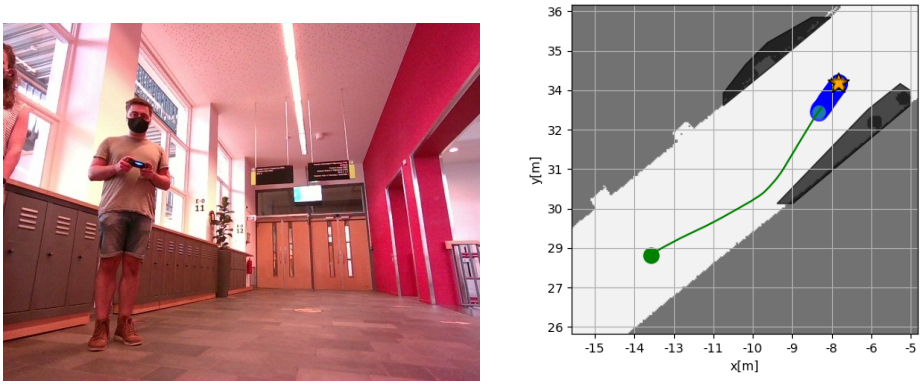
(a) t = 2 s.



(b) t = 4 s.

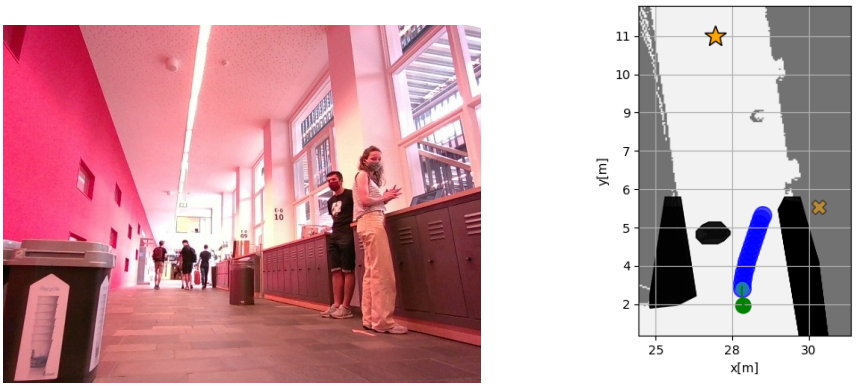


(c) t = 8 s.

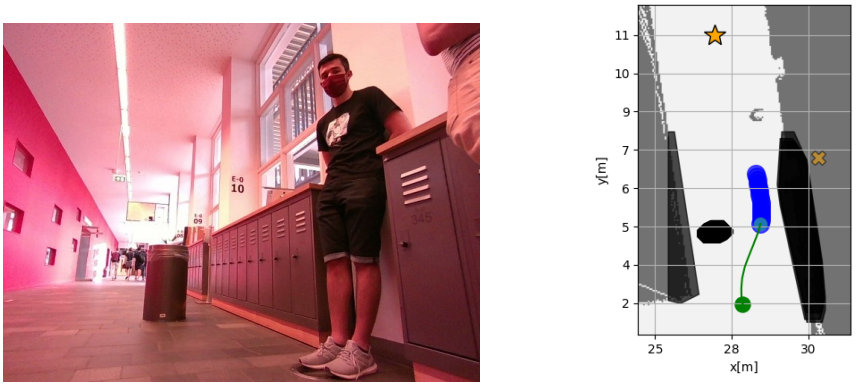


(d) t = 12 s.

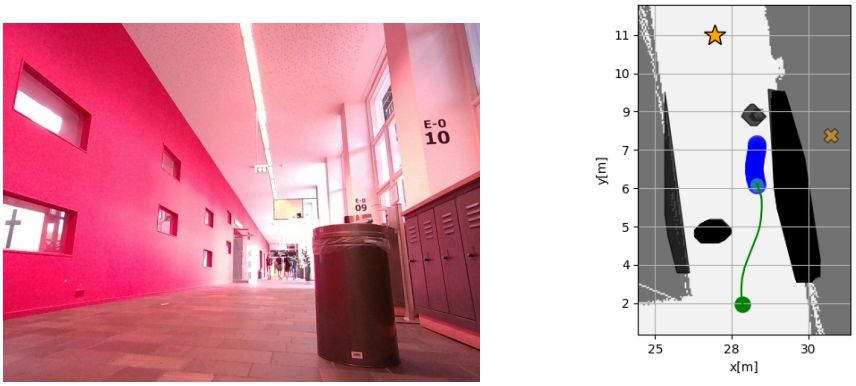
Figure D.5: Collision avoidance with Jackal robot in a corridor scenario. The robot is required to reach an user-defined goal position. Top row: Snapshots of the experiment. Bottom row: Driven trajectory of the robot (green). The MPC plan is marked by the blue circles. Robot initial, subgoal and goal positions are marked in circle disks (green),yellow crosses, and solid stars (yellow), respectively. Black polygons are static obstacles detected by the robot.



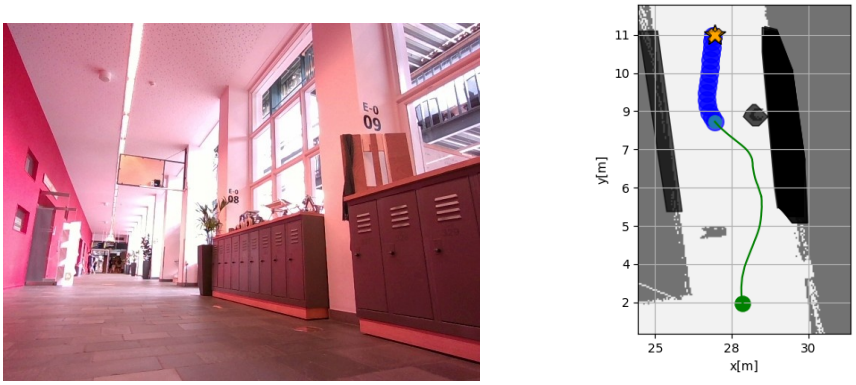
(a) t = 2 s.



(b) t = 6 s.

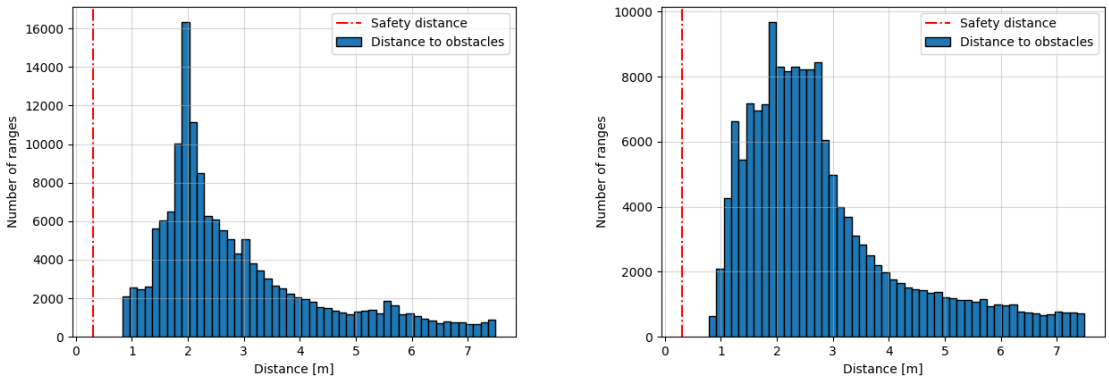


(c) t = 12 s.



(d)t = 15 s.

Figure D.6: Collision avoidance with Jackal robot in a corridor scenario with two obstacles. The robot is required to reach an user-defined goal position. Top row: Snapshots of the experiment. Bottom row: Driven trajectory of the robot (green). The MPC plan is marked by the blue circles. Robot initial, subgoal and goal positions are marked in circle disks (green),yellow crosses, and solid stars (yellow), respectively. Black polygons are static obstacles detected by the robot.



(a) Empty corridor

(b) Corridor with obstacles

Figure D.7: Histogram of distance between Jackal robot and obstacles in (a) an empty corridor and (b) a corridor with two obstacles. Distance is measured by LiDAR (628 ranges each timestep).

Bibliography

[1] Rahaf Aljundi, Klaas Kelchtermans, and Tinne Tuytelaars. “Task-Free Continual Learning”. In: (Dec. 2018). URL: http://arxiv.org/abs/1812.03596.

[2] Jur van den Berg et al. “Reciprocal n-Body Collision Avoidance”. In: vol. 70. Apr. 2011, pp. 3–19. ISBN: 978-3-642-19456-6. DOI: 10.1007/978-3-642-19457-3_1.

[3] Bruno Brito et al. “Where to go next: Learning a Subgoal Recommendation Policy for Navigation in Dynamic Environments”. In: (2021).

[4] Changan Chen et al. “Crowd-Robot Interaction: Crowd-aware Robot Navigation with Attention-based Deep Reinforcement Learning”. In: (Sept. 2018). URL: http://arxiv.org/abs/1809.08835.

[5] Yu Fan Chen et al. “Socially Aware Motion Planning with Deep Reinforcement Learning”. In: (Mar. 2017). URL: http://arxiv.org/abs/1703.08862.

[6] Matthias Delange et al. “A continual learning survey: Defying forgetting in classification tasks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), pp. 1–1. DOI: 10.1109/TPAMI.2021.3057446.

[7] Sayna Ebrahimi et al. “Uncertainty-guided Continual Learning with Bayesian Neural Networks”. In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=HklUCCVKDB.

[8] Michael Everett, Yu Fan Chen, and Jonathan P. How. “Motion Planning Among Dynamic, Decision-Making Agents with Deep Reinforcement Learning”. In: (Sept. 2018). URL: https://arxiv.org/pdf/1805.01956.pdf.

[9] Tingxiang Fan et al. “Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios”. In: *The International Journal of Robotics Research* 39 (7 2020), pp. 856–892. DOI: 10.1177/0278364920916531. URL: https://sites.google..

[10] Ian J Goodfellow et al. *Generative Adversarial Nets*. 2014. URL: http://www.github.com/goodfeli/adversarial.

[11] Dirk Helbing and Peter Molnar. “Social Force Model for Pedestrian Dynamics”. In: *Physical Review E* 51 (May 1998). DOI: 10.1103/PhysRevE.51.4282.

[12] Jerry Hirsch. *The Rise of the Warehouse Robot*. https://www.ttnews.com/articles/rise-warehouse-robot. Accessed: 2021-07-02.

[13] iRobot. *History*. https://www.irobot.nl/about-irobot/company-information/history. Accessed: 2021-07-10.

[14] Victoria Kickham. *E-commerce growth sparks demand for warehouse automation*. https://www.dcvelocity.com/articles/47172-e-commerce-growth-sparks-demand-for-warehouse-automation. Accessed: 2021-07-10. Sept. 2020.

[15] James Kirkpatrick et al. “Overcoming catastrophic forgetting in neural networks”. In: (Dec. 2016). URL: http://arxiv.org/abs/1612.00796.

[16] Henrik Kretzschmar et al. “Socially compliant mobile robot navigation via inverse reinforcement learning”. In: *The International Journal of Robotics Research* 35 (11 Sept. 2016), pp. 1289–1307. ISSN: 0278-3649. DOI: 10.1177/0278364915619772. URL: http://journals.sagepub.com/doi/10.1177/0278364915619772.

[17] Thibault Kruse et al. “Human-aware robot navigation: A survey”. In: *Robotics and Autonomous Systems* 61.12 (2013), pp. 1726–1743. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2013.05.007. URL: https://www.sciencedirect.com/science/article/pii/S0921889013001048.

[18] Steven M. LaValle. *Planning algorithms*. Cambridge University Press, 2006. ISBN: 0521862051. DOI: https://doi.org/10.1017/CBO9780511546877. URL: http://planning.cs.uiuc.edu/.

[19] Timothée Lesort et al. “Continual Learning for Robotics: Definition, Framework, Learning Strategies, Opportunities and Challenges”. In: (June 2019). URL: http://arxiv.org/abs/1907.00182.

[20] Vincenzo Lomonaco et al. “Continual Reinforcement Learning in 3D Non-stationary Environments”. In: (2020). arXiv: 1905.10112 [cs.LG].

[21] Bob Malone. *George Devol: A Life Devoted to Invention, and Robots*. Ed. by IEEE Spectrum. https://spectrum.ieee.org/automaton/robotics/industrial-robots/george-devol-a-life-devoted-to-invention-and-robots?utm_source=robots.ieee.org. Accessed: 2021-07-14. Sept. 2011.

[22] Cosmin Paduraru et al. “Challenges of Real-World Reinforcement Learning:Definitions, Benchmarks Analysis”. In: *Machine Learning Journal* (2021).

[23] Joan Serrà et al. *Overcoming catastrophic forgetting with hard attention to the task*. 2018. arXiv: 1801.01423 [cs.LG].

[24] Shai Shalev-Shwartz. “Online learning and online convex optimization”. In: *Foundations and Trends in Machine Learning* 4 (2 2011), pp. 107–194. ISSN: 19358237. DOI: 10.1561/2200000018.

[25] Daniel Silver, Qiang Yang, and Lianghao Li. “Lifelong Machine Learning Systems: Beyond Learning Algorithms”. In: Mar. 2013.

[26] Stanford Artificial Intelligence Laboratory et al. *Robotic Operating System*. Version ROS Melodic Morenia. May 23, 2018. URL: https://www.ros.org.

[27] Pete Trautman and Andreas Krause. “Unfreezing the Robot: Navigation in Dense, Interacting Crowds”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems* (Sept. 2010), pp. 797–803. DOI: 10.1109/IROS.2010.5654369. URL: https://www.researchgate.net/publication/221064452.

[28] Friedemann Zenke, Ben Poole, and Surya Ganguli. “Continual Learning Through Synaptic Intelligence”. In: *Proceedings of the 34th International Conference on Machine Learning*. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, June 2017, pp. 3987–3995. URL: http://proceedings.mlr.press/v70/zenke17a.html.