Article

Robot Navigation in Crowded Environments: a Reinforcement Learning Approach

Matteo Caruso [1,*,†], Enrico Regolin [2,†], Federico Julian Camerota Verdù [2], Stefano Alberto Russo [2], Luca Bortolussi [2] and Stefano Seriani [1]

[1] Department of Engineering and Architecture, University of Trieste, via A. Valerio 6/1, 34127 Trieste, Italy; matteo.caruso@phd.units.it (M. C.), sseriani@units.it (S. S.)
[2] Department of Matematics and Geoscience, University of Trieste, Via Edoardo Weiss 2, 34128 Trieste, Italy; enrico.regolin@gmail.com (E. R.), federicojulian.camerotaverdu@phd.units.it (F. J. C. V.), stefanoalberto.russo@phd.units.it (S. A. R.), lbortolussi@units.it (L. B.)
* Correspondence: matteo.caruso@phd.units.it (M. C.)
† These authors contributed equally to this work.

Abstract: For a mobile robot, navigation in a densely crowded space can be a challenging and sometimes impossible task, especially with traditional techniques. In this paper, we present a framework to train neural controllers for differential drive mobile robots which must safely navigate a crowded environment while trying to reach a target location. To learn the robot's policy, we train a convolutional neural network using two reinforcement learning algorithms, *Deep Q-Networks* (DQN) and *Asynchronous Advantage Actor Critic* (A3C), and develop a training pipeline that allows to scale the process to several compute nodes. We show that the asynchronous training procedure in A3C can be leveraged to quickly train neural controllers and test them on a real robot in a crowded environment.

Keywords: mobile robotics; neural networks; control systems; reinforcement learning; crowd navigation

1. Introduction

Since the early days of mobile robotics, there has been a lot of interest in robot autonomy and the possibility of using robots in scenarios that involve interaction and collaboration with human beings. A crucial requirement to enable such applications is that people must feel safe and comfortable with an autonomous robot moving and performing tasks around them. This is particularly true in contexts such as Smart Factories. The use of mobile robots, often referred to as *Autonomous Mobile Robots* (AMRs), within this scenario is relatively new and it is spreading in the industry [1,2], usually as "fleets" that are controlled by a fleet manager [3,4]. In Smart Factories, mobile robots may perform their tasks on their own or in a collaborative manner either with other machines, robotic systems or human operators. In the latter case, safety should be the main concern, since the robot must not harm humans in its vicinity. On the other hand, the robot needs to be capable of inferring human intentions and properly react when operating among humans [5]. A common issue arising in highly populated environments is the so-called *freezing robot problem* [6], i.e. the robot getting stuck when surrounded by sufficiently dense crowds. Due to the huge potential of fully autonomous systems in industrial and commercial applications, the crowd navigation problem for mobile robots has been investigated by many authors in past years. Trautman et al. in 2013 used interactive Gaussian Processes to achieve improved cooperation between robots and humans in dense crowd navigation [7] settings. Instead, Abdulov and Abramenkov employs a communication scheme to reduce the collisions within a fleet of autonomous robots. This work is the natural continuation of our previous work [9], where the controller for differential drive mobile robot, represented by an artificial neural network (ANN), was trained by means of the NEAT algorithm [10] using an evolutionary strategy.

 doi:10.20944/preprints202212.0233.v1

The aim of this work is to exploit computer simulations that are cheap and fast in order to train a controller for a mobile robot that will operate in the real world. Such a setting is often referred to in the literature as *Sim2Real* [11] and has gained a lot of interest in recent years. A crucial aspect of sim2real is that there must be a good degree of alignment between the simulated environment and the real scenario, in order to make the learned policy transferable to the real world. In our case, in order to obtain a reliable environment it's important to carefully choose the method used to model the crowd behaviour.

Crowd simulation [12] is an established approach to simulate crowds in several applications like video games.

There are different approaches that can be used to perform crowd modelling and simulation: force-based interactions, pedestrians flow, rule-based, psychology-sociology inspired and others. Comprehensive surveys and discussions about crowd modelling and simulation techniques are available in literature [13,14]. In our work, we rely on a *social force model* based on the work of Helbing *et al.* shown in [15,16] that model the behaviour of crowds reacting to panic situations using a model based on forces that are inspired by self-driven many-particle systems; furthermore, a similar approach has been used in other crowd models [17]. In particular, they consider so-called *interaction forces* to model the pedestrian velocity changes. Such forces are used to represent empirical observations about walking humans, like the fact that pedestrians tend to keep a velocity-dependent distance between each other and with walls. The authors also introduce a *repulsive interaction force* to encode the psychological observation that pedestrians tend to stay away from each other. Finally, *body forces* and *sliding friction forces* are included to account for granular interactions that occur, especially in panicking crowds. The same authors have also developed the software implementation of this crowd model, and they called it PySocialForce and released as a python package. Our work implements their software, which has been further extended within this work.

With the increasing availability of resources and computing power, Reinforcement Learning techniques have been successfully used to solve control and optimisation problems in several domains ranging from video games [18], chip placement [19] and control of stratospheric air balloons [20] and nuclear reactors [21]. The use of deep neural networks that can easily handle high dimensional inputs, has been a key enabler in the recent success of reinforcement learning. In robotic applications, the input of the control system is usually composed by the readings of many sensors and actuators mounted on the robot. Hence, directly learning a policy to solve the desired task can be extremely difficult [22]. Yet, deep RL made mapless robot navigation possible by leveraging perceptual information and neural networks [23]. To tackle navigation in crowded environments, previous work tries to leverage RL and simulation models to train the robot's controller [24] [25]. Furthermore, Regier *et al.* used the SFM for crowd motion and a NN, based on the chunk concept for its input layer, to control a mobile robot [26].

Other examples of RL applied to PySocialForces exist, e.g. the work from Katyal *et al.* [24]. However, our approach differs in several ways. We extended the social forces model to include more complex social behaviours (agents can stop, split, group, change direction, etc.). The paper from Katyal *et al.* uses Proximal Policy Optimization (PPO) first presented by Schulman *et al.* [27], while we use two algorithms called Deep Q-Networks (DQL) Mnih *et al.*[28] and Asynchronous Advantage Actor Critic (A3C) Mnih *et al.*[29]. The main contributions of our work are:

- Development of an extended social forces model, which allows the introduction of more general social behaviour such as pedestrians stopping in the environment, grouping, splitting, sudden change of direction and so on, together with the introduction of the pedestrian-to-robot repulsive force;
- Development of a functional and dimensionally efficient CNN-based architecture to tackle the Crowd Navigation problem;
- Rigorous benchmarking of DQL and A3C RL algorithms applied to a crowd navigation problem;

 doi:10.20944/preprints202212.0233.v1

- Detailed presentation of the parallel and asynchronous computational strategies employed to speed up training, with illustration of the full pipelines used for the two variants. In the two applications we consider the cases in which different computational resources are available (GPU or multiple CPUs);
- Development of a Robot Operating System (ROS) [30] package for robot control using the trained NN, mapping, visualization, localization, position estimation, trajectories definition. The user can easily provide target and way-points through the handy Rviz GUI. The developed package can be used both in simulated and experimental environments.
- Experimental validation of the trained controller on a commercially available mobile robot, testing in a realistic scenario the strategy trained on the newly proposed extended Social Forces model.

The paper is structured as follows: in Sec. 2 the problem statement is reported, i.e. the modelling of the single components of the simulated environment, which comprise the crowd dynamics model, the mobile robot model, the model of the environment and the robot's perception system; in Sec. 3 we describe the methodology adopted to solve the problem, i.e. the architecture of the RL, the algorithm and the topology of the NN used; in Sec. 4.1 the training results obtained for both the DQL and A3C algorithms are reported, while in Sec. 4.2 the validation process findings are summarized; in Sec. 5 the experimental validation and results are described; finally in Sec. 6 we present the concluding remarks as well as the planned future works along this line of research.

2. Problem Statement

This section includes the description of the problem statement. More specifically: (i) the crowd model that has been selected in order to represent the moving crowd; (ii) the kinematics model of the considered mobile robot, together with the modeling of its perception system; (iii) the descriptio, within the model, of the simulated environment; (iv) the "map chunk" model that has been introduced to help speeding up the training process of the controller.

2.1. Social Forces Model

In this work, in order to simulate the moving crowd and its behaviour we employ an engine based on the *Social-Forces Model*. In particular, we use the python module `PySocialForce` which is an implementation of the *Extended Social Forces* (ESF) model [31], extending [32]. Within this work, the original implementation has been further extended in order to increase generality, taking into consideration general social actions that can be observed in real crowds in day-to-day activities. In particular, we introduce terms to the model that encode the following phenomena: pedestrians and groups stopping in the environment; pedestrians dynamically grouping with other pedestrians or existing groups; single pedestrians leaving their groups and heading in other directions; groups splitting into smaller groups; finally, the possibility for pedestrians and groups to meet with each other.

The Social-Forces model, which has been largely studied in past, is a*Microscopic Approach* that aims at describing and simulating a moving crowd. The model assumes that the motion of the single pedestrian composing the crowd can be described by Newton's second law, i.e. its dynamics through a sum of "forces" contributions. The Extended Social-Forces Model, to which the python implementation refers, has been developed by the same research team. In their multiple works, they have characterized every force component and tuned the model through experimental observations of real crowds. In the following we provide a brief summary of these contributions; however, complete and detailed discussions are available in the original works of the authors [31–34]. The Extended Social-Forces Model assumes that the motion of the single pedestrian i can be described by the Newton's second law as follows,

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2022 doi:10.20944/preprints202212.0233.v1

$$\ddot{\boldsymbol{x}}_{\boldsymbol{i}} = \boldsymbol{f}_{t,i} + \sum_{j=1}^{n} \boldsymbol{f}_{\boldsymbol{i,j}} + \boldsymbol{f}_{\boldsymbol{w,i}} + \boldsymbol{f}_{\boldsymbol{g,i}} \tag{1}$$

where the specific external force $\boldsymbol{f}_{\boldsymbol{t,i}}$ quantifies the motion of pedestrian i to a desired location; $f_{i,j}$ represents the repulsive force contribution coming from the interaction with another pedestrian j; $f_{w,i}$ represents the repulsive force contribution due to an obstacle w present in the environment and finally $f_{g,i}$ is a grouping force. The motion of the pedestrian i, then happens as a consequence of the sum of these forces.

Helbing et al. in [32] proposed that the target acceleration term is described by the following relation:

$$f_{t,i} = \frac{v_i^t \hat{e}_i^{\,t} - \boldsymbol{v}_{\boldsymbol{i}}(t)}{\tau} \tag{2}$$

where v_i^t is the desired speed of the individual i, $\hat{e}_i^{\,t}$ is the versor of the desired direction, $v_i(t)$ is the individual current speed vector and finally τ is a relaxing factor.

Moussaïd et al. in [34] stated that the single repulsive interaction force acting between the pedestrian i and a pedestrian j can be described as,

$$f_{i,j} = -Ae^{-d/B}\left[e^{-(n'B\theta)^2}\boldsymbol{t} + e^{-(nB\theta)^2}\boldsymbol{n}\right] \tag{3}$$

where, d is the distance between the pedestrians; $\boldsymbol{t}$ is the interaction direction, i.e. the versor pointing from i to j; $\boldsymbol{n}$ is the normal versor to $\boldsymbol{t}$ oriented to the left; θ is the angle between $\boldsymbol{t}$ and the versor pointing from i to j; A, B, n, n' are model parameters.

Johansson et al. in [33] stated that the repulsive force contribution to individual i, due to proximity to obstacles and walls can be modeled as a decaying exponential function as follows,

$$f_{w,i} = ae^{-d_w/b} \tag{4}$$

where, d_w is the normal distance between individual i and the obstacle, a and b are model parameters.

Finally, Moussaïd et al. in [31] introduced the grouping contributions – needed to keep groups individuals close to each other – defined to be composed of three terms as follows,

$$f_{g,i} = f_{g,i}^{g} + f_{g,i}^{a} + f_{g,i}^{r} \tag{5}$$

where, $f_{g,i}^{g}$ is the gazing term which is responsible for adjusting the i-th pedestrian position in order to reduce its head rotation, $f_{g,i}^{a}$ is the group attraction term responsible for attracting pedestrians belonging to the same group, and $f_{g,i}^{r}$ is the group repulsive term responsible for avoiding that the group members collide and overlap with each other. A complete definition of the single terms is available in literature [31].

It has been noted that simulated pedestrians, during the process of avoiding static obstacles, exhibit a lane-following behaviour; furthermore, when close to obstacles of complex shape, some of them get stuck in a local minimum. In order to help pedestrians avoid static obstacles more effectively, within this work they have been equipped with an ad-hoc controller. Finally, compared to the original implementation of the ESF, our implementation of the pedestrians' dynamics takes into account that the single pedestrian is aware of the robot. This is done through an appropriate small repulsive force $f_{i,r}$, which activates only in the vicinity of the robot, i.e. when the distance is below the d_R threshold. Eq. (1) is then modified by adding the robot to the pedestrian repulsive force. However, it must be pointed out that we elected to keep this force component deactivated during the training and validation processes, which means that the pedestrians are not aware of the robot moving in the environment. We elected to deactivate this component on purpose, because it represent a very cautionary case, in which the mobile robot has to learn in a more

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2022 doi:10.20944/preprints202212.0233.v1

complex environment. In fact, the robot is considered as a ghost, raising no feedback from the crowd.

2.2. Mobile Robot Kinematics

As a case study, within this paper a differential drive Wheeled Mobile Robot (WMR), characterized by two independent driving wheels sharing a common axis of rotation, has been considered. This configuration allows the robot to drive straight, steer and rotate in place; the robot cannot move laterally due to its kinematic and non-holonomic constraints. To describe its motion, we introduce two reference frames: an *inertial reference frame* $(\boldsymbol{O}, \hat{e}_x, \hat{e}_y)$ and a *local frame* integral with the WMR $(\boldsymbol{C}, \hat{\imath}, \hat{\jmath})$. The rotation ϑ between the two reference frames represents the robot heading. This can be seen in Figure 1.

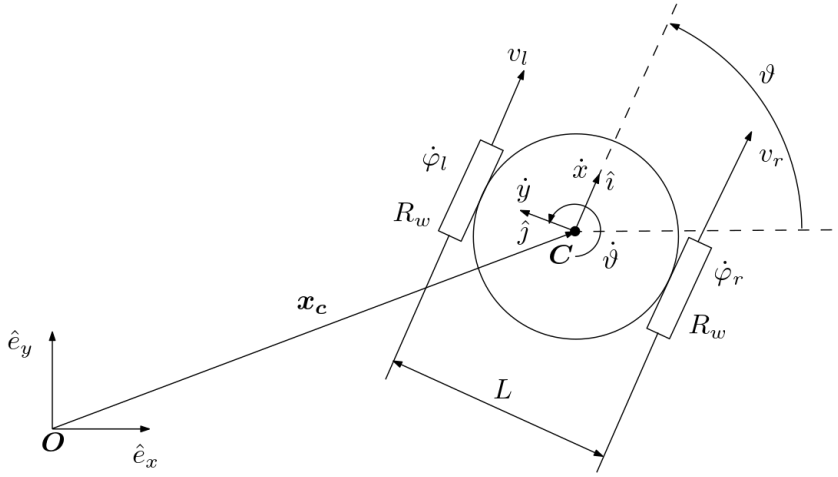


Figure 1. Kinematic model of a differential drive mobile robot, with focus on the relation between the robot's frame and the inertial frame.

Using the notation in Figure 1, and under the assumptions of pure rolling and no lateral slip, the forward differential kinematics model for a differential drive mobile robot, expressed in the robot local reference frame, is described as follows,

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\vartheta} \end{bmatrix} = \begin{bmatrix} \frac{R_w}{2}(\dot{\varphi}_l + \dot{\varphi}_r) \\ 0 \\ \frac{R_w}{L}(\dot{\varphi}_r - \dot{\varphi}_l) \end{bmatrix} \tag{6}$$

where R_w is the wheels radius, $\dot{\varphi}_l$ and $\dot{\varphi}_r$ are the wheels angular speeds, respectively for the left and the right one, and finally L is the distance between the wheels. Moreover, $\dot{x}$, $\dot{y}$ and $\dot{\vartheta}$ are the linear and angular speed of the WMR expressed in its local reference frame, as shown in Figure 1. The same speeds – and thus the kinematic model itself – can be expressed in the inertial frame, by means of a transformation, as follows,

$$\dot{\boldsymbol{x}}_{\boldsymbol{I}} = \boldsymbol{R}^{\boldsymbol{T}} \dot{\boldsymbol{x}} \tag{7}$$

where $\boldsymbol{R}$ is the rotation operator between local and inertial frame, which is shown below,

$$\boldsymbol{R} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

Eq. (7) represents the differential kinematics of the WMR in the inertial frame, and is the staring point for the localization of the mobile robot. Indeed, a system has been implemented which simulated the estimation of the position of the mobile robot through wheels odometry.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2022 doi:10.20944/preprints202212.0233.v1

The modeled WMR is limited to positive values of linear speed, which is a safety measure consistent with the fact that the installed LiDar only scans the portion of space ahead. Moreover, the WMR speed is limited to maximum values in order to reduce the admissible robots speeds and keep it close to a maximum. These limitations are summarized in the following equations,

$$\begin{cases} \dot{x} = v \in [0, v_{max}] \\ \dot{y} = 0 \\ \dot{\vartheta} = \omega \in [-\omega_{max}, \omega_{max}] \end{cases} \tag{9}$$

The parameters characterizing the WMR, the 2D-LiDAR and the odometric system are summarized in Table 1.

From a high level control point of view, the robot receives linear and angular velocity setpoints $[v^*, \omega^*]$ which are tracked exploiting (6). In order to have a finite set of inputs which a NN can choose from, the setpoints are updated at each step, considering the differential input

$$u = [\Delta v, \Delta \omega] \tag{10}$$

so that at time instant k, having expressed (9) with the saturation functions sat_v, sat_ω, one has:

$$v^*(k) = sat_v(v^*(k-1) + \Delta v(k)) \tag{11}$$

$$\omega^*(k) = sat_\omega(\omega^*(k-1) + \Delta \omega(k)) \tag{12}$$

2.3. Environment Description

The environment used for in-simulation training is defined as a bi-dimensional space, having width W and height H with static obstacles and moving pedestrians, where the environment's reference frame coincides with the inertial frame. At the beginning of an episode, the static obstacles are generated with random positions and shapes in order to expose the robot to diverse situations and generalize the problem. The pedestrians instead can enter and exit the environment on the boundaries. On the other hand, no information is given to the WMR about the environment, as it gathers local information via its perception system, which is discussed in detail in the next section.

In Figure 2, we can see an example of training scenario where the robot spawns in a randomly generated point P and has to reach the randomly generated target location T while avoiding collisions with the moving crowd, represented by the points p_i that denote the single pedestrians, while the group of pedestrians is indicated with g_i. Since both the robot's position and the target are randomly generated, in order to ensure that all trajectories are characterized by comparable length and degree of difficulty we have elected to set a condition on the minimum initial distance between the two, that is:

$$dist(P, T) > d_{min}. \tag{13}$$

The choice of randomly generating the robot's initial position and the target location has been made in order to reduce the possibility for the robot to find workarounds to reach the target location (e.g.. move close to the edges of the environment).

To enforce a safety distance between the robot and the other elements of the environment and to take into account the physical dimensions of both the robot and the pedestrians, we set a safety radius R_R for the former, while for the latter we set a safety radius R_P as shown in Figure 3. Static obstacles instead have been grown in size to consider a safety turning radius, needed for obstacle avoidance manoeuvres.

In practice, in the grid-like simulated environment, the cells of the grid in which these elements stand are marked as occupied.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2022 doi:10.20944/preprints202212.0233.v1

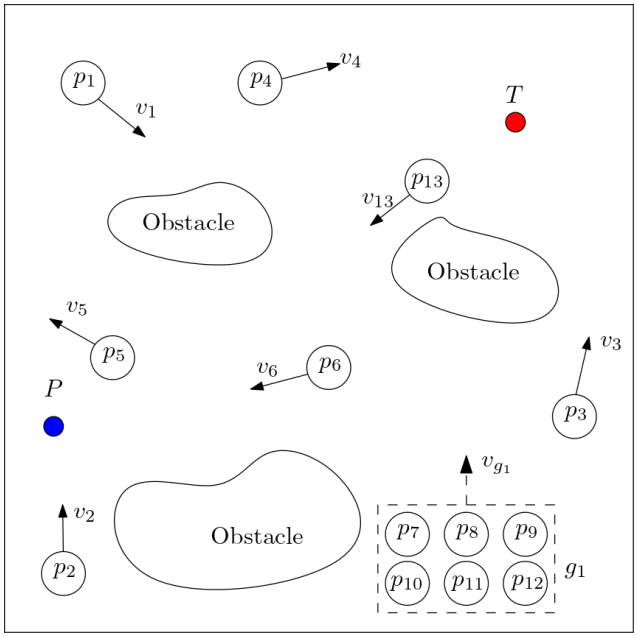


Figure 2. Schematic representation of the simulated environment: the robot spawning location and its target destination are highlighted in blue and red respectively; three static obstacles are represented. Each pedestrian p_i is indicated as a circle having its own speed v_i, while the group is represented with an enveloping dashed rectangle and depicted with g_i.

Table 1. Summary of the parameters describing the whole environment simulation, including robot, pedestrians and range sensor.

Parameter	Value	Parameter	Value
R_w	0.1m	R_p	0.3m
L	0.5m	R_R	0.6m
v_{max}	$1\,\mathrm{m\,s^{-1}}$	ω_{max}	$1\mathrm{rad\,s^{-1}}$
r_{min}	0.3m	r_{max}	10m
$\Delta\alpha$	$[-\pi/2, \pi/2]$	n_{rays}	135
$W \times H$	$20\,\mathrm{m} \times 20\,\mathrm{m}$	d_{min}	32m

2.4. Robot Perception

It is assumed that the WMR is equipped with a 2D LiDAR laser scanner, which grants perception to the robot and gathers information from the surrounding environment. This system is based on a ray-casting algorithm and enables the robot to detect objects within the LiDAR range. However, in order to allow the robot to react to dynamic obstacles such as pedestrians, the controller's policy is given in input the sequence of the last k LiDAR readings, where k is a tunable parameter.

The rays of the perceptual system mounted on the WMR span in a radial area around the robot that is defined by the sensor's range r_{max} and the scanning angle $\Delta\alpha$. The ray density in the scanning area is regulated by the scan resolution s_r, defined as $s_r = \Delta\alpha / n_{rays}$, where n_{rays} is the number of the rays. In Figure 3 we show a representation of the above setup: when a ray meets an obstacle, it returns the distance r_i of the intersection point,otherwise the scanner maximum range is returned. Since the angle α_i is known implicitly (in fact it's an arbitrarily defined value) the information about all intersection points is readily known in polar coordinates (r_i, α_i).

In order to discard false readings coming from the robot geometry, a minimum scanning range r_{min}, slightly greater than the robot, is introduced. In the simulated environment, we do not employ an ideal 2D LiDAR, but we consider every ray to be subject to false positive and negative readings with probabilities of p_{fp}, p_{fn}, respectively.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2022 doi:10.20944/preprints202212.0233.v1

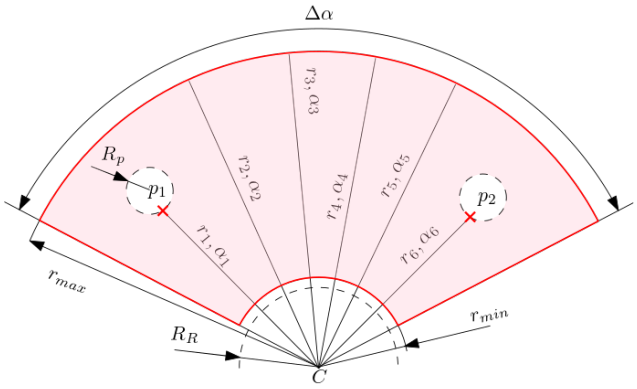


Figure 3. Working principle of the 2D LiDar range sensor. The figure shows a general scanning area, positioned ahead of the WMR, defined by $(r_{min}, r_{max}, \Delta\alpha, n_{rays})$, in which two pedestrians are detected.

The real WMR that we use has a scanner that only points in the forward direction of the robot. This increases considerably the complexity of the studied problem making it much more difficult for the agent to navigate the crowd. Furthermore, the scanner cannot have a full polar view of the space around the WRM due to geometry constraints and to the LiDAR location (see Figure 15). In Table 1 we report the parameters and specifications for the scanner and robot environment used in simulation.

2.5. Map Chunk Model

In Figure 4, we illustrate the robot perception model (referred to as *chunk model*) we used in the RL setup, described in Section 3.1.3. In the depicted scenario, both static obstacles and pedestrians are present. The scanning area of the robot, i.e. the space around it that is spanned by the LiDAR system, is split into n_q sections, each containing information about the closest object. This approach was designed in order to reduce the dimensionality of meaningful data associated with the environment's status and possibile robot collisions.

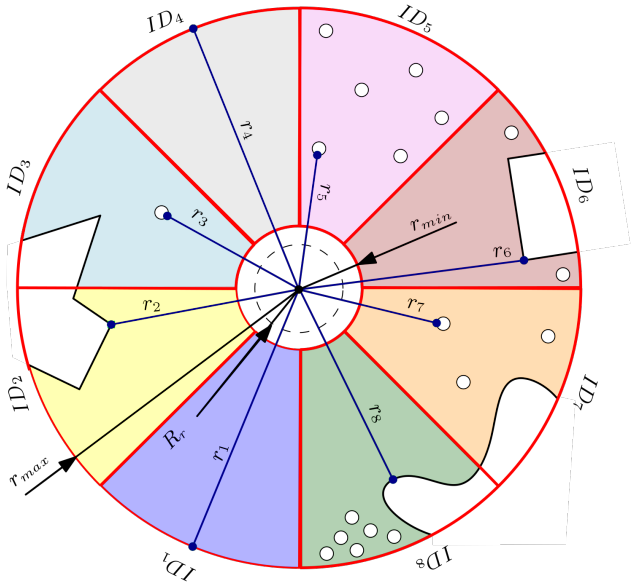


Figure 4. Schematic view of the chunking principle of the field of view of the robot. In this example the area is split into eight sections: for each section the distance from the closest obstacle/pedestrian is taken.

 doi:10.20944/preprints202212.0233.v1

With the above *chunking* process, we can obtain two perceptual outputs represented by the vectors d^p, if only pedestrians are considered, and d^m, when the closest of all map obstacles is taken (fixed obstacles, box edges and pedestrians), where

$$d_i^m, \quad i \in [1, \ldots, n_q] \tag{14}$$
$$d_i^p, \quad i \in [1, \ldots, n_q] \tag{15}$$

and $d_i^m \leq d_i^p, \forall i$.

Although the perception system only scans ahead of the robot with an 180° angle of view, the chunking process can be leveraged in two ways to speed up the training process of the controller. Indeed, we devise two auxiliary tasks that share weights with the controller NN but have different final layers for the specific problem:

- the first auxiliary task consists in estimating the position of all surrounding obstacles using past observations (here d^m is used);
- the second task instead optimizes a policy that maximizes a one-step reward penalizing states, where the robot is surrounded by pedestrians from multiple directions (d^p is used, since proximity to a fixed obstacle does not necessarily pose a collision threat).

3. Methodology

The problem of navigation in a dynamic crowded environment can be seen as a Markov Decision Process (MDP), and therefore solved with Reinforcement Learning techniques [35], in which an agent observes states s and performs actions a. In this setting, reinforcement learning can be used to find a policy, i.e. a mapping from states to actions, that controls the agent and optimizes a given criterion represented by the *reward function* associated with the MDP.

A MDP describes, in probabilistic terms, a transitions system defined by a tuple (S, A, P_a, R_a), where S is the state space, A the action space, $P(s'|s, a)$ the probability of transitioning to state s' if action a is chosen at state s, and $R(s, a, s')$ is the reward associated with the transition $s \xrightarrow{a} s'$. In the crowd navigation problem, the state is represented by the WMR state, the static obstacles configuration and the pedestrians dynamics. On the other hand, the agent actions are the possible signals the controller can send to the robot while the transition probability $P(\cdot)$ is defined by the joint dynamics of the robot and the pedestrians. In such a complex MDP model, it is not practically feasible to infer the probabilistic model $P(\cdot)$ due to the stochastic behaviour of pedestrians and the dependency on the number of agents involved. Therefore, we employ the RL framework and leverage the avility of deep neural networks to learn the optimal policy π.

In the next section we describe the crowd navigation MDP (Sec. 3.1), the RL algorithms we use (Sec. 3.2), the neural network architecture employed for the robot controller (Sec. 3.3) and details about the deployment of the distributed training environment we developed (Sec. 3.4).

3.1. Elements of the Markov Decision Process

When solving MDPs with RL, the definition of the MDP elements plays a critical role in making the problem feasible. In the following sections we describe how we model such elements in our work.

3.1.1. State Space

To enable robot navigation in a crowded environment, the state space S, i.e. the space of possible inputs, must be informative and include relative position, orientation and speed of the robot with regards to the target location, obstacles, pedestrians and other environment objects. Furthermore, the state observed by the controller needs to provide enough information to determine the motion of pedestrians. We can identify two main objectives in our task: (i) reach the desired target, (ii) avoid collisions with pedestrians

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2022 doi:10.20944/preprints202212.0233.v1

and objects. While these two goals are not completely independent, we can consider them separately in order to determine the information required to achieve each of the tasks.

If we assume no obstacles or pedestrians are present, the policy $\pi(s)$ would only require the robot inertial and dynamic information in order to reach the target state. So, we define the *internal robot state* as:

$$s_{rbt} = [x_T, \vartheta_T, v, \omega] \tag{16}$$

where, in this case, x_T and θ_T are relative distance and orientation between the robot and target point.

In order to avoid collisions, the robot policy has to know the dynamics of every solid element in the environment reference frame. In particular, using the limited information provided by the LiDAR sensor, position and speed of each element cannot be known exactly. However, it is possible to infer such quantities by providing the agent with past observations of the perceptual system and the evolution of its internal state.

For this reason, given p equally spaced LiDAR observations and a time window of m past observations, we define the *environment observation state* as:

$$s_{env} = \{s_1, \dots, s_m\} \tag{17}$$

where

$$s_i = (l_{1,i}, \dots, l_{p,i}, v_i, \omega_i) \tag{18}$$

In this notation, at time instant i, l_{ji} is the LiDAR detected distance of the j-th ray, with $j = 1, \dots, p$; on the other hand v_i and ω_i are respectively the linear and angular speed of the robot observed still at time i.

In conclusion, each element s of the State Space S is defined by the pair

$$s = (s_{env}, s_{rbt}) \tag{19}$$

3.1.2. Action Space

To keep the learning problem size manageable, we consider a discretization of the robot's actions. Specifically, for the linear and angular speed variations which the differential drive mobile robot can generate, we consider nine combinations of three values for each action. Hence the action a is composed as:

$$a = (\Delta v, \Delta \omega) \tag{20}$$

where

$$\Delta v \in \{-\Delta v_{max}, 0, \Delta v_{max}\} \tag{21}$$
$$\Delta \omega \in \{-\Delta \omega_{max}, 0, -\Delta \omega_{max}\} \tag{22}$$

3.1.3. Reward Function

In reinforcement learning methods, the design of the reward function plays a key role in defining the *hardness* of the learning problem. For example, a simple reward structure may only provide positive or negative feedback when the target is reached or a collision occurs, respectively. However, such a sparse signal hampers the learning capabilities of the agent and makes it much more difficult to achieve the optimal policy. Furthermore, a properly designed reward function can considerably speed up the convergence of the agent's policy.

The simulation environment we consider in this work is composed of a square shaped space where pedestrians have free access through the perimeter; conversely, for the robot, this boundary represents an obstacle. Static obstacles, for both the pedestrians and the robot, are placed randomly within the environment and have various random polygonal shapes (see Figure 5). At the beginning of each training episode, a new map is generated

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2022 doi:10.20944/preprints202212.0233.v1

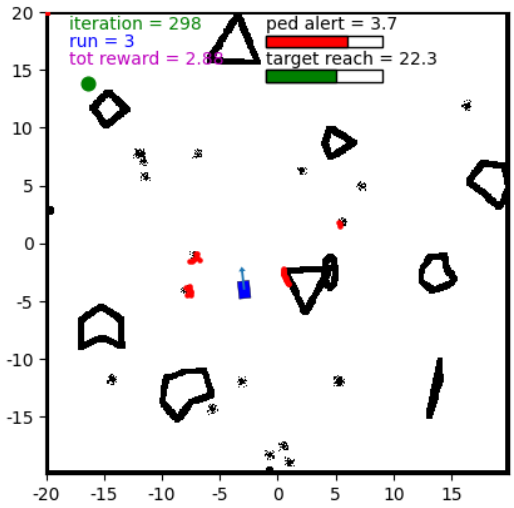


Figure 5. Learning environment: the blue rectangle represent the robot and the arrow its current heading; the red dots the LiDAR scanner readings; the green circle represent the target location the robot must reach; finally, in black are represented the obstacles. Obstacles can be either be static and dynamics (black dots represent pedestrians).

with random target and robot initial coordinates satisfying the minimum initial distance. A trajectory, i.e. a single episode run, is considered to be successful when target coordinates are reached and no collision, with either pedestrians or obstacles, occurred.

Taking into account the previous considerations, we defined a reward function R, that in the MDP framework, provides a signal to the agent after each transition $s \xrightarrow{a} s'$. In particular, we consider three possible scenarios:

- in the state s', the episode stops because the robot has successfully reached the target, hence a positive reward is provided: $R = +K$;
- in the state s', the episode ends because a collision occurs or the simulation time has expired, i.e. the maximum number of environment interactions has been reached, hence a negative reward is given: $R = -K(0.75 + FD)$;
- a terminal state is not reached and the robot can keep progressing and receives a reward $R = +k(DB - SM - PPM)$

where K, k are final and intermediate rewards constants, and FD is the robot's distance to the target achieved at the end of the episode that has been normalized with respect to the maximum possible distance in the environment space (i.e. the square diagonal). The other terms in the third case play the role of providing intermediate bonus and penalty components, they are defined as follows:

- the *direction bonus* is $DB = 1$ if the distance from the target has decreased in the current transition after action a, else is set to $DB = 0$;
- the *saturation penalty* is $SP = 1$ if the actuator has been saturated as an effect of action a, else it is $SP = 1$;
- the *pedestrians proximity penalty* is given by $PPM = c_P \sum_{i=1}^{n_q} (1 - d_i^p)^3$ where c_P is a constant and d_i^p was defined in (15) as the normalized distance of the closest pedestrian in the i-th chunking sector. We consider the term d_i^p instead of d_i^m, because it only considers pedestrians, the rationale being that a trajectory running close to an obstacle in order to avoid pedestrians should not be penalized. Finally, the cubic exponent ensures that pedestrians farther than $\approx 1/3$ of LiDAR range don't have a negative impact on the intermediate rewards.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2022 doi:10.20944/preprints202212.0233.v1

3.2. Reinforcement Learning Architecture

In this paragraph we show how the NN has been trained with two classical RL techniques: parallel Deep Q-Learning (DQL) and Asynchronous Advantage Actor Critic (A3C).

3.2.1. Deep Q-Learning

Deep Q-Learning was the first RL method to be applied to deep NNs [28]. It is derived from Q-Learning, a classical reinforcement learning algorithm to estimate the *State-Action value function* $Q^{\pi}(s,a)$, also called Q-function, of the optimal policy π^*. The Q-function of a given policy π estimates the average of the discounted return, i.e. the sum of all future rewards, achieved by using π to choose actions after starting in state s and applying action a. In the case of π^*, the value function $Q^*(s,a)$ provides the return of the optimal policy and satisfies the *Bellman optimality equation*,

$$Q^*(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q^*(s',a')\right] \tag{23}$$

where γ is the discount factor for future rewards, and the expectation is computed with respect to the probability distribution of rewards r and the environment dynamics. Ideally, if $Q^*(s,a)$ is known, the optimal policy selects action a as:

$$a = \underset{a'}{\operatorname{argmax}}\, Q(s,a') \tag{24}$$

Mnih et al. have shown in their research [28] that the iterative version of (23) converges to Q^*:

$$Q_{i+1}(s,a) \leftarrow \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a')\right] \tag{25}$$

hence it is possible to learn the optimal policy by improving iteratively the estimated Q-function. When the Q-function is approximated using a NN ($Q(s,a;\theta) \approx Q^*(s,a)$), the convergence in (25) is achieved by solving a regression problem. In particular, observed transitions (s,a,r,s') are used to provide a target value – this practice is known as *bootstrapping* – by leveraging the approximate Q-function to estimate future rewards. This step acts as ex-post assessment of the value of action a, and results in the loss function:

$$L(\theta_i) = \mathbb{E}\left[(y_i - Q(s,a;\theta_i))^2\right] \tag{26}$$

where the expectation has to be computed over MDP variables s,a,r,s', and the Temporal Difference target y_i is defined as

$$y_i = r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) \tag{27}$$

To estimate the expectation in Eq. (26), the DQL algorithm exploits "experience replay" [36] that allows to minimize the loss $L(\theta)$ via Stochastic Gradient Descent (SGD). This technique requires that observed transitions $(s_t, a_t \rightarrow s_{t+1})$ are stored in a circular memory, that is then used to train the Q-function with mini-batch SGD. This approach has two main advantages: (i) each transition is used for multiple updates, (ii) variance is reduced by using uncorrelated transitions in a batch.

In DQL, the balance between exploitation and exploration during training is obtained by defining using an epsilon-greedy policy [37] to choose action a during interaction with the environment:

$$a_t = \begin{cases} a = & \operatorname{argmax}_{a'} Q(s,a') & , & \text{w.p. } \epsilon_t \\ a \sim & \mathcal{U}\{a_1, a_n\} & , & \text{w.p. } 1-\epsilon_t \end{cases} \tag{28}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2022 doi:10.20944/preprints202212.0233.v1

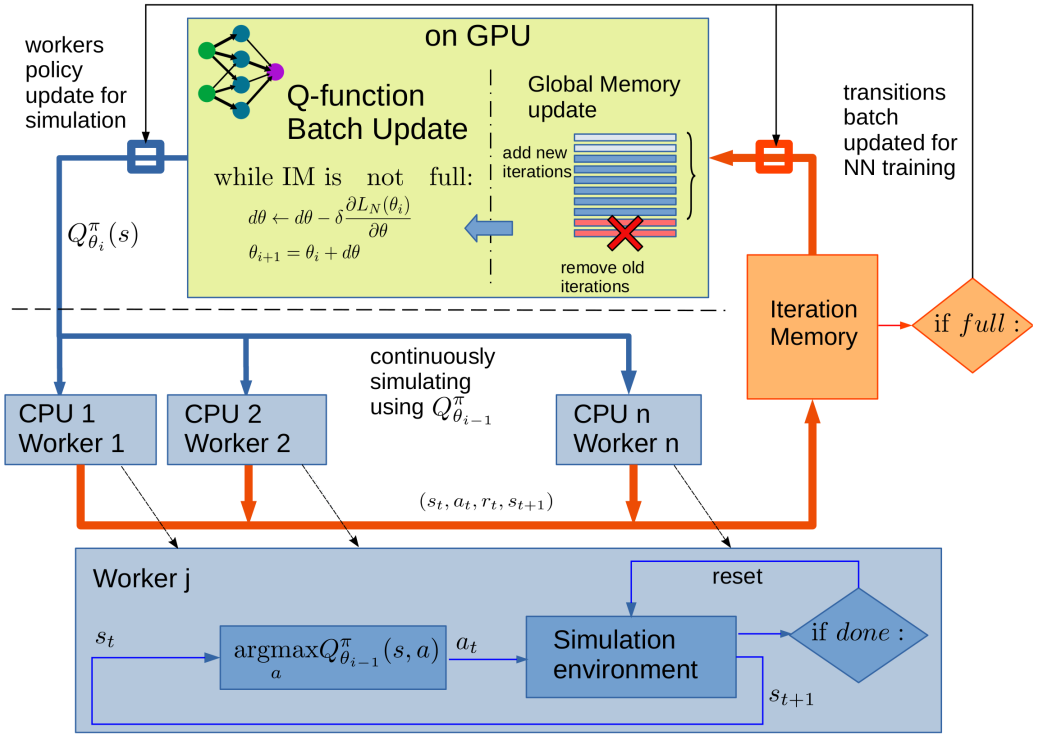


Figure 6. Schematic representation of the multi-node asynchronous DQL algorithm. The upper part of the diagram (NN update) and the lower (agents simulations) are performed in parallel on different nodes, at the end of each iteration data is shared (NN coefficients from the updater node and transitions memory from the simulating nodes).

and ϵ_t shrinks at each iteration with a factor $0 < c_\epsilon < 1$ until a minimum value $\epsilon_{\min}$ is reached:

$$\epsilon_t = \max(c_\epsilon \cdot \epsilon_{t-1}, \epsilon_{\min}) \tag{29}$$

In this paper a multi-node asynchronous version of DQL has been implemented (see Figure 6). The training is divided in *iterations*, which are defined by the number of transitions which are simulated with a given Q-function version. During each iteration i, the reference Q-function used to run the simulations ($Q^{\pi}_{\theta_{i-1}}$) is exported to multiple environment instances on different worker nodes, and the transitions are stored in an *iteration memory* (IM). The time required to fill the IM is used by the main thread to update the NN weights on the GPU with batches extracted from the "global memory" GM, providing the new Q-function $Q^{\pi}_{\theta_{i-1}}$. The oldest transitions within the GM are replaced with the IM, so the next iteration can be run.

This implementation has multiple advantages: (i) at all times all computational resources are exploited, (ii) fixed sized iterations are a natural choice for evaluation and storage of training progress, (iii) memory relocation from CPU to GPU occurs only between iterations, (iv) when deploying on a cluster, local memory capacity at each worker node can be adapted to the available resources at each node. An overview of deployment aspects is provided in Section 3.4.

3.2.2. Asynchronous Advantage Actor Critic (A3C)

The A3C algorithm [29] has been implemented for the same discrete action space specified in Sec. 3.1.2. This algorithm is a Policy Gradient (PG) method, i.e. the policy is directly optimized by computing gradients that update its weights in order to maximize the objective function:

$$J^{\pi}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta, P}[G(\tau)] \tag{30}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2022 doi:10.20944/preprints202212.0233.v1

that is the average return achieved by the policy and can be estimated by sampling trajectories form the environment using the policy π_θ:

$$J^\pi(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T_i-1}\gamma^t r(s_{i,t}, a_{i,t}) = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T_i-1} G_{i,t} \tag{31}$$

where $G_{i,t}$ is the compounded sum of all rewards in a given trajectory, from any intermediate point until the end.

A3C is derived from the REINFORCE algorithm (or Vanilla PG) in which (31) is maximized through SGD. The modifications introduced in A3C are mostly aimed at reducing the high variance which characterizes Monte Carlo sampling. One way to reduce variance is to consider the *advantage* $A(s_t, a_t)$ of taking a certain action with respect to a baseline, instead of $G_{i,t}$ computed from the raw rewards (whose signs may be arbitrary). The actor-critic algorithms implement a version of this concept in which the baseline is provided by a *critic* function, that approximates the state-value function $V_\phi^\pi(s)$ for policy π (the *actor*). Moreover, causality allows the advantage at the t-th instant to be calculated without considering past rewards. Using Actor-Critic with "reward to go", the Advantage becomes:

$$A(s_t, a_t) = r - V_\phi(s_t) = \left(\sum_{t'=t}^{T-1}\gamma^{t'-t} r(s_{t'}, a_{t'})\right) - V_\phi(s_t) \tag{32}$$

The overall loss function to be minimized becomes

$$L_{\text{A3C}} = -J^\pi(\theta) + J^\phi(\theta_c) - \beta H^\pi(\theta) \tag{33}$$

where the *advantage loss* is

$$J^\phi(\theta_c) = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T_i-1} A(s_{i,t}, a_{i,t})^2 \tag{34}$$

and $H^\pi(\theta)$ is the entropy component added to avoid early convergence to local minima

$$H^\pi(\theta) = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T_i-1}\left(-\sum_a \pi(s_i, a)\log\pi(s_i, a)\right) \tag{35}$$

The formula for the gradient which maximizes (31), can be approximated as:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T_i-1}\nabla_\theta \log\pi_\theta(a_{i,t}, s_{i,t}) A(s_{i,t}, a_{i,t}) \tag{36}$$

The adopted implementation is schematically represented in Figure 7: the A3C configuration is obtained by running the threads $i = 1, \ldots, N$ in parallel, accumulating the gradients computed on different nodes and performing the optimization step locally in the main thread. In order to make performances between the different RL methods comparable, the code maintains the "iteration" structure described in Section 3.2.1, effectively ending the iteration when the total number of simulated steps reaches the "Iteration Memory" size.

3.3. Neural Network Topology

The NN that we use is comprised of three main parts (see Figure 8):

- a 3-layers Convolutional Neural Network (CNN) which "reads" the surrounding map by evaluating the information from the 2D LiDAR range sensor, together with the robot longitudinal and rotational speed (s_{env});

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2022 doi:10.20944/preprints202212.0233.v1

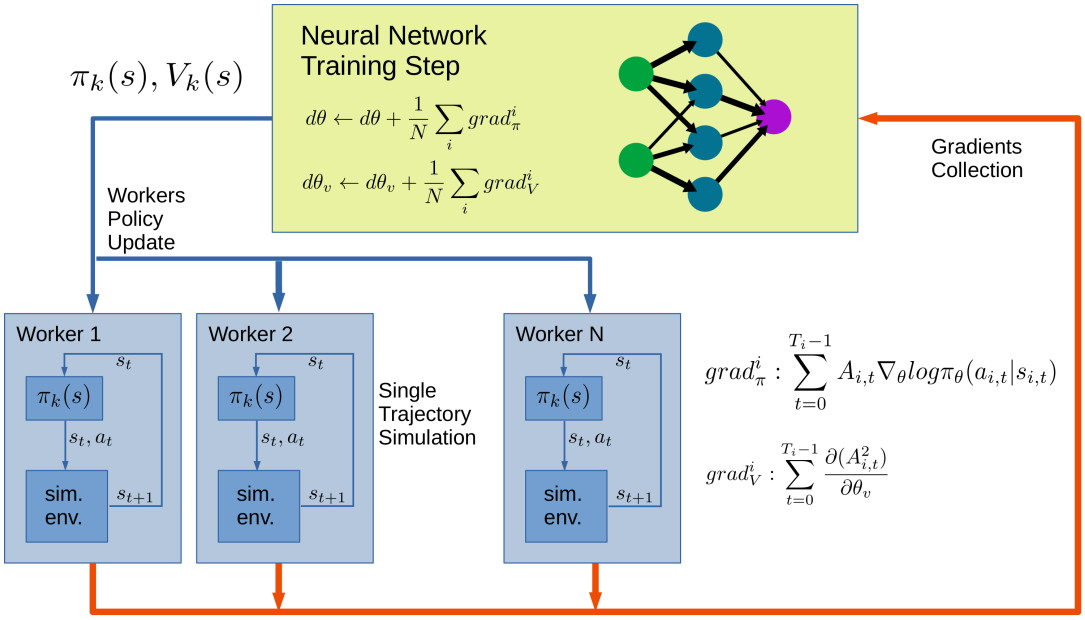


Figure 7. A3C algorithm scheme. Worker nodes simulate with action a chosen according to policy $\pi_k(s)$. Gradients are accumulated until a transitions threshold is reached, then NN weights are updated via backpropagation and policy π_{k+1} is obtained.

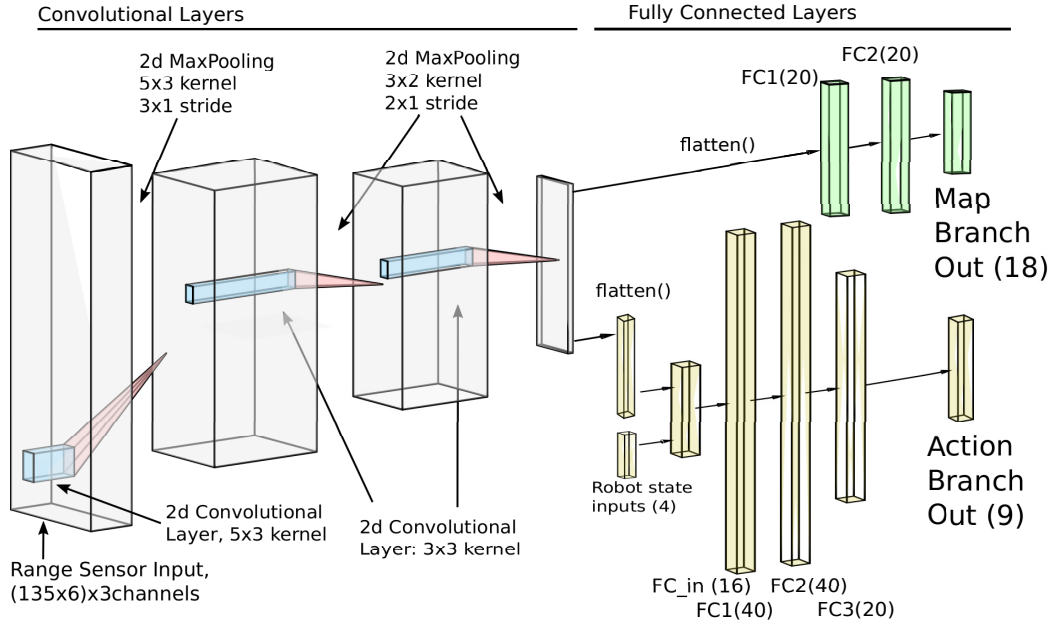


Figure 8. Schematic representation of the NN used in the DQL case $Q(s)$. For policy learning $\pi(s)$ a SOFTMAX layer is added at the end, while in case of the state-value function $V(s)$ the last layer has a unique output.

- a Fully Connected Neural Network (FCNN) with 3 hidden layers (that we call the *action branch*), that defines a navigation policy using robot state information s_{rbt} and the CNN outputs;
- a secondary FCNN which takes the the CNN outputs as inputs and returns the estimated distance of the closest identified object in each direction (referred to as *map branch*). For this purpose, not only the visible angle is considered but the full 360° area surrounding the robot, appropriately divided into n_q sectors. This output is only used during training to accelerate the map reconstruction convergence.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2022 doi:10.20944/preprints202212.0233.v1

Since the sensory input provided to the robot's controller is three-dimensional with length p, width m and $n_{ch} = 3$ channels, where the components are filled with the data in (17), the CNN is a natural choice for the range sensor inputs, given its continuous spatial distribution. The CNN is three-dimensional because the third dimension represent the time, i.e. the third dimension is used to stores the past observations. Furthermore, v and ω components are taken constant along the tensor length in the complete controller's input. The convolutional architecture applies a 2-dimensional convolution over the normalized input tensor built from s_{env} and is designed with internal layers that have 16 and 10 channels respectively.

In our neural network, each CNN layer consists of:

- a 2d convolution with kernel 5×3 for the first layer, 3×3 for the following ones;
- a ReLU activation function;
- a 2d MaxPooling layer with stride 3×1 for the first layer, 2×1 for the following ones;
- a normalization layer across the features dimension.

In the map branch, the flattened CNN output is passed to two 20-neurons fully connected layers that return the sector based obstacle distance used during training:

$$\hat{d}_i^m, \quad i \in [1, \ldots, n_q] \tag{37}$$

For the action branch, the CNN outputs are flattened and stacked with the internal robot state s_{rbt}. The resulting new tensor does not have intrinsic spatial features to be detected, therefore a FCNN is used (again with ReLU activation functions and batch normalization after each intermediate layer). The final number of outputs corresponds to the number of actions, as explained in 3.1.2.

When the NN is used for policy learning (as in the A3C case), a SOFTMAX layer is added after the final one, whereas for value function $V(s)$ the last layer consists of a single node.

To summarize, from an input/output perspective, the NN can be seen as 2 different functions that share some weights and return objects p and $\hat{d}^m$:

$$\overline{p}, \hat{d}^m = \pi(s_{env}, s_{rbt}) \tag{38}$$

with

$$\overline{p} = [p(a_1|s_{env}, s_{rbt}), \ldots p(a_{n_a}|s_{env}, s_{rbt})] \tag{39}$$

3.3.1. "Map Loss" Addition for Faster Convergence

RL loss functions (26), (33) can be complemented with additional information, in order to enhance the learning process. In the crowd navigation problem, the dual nature of the information which should be learned (map and strategy) allows for the use of extra-information, besides the rewards.

Here we define the additional component "map loss" as:

$$L_{map}(\theta_i) = \sum_{i=1}^{n_q} (\hat{d}_{\theta_i}^m - d_i^m)^2 \tag{40}$$

where the distances d_i^m are extracted from the map chunk model 2.5. The back-propagation effect with L_{map} is to ensure that the CNN learns faster how to exploit present and past information, in order to infer as accurately as possible the position of all fixed and moving obstacles, including those in the shadow cone behind the robot.

3.4. Parallelization on HPC Cluster

The NN training requires computational resources compatible with the parallelization proposed for both A3C (CPUs) and DQL (CPUs and GPUs) algorithms. In this work it was possible to exploit a HPC cluster that operates in batch mode using the Slurm Workload

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2022 doi:10.20944/preprints202212.0233.v1

Management System [38], wich is designed to work with MPI (MessagePassing Interface) [39] as parallelization library.

As the NNs code was developed using PyTorch package for Python, a direct conversion to MPI would be burdensome and inefficient. We elected instead to use Ray [40], a universal API for building distributed applications with particular focus on reinforcement learning applications, which allows to make minimal changes in the code through Python decorators in order to make it parallel.

Ray is a cluster itself, which being particularly lightweight is instantiated for each job run as a dedicated instance. A Ray-enabled job requires therefore a Ray head node, which coordinates the computation of the other tasks. Each Ray task registers to the head node by connecting to a web service on its IP address, on a given port, and from which it receives instructions on what to compute. Ray tasks will also send back the results to the Ray head node.

In order to make Ray work on Slurm, it is therefore required to run a cluster-in-a-cluster, and to allocate an extra CPU for the Ray head node so that, for a n degree of parallelization, the Slurm requirement must be of $n+1$ CPUs. Also, the Ray head node must be up and running before the ray tasks try to connect. To orchestrate this Slurm job structure a SBATCH environment was used, where the Ray head node is first run on a given port, retrieves its IP address, waits a few seconds to let the head node to boot up, and then runs the other tasks providing them the IP and address and port of the Ray head node. After this setup is complete, the computation starts nearly flawlessly, allowing the code to run in coarse-grained parallelism [41] on the cluster.

4. Results

4.1. Training Results

The training settings chosen for the DQL and A3C algorithms allow to compare the RL performance in terms of convergence speed and reward outcome.

Table 2. Main Simulation hyper-parameters used for the simulated environment and of the training of the NN

Parameter	DQL	A3C
pedestrians density	0.04 ped/m^2	
scan noise: lost scans	0.5%	
scan noise: corrupt scans	0.2%	
simulating agents	25	
steps per iteration	25000	
minibatch size	256	-
n. epochs per iteration	600	-
optimizer	ADAM	
initial learning rate	$1e^{-5}$	$2e^{-3}$
γ discount factor	0.9	
β (AC enthropy coefficient)	-	0.05

However, due to the training setup that we employ, it is not possible to compare directly the training simulations. In fact, at iteration t, the A3C training algorithm generates a certain number of full trajectories, starting from random initial conditions and following policy π_t.

On the other hand, the ϵ-greedy approach in DQL shown in Eq. (28) doesn't allow for a straight-up comparison of all training trajectories. Hence, in order to perform a statistical analysis on the outcome of the simulated trajectories and on the total cumulative rewards, with DQL we simulate one iteration every 5 being using a pure exploitation policy (24).

In both scenarios, at each iteration approximately the same number of transitions $s_t, a_t \rightarrow s_{t+1}$ is calculated. Trajectories in general have variable duration, so an exact

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2022 doi:10.20944/preprints202212.0233.v1

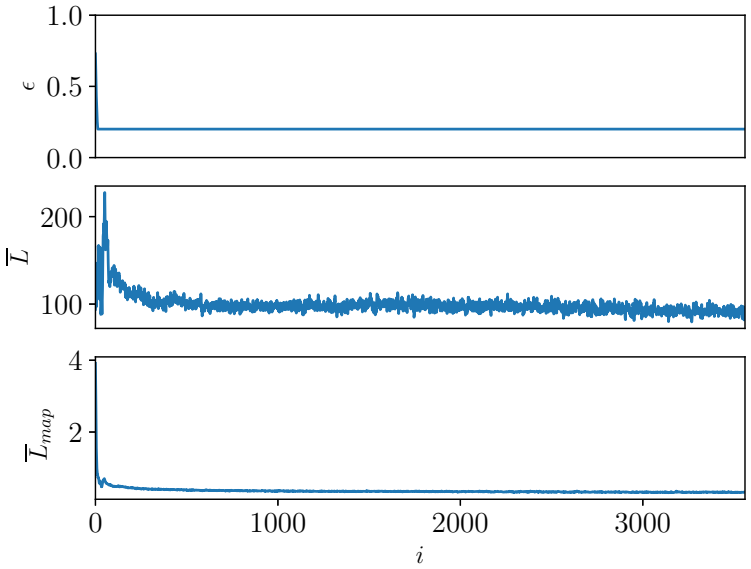


Figure 9. DQL training variables evolution: in the top row for ϵ; in the middle row for the average of the q-value loss $\overline{L}$; in the bottom row for the average of the "map loss" $\overline{L}_{map}$.

amount of transitions per iteration can not be guaranteed. Finally, simulation settings during training for RL and navigation environment are shown in Table 2.

For each iteration in the horizontal axis, training statistic shown in this section (Figure 9, 10, 11, 12) are computed averaging out the results obtained for all simulation steps/trajectories in the DQL and A3C cases respectively.

In Figure 9 it is shown the evolution over the DQL training process of the following quantities: (i) in the top row the ϵ_t that regulates the degree of exploration in action selection, (ii) in the middle row the average q-value loss $\overline{L}$ (26) on one transition for the current iteration, and (iii) in the bottom row the average loss on the map estimation task $\overline{L}_{map}$ (40). It can be seen also that both the average losses $\overline{L}$ and $\overline{L}_{map}$ tend to converge to a limit value, which is necessarily greater than 0, given that half of the map evolution can only be estimated and not "seen", due to the 180 degrees opening of the LiDAR sensor.

Still regarding the DQL training process, in Figure 10 are reported the performance indicators for the algorithm. It must be noted that only iterations with pure exploitation have been considered, e.g. one every 5: that is iterations 1, 6, 11, and so on. More precisely, the top row shows the evolution of the average of the duration of the single run in the simulated environment $\bar{t}_c$; the middle row shows the evolution of the cumulative reward $\overline{G}$; while the bottom row shows a statistic of the simulation termination reason ratio. In this figure it can be seen also that the NN learns pretty fast and the average cumulative reward $\overline{G}$ converges to positive values; while after iteration $i \approx 1000$ the success ratio seems to be stable and not have substantial changes, thus indicating that the NN is not learning anymore.

On the other hand, the training findings that has been obtained for the A3C training process are shown in Figure 11 and Figure 12. Specifically, Figure 11 shows the evolution of the following quantities: (i) in the first row the average of the advantage loss $\overline{J}^{\phi}$ in Eq. (34); (ii) in the second row the average of the policy loss $\overline{J}^{\pi}$ in Eq. (31); (iii) in the third row the average map loss on the map estimation task $\overline{L}_{map}$; (iv) in the bottom row the normalized average of the entropy $\overline{H}^{\pi}$ in Eq. (35). It can be seen that the $\overline{J}^{\phi}$, $\overline{J}^{\pi}$ and $\overline{L}_{map}$ and $\overline{H}^{\pi}$ all converge to a certain value. Between the losses it can be noted that the "map loss" is the first to converge, although it remains at a considerably higher level than the DQL case, due to the higher performance of large batches training for supervised classification of image like data.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2022 doi:10.20944/preprints202212.0233.v1

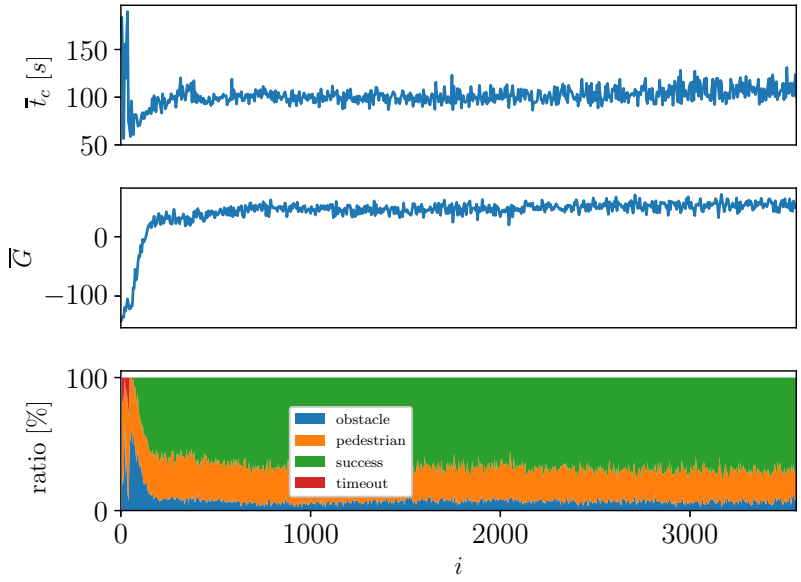


Figure 10. Summary of the DQL performance: in the top row the evolution of the average of the duration of the single simulation run $\bar{t}_c$; in the middle row the evolution of the average of the cumulative reward $\overline{G}$; in the bottom row the evolution of the statistic of the simulation termination reason ratio.

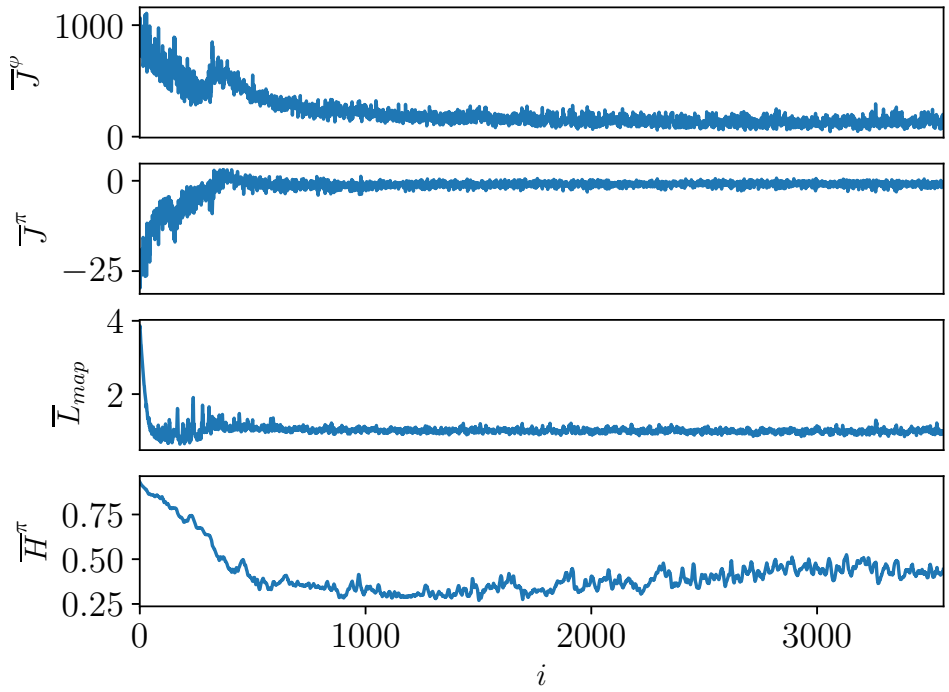


Figure 11. A3C training variables evolution: in the top row for the average of the advantage loss $\overline{J}^{\phi}$; in the second row for the average of the policy loss $\overline{J}^{\pi}$; in the third row for the average of the "map loss" $\overline{L}_{map}$; in the bottom row for the average of the normalized entropy $\overline{H}^{\pi}$.

Still regarding the training process that has been done with the A3C algorithm, in Figure 12 the performance of these algorithms is shown. Specifically, in the top row the evolution is shown of the average of the duration of the single run in the simulated

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2022 doi:10.20944/preprints202212.0233.v1

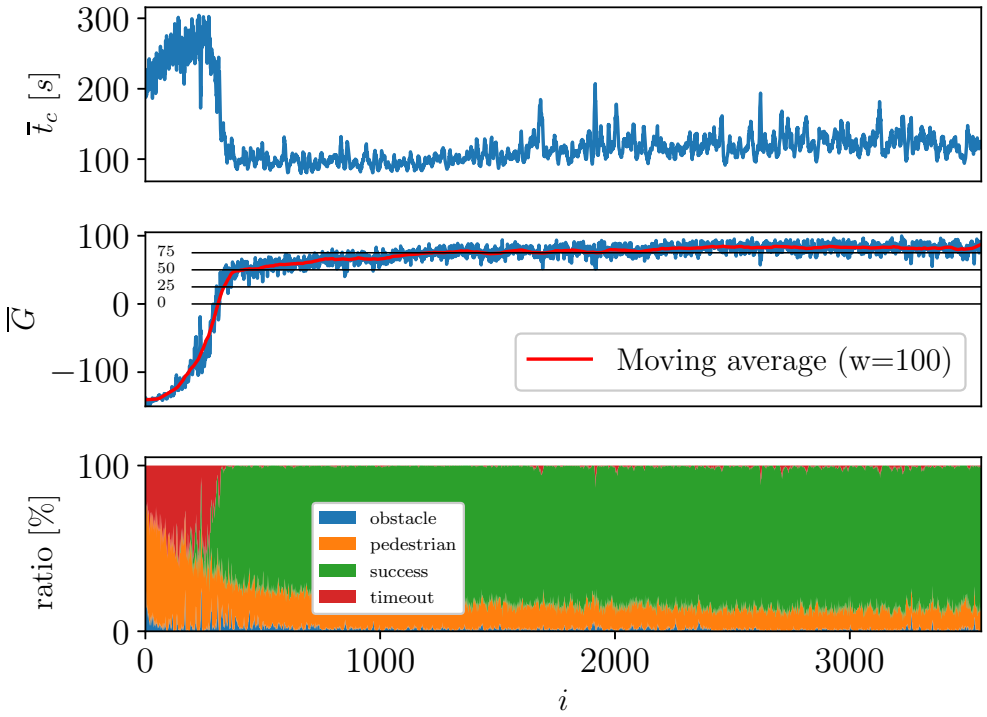


Figure 12. Summary of the A3C performance: in the top row the evolutin of the average of the duration of the single simulation run $\bar{t}_c$; in the middle row the evolution of the average of the cumulative reward $\overline{G}$; in the bottom row the evolution of the statistic of the simulation termination reason ratio.

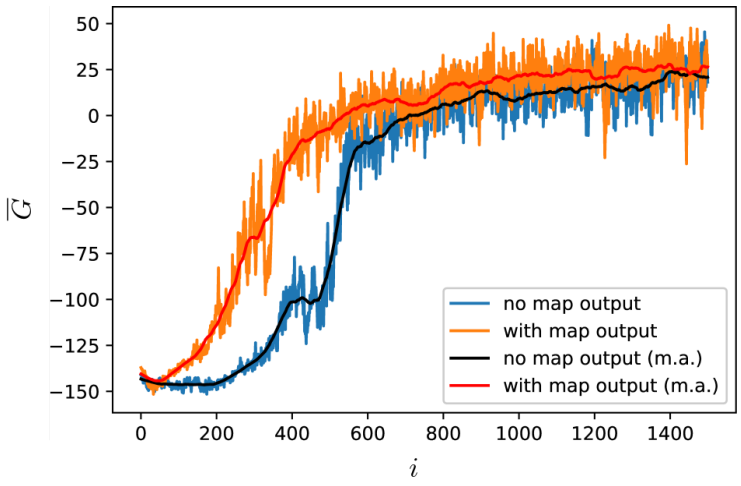


Figure 13. A3C Average cumulative reward with and without "map output". First 1500 iterations.

environment $\bar{t}_c$; the middle row shows the evolution of the cumulative reward $\overline{G}$, together with the filtered curve using a moving average filter with window size $w = 100$ (red curve); the bottom row shows a statistic of the simulation termination reason ratio. In this figure it can be seen that the NN compared to the one trained with DQL algorithm is slower to learn a strategy, however after it starts to learn, it shows a region with constant learning, until it reaches convergence; after this there appears to be a region in which there is no substantial performance improvement. Nevertheless, it is apparent that this algorithm compared to DQL exhibits a higher success ratio.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2022 doi:10.20944/preprints202212.0233.v1

Finally, in order to evaluate the effects of the introduction of the additional loss component L_{map} defined in Eq. (40), a sample training with the A3C algorithm both with and without this element has been performed. The findings are shown in Figure 13, which compares the convergence speed of the A3C for both cases in terms of average cumulative reward $\overline{G}$. The average cumulative reward is shown with an orange curve for the case of using the auxiliary task and with a red line its filtered values by using a moving average filter with window size $w = 100$; while the same are shown with a blue and purple curves for the case of not using the auxiliary task. From the figure it can be clearly seen that the auxiliary task is especially helpful during the initial phase of the training process, and bringing as a result the speed up of the learning by approximately 200 iterations. Finally, the addition of the additional supervised task is a key feature allowing the NN to learn good features representations for the control problem.

4.2. Testing Results

In order to quantify and evaluate the performance of the controllers – for the NNs trained with both DQL and A3C algorithms – it has been elected to perform a validation testing campaign. More precisely, the trained NN has been extracted and loaded in the simulated model of the robot, and used to produce the proper actions for the robot, leading it from a starting point safely to the target. Specifically, the NN version chosen to be extracted, for both DQL and A3C, in order to have a coherent comparison, is the one represented by the last iteration $i = 3560$. Additionally, for both the NNs the testing process has been evaluated for four different difficulty levels, i.e. the pedestrians density in the simulated environment. The mapping between the difficulty levels and the pedestrian density in the environment is listed in Table 3. Moreover, each of the validation process that has been conducted refers to an average number of $n \approx 10000$ complete simulation runs that has been performed in the simulated environment.

The findings obtained from the testing processes are shown in Figure 14 in form of bar plots. Figure 14a depicts the success ratio, using the DQL trained controller, as a function of the difficulty of the problem, i.e. the density of pedestrians in the environment. In the same figure are shown also the single failures ratios, that are the collision with pedestrian ratio, the collision with obstacle ratio and finally the timeout ratio. Moreover, Figure 14b shows the same, but in this case it has been used the NN which has been trained using the A3C algorithm. Both figures provides and shows a qualitative and quantitative comparison of the capabilities of both controllers, while the exact values that has been obtained, and graphically shown in Figure 14, are listed in Table 4.

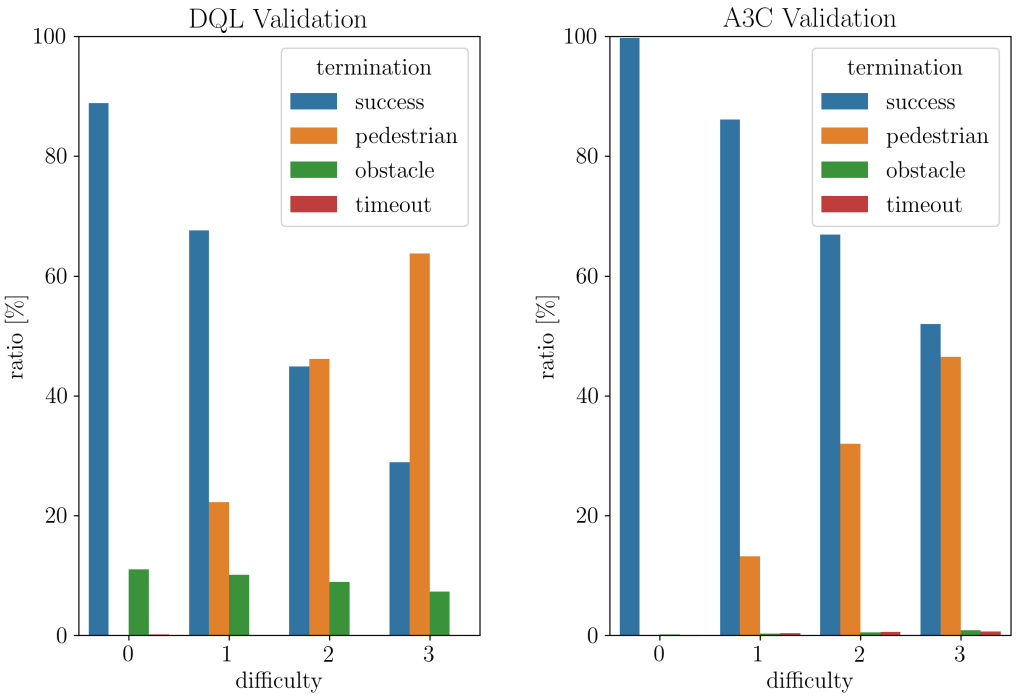
Table 3. Mapping between the difficulty levels and the pedestrian density in the environment.

Difficulty	Pedestrian density [peds/m^2]
0	0
1	0.02
2	0.08
3	0.1

As expected, for both validated NNs the success ratio tends to decrease as pedestrian density increases. This is due to the increased complexity of the problem, such that the robot finds itself more often in situations in which it can not avoid collisions. When the pedestrian density increases, the obstacle collision ratio exhibits a positive trend. This is due to the robot trying evasive maneuvers to avoid pedestrians, ending up in collisions with close obstacles as a result of that.

It is possible to note remarkable differences between the two trained NNs: in the DQL case the pedestrians collision ratio is greater than the A3C one for all difficulty levels. Regarding the DQL trained NN, it has been noted that only for the case of difficulty 0 the simulations end for timeout reasons, however with an insignificant value of 0.17%. For the remaining three cases the timeout termination criteria is never reached, hence the

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2022 doi:10.20944/preprints202212.0233.v1



(a) **(b)**

Figure 14. Barplots indicating the performances of the trained NNs and obtained with the validation process: in (a) the barplots showing the success, pedestrian collision, obstacle collision and timeout ratios, respectively for the DQL trained NN; in (b) the barplots showing the same but for the case of the A3C trained NN.

simulations end only because collisions either with static obstacles or pedestrians. The same cannot be said for the case of the A3C trained NN, in fact for each difficulty level, it shows some timeout failure ratio. However, it must be said that this termination failure ratio is always below the 1% of the cases. Moreover, as for the case of obstacle collision ratio, among the different difficulty levels, the termination failure ratio shows a positive trend, and it is likely because the robot tries many evasive manoeuvres leading it to prefer moving until timeout is reached rather than colliding.

The A3C trained NN shows in general better performances with respect to the DQL, thus confirming that policy gradients algorithms are better suited in order to tackle these kind of complex problems. Indeed, among the different difficulty levels the A3C algorithm is able to generalize in a better way. Additionally, the not much satisfying performances exhibited by the DQL trained NN, especially for higher difficulty levels, suggest that the training algorithm might have reached a local minimum. This is suggested by the graph in 10 where it can be seen that already around iteration $i = 200$, the DQL trained NN has learned a strategy, whereas the A3C keeps on learning and improving its performances.

Table 4. Summary of the results for the validation campaign conducted over the two trained NNs

	Success [%]		Pedestrian [%]		Obstacle [%]		Timeout [%]	
Difficulty	DQL	A3C	DQL	A3C	DQL	A3C	DQL	A3C
0	88.83	99.71	0.00	0.00	11.00	0.17	0.17	0.12
1	67.62	86.13	22.28	13.20	10.10	0.31	0.00	0.36
2	44.92	66.91	46.145	31.96	8.93	0.55	0.00	0.58
3	28.92	52.01	63.75	46.50	7.33	0.85	0.00	0.63

 doi:10.20944/preprints202212.0233.v1

5. Experimental Validation

In the following section the results are reported of the experimental validation campaign that has been conducted. It must be said that for the experimental validation we elected to use the NN which has shown overall better performance in Sec. 4.2, i.e. the A3C-trained NN. More precisely, we chose to investigate two different use case scenarios:

- *Case 1. No pedestrians*: the environment is composed only by static obstacles. The robot, through its trained controller, must be able to navigate the environment and it's required to move from a starting point $\mathbf{P_i}$ to a target point $\mathbf{P_f}$, safely. This required motion can happen either by doing a point-to-point motion, i.e. providing only the terminal point location, or by doing a path following motion i.e. providing a set of way-points the robot must pass through.
- *Case 2. With pedestrians*: the environment is composed of both static obstacles and pedestrians moving in the environment. The mobile robot as for the previous case must be able to reach a target location safely, but at the same time being capable of avoiding incoming pedestrians.

In order to carry out the experimental campaign we have elected to use a Neobotix MP-500, which is a ROS-Enabled differential drive mobile robot shown in Figure 15. Perception is provided by a SICK S300 bi-dimensional LiDAR Scanner, which is placed on the front side of the robot, and provides the sensor readings needed by the NN to take actions. In order to be able to control the robot, an appropriate ROS package has been developed and deployed on the robot onboard computer. Right before deployment of the ROS package on the experimental robot, the package combined with the trained NN, has been tested in the Gazebo [42] dynamic simulation environment, in order to identify possible problems or malfunctions. The package role is primarily to gather sensors data, pre-process and feed them to the previously loaded NN and finally compute and apply the robot speed corrections, based on the actions chosen following the NN policy. Moreover, the package implements the following additional functionalities: mapping the environment, performing Simultaneous Localization and Mapping (SLAM), performing robots pose estimation and correction of the robot pose. Additionally, it allows also the user to specify through the software Rviz the single target point $\mathbf{P_f}$ the robot has to reach, or a set of way-points the robot must pass through. Otherwise, it allows the user to load from file a predetermined path. The ROS package as it has been seen, it has been designed in order to be general and can be therefore used both on real robot and in Gazebo dynamic simulator.

Figure 15. The elected differential drive mobile robot used in the experimental campaign: a Neobotix MP-500.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2022 doi:10.20944/preprints202212.0233.v1

5.1. Case 1. No pedestrians

For this use case scenario it has been elected to conduct the experimental validation tests for both the point-to-point motion and the motion through way-points. Specifically, the tests have been conducted in an indoor environment, that is the C6 laboratory of the University of Trieste.

In Figure 16 it can be seen the occupancy bi-dimensional map of part of the laboratory environment, which has been obtained by leveraging the mapping functionality provided by the ROS package, at experimental testing setup. The generated map shows the presence of many static obstacles. These can be structured as well as represented; however there are some obstacles that the LiDAR scanner was not able to detect.

Figure 16, refers to the test case for the point-to-point motion. In this case the robot starts from the top left corner of the laboratory environment (indicated with the green “START" label), and must reach the commanded target point (indicated with the red “TARGET" label) which is chosen to be placed at the bottom-right corner of the laboratory. The target point is chosen to be placed there, so that the robot is forced to negotiate with various obstacles such as desks, different cabinets and various machines, as well as performing a 90 degrees turn and passing through a narrow corridor right before the target location. Moreover, while moving the mobile robot must be able to avoid the static objects which are present on the way inside the laboratory in a safe way. The blue line instead represents the trajectory that the mobile robot has followed. It can be claimed that the robot has performed successfully the assigned point-to-point motion task, avoiding the static obstacles. Moreover, it can also be seen that, during its motion, the robot has such a behaviour to keep a safety distance from the obstacles.

Figure 16. Trajectory of the mobile robot for the point-to-point motion in the absence of dynamic obstacles.

Contrary to the previously performed validation test, for the motion through way-points we elected to allow the mobile robot to move along a longer path, therefore, the target point has been placed on the other side of the laboratory. This can be seen in Figure 17, in which, again, the starting and target points are indicated with the “START" green label and “TARGET" red label, respectively. The assigned way-points, instead, are indicated withe the black “WAYPOINTS" label, while the trajectory the robot has performed is still shown as the blue curve. Regarding the same figure, it can be noted that the robot has been assigned to a path through a very narrow corridor for most of its motion. However, similarly to the previous test, it can be seen that the robot has performed successfully the motion through

 doi:10.20944/preprints202212.0233.v1

way-points, avoiding the static obstacles that are present in the environment. Additionally, it has been noted that its able to move in narrow corridors successfully.

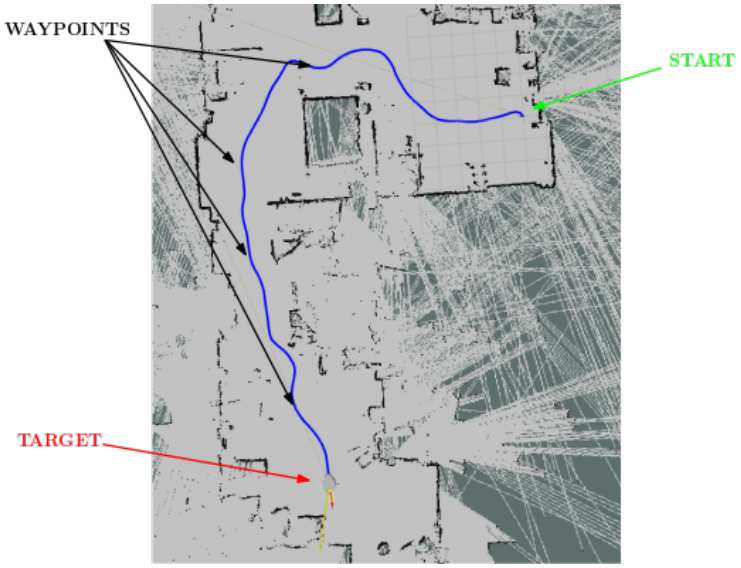


Figure 17. Trajectory of the mobile robot for the path following through given way-points in the absence of dynamic obstacles.

5.2. Case 2. With pedestrians

The location for this final experimental test is still the the laboratory elected as indoor testing environment. Conversely however, in this case other than performing a point-to-point motion and avoiding static obstacles, the robot is required to avoid moving pedestrians that cross its path and move about in its close proximity. With reference to Figure 18, the starting and target points assigned to the robot are the same ones used for the point-to-point motion test case described in the previous paragraph, and indicated in green and red respectively. The trajectory followed by the mobile robot during this point-to-point motion is again indicated with a blue curve.

Figure 18. Trajectory of the robot for the point to point motion in presence of moving pedestrians as dynamic obstacles.

 doi:10.20944/preprints202212.0233.v1

The first detail that it is possible to notice in Figure 18 is that the curve representing the trajectory followed by the robot has tangles in some regions. This is because in those regions a moving pedestrian was in close proximity of the mobile robot, thus the latter had to perform some evasive maneuver in order to avoid the former. It can be noted that despite the complexity introduced by moving pedestrians in the experimental environment, the mobile robot has been still able to avoid both static obstacles and pedestrians while at the same time move up to the assigned target completely safe without any collisions. The findings of this experimental testing suggest also that a simple bi-dimensional LiDAR scanner is able to detect the legs of the pedestrians, and it is suitable for crowd navigation applications.

Finally, Figure 19 reports some video frames that has been extracted from the robot camera during its point-to-point motion. In some of the frames the moving pedestrians can be seen crossing the mobile robot path and being in its close proximity.

Figure 19. Extracted video frames from the robot camera during its point to point motion in presence of moving pedestrians.

6. Conclusions

In this work we explore the development of a controller for a mobile robot that is capable to move towards target locations, while at the same time to avoid both static and dynamic obstacles e.g. pedestrians. The simulated environment in all its single constituting parts has been presented. The core controller of the robot is based on the use of a NN which has been trained through two different RL algorithms: DQL and A3C. Comparisons between trained NNs has been carried out, and findings show that the A3C-trained NN performs better and generalize the problem more effectively with respect to the DQL-trained NN. Both algorithms are able to reach convergence during the training process, however, we show that the DQL-trained NN learns a sub optimal strategy.

We leveraged the developed pipeline in order to run the NNs training in a highly parallelized environment, thus cutting down the training time by an extreme amount for this kind of complex problem; incidentally, on a single machine, i.e. without parallelization, the training time would render the problem very hard to solve.

Additionally, it has been seen that a complex problem such as the navigation in a very dynamic environment with moving pedestrians, can be effectively tackled by using NN trained with reinforcement learning techniques. In fact, it has been seen that in the simulation environment, using both trained NNs, the robot is able to successfully complete the assigned task with a remarkable success ratio for different environment pedestrian

densities. Additionally, the approach has been also validated experimentally, and in this process it has been seen that the mobile robot is capable of performing point-to-point motion and moving through way-points in both absence and presence of dynamic obstacles as well as static obstacles; the robot show very good obstacle avoidance performance. Moreover, it has been also seen that a simple perception system such a bi-dimensional LiDAR scanner can be effectively used to detect pedestrian legs and thus provide meaningful data to the input of the NN. Henceforth, the use of raycasting algorithms in the simulated environments represents well the LiDAR functioning principle.

As for the experimental validation of the controller trained with the A3C, it must be pointed out that the environment in which the robot and the controller has been tested was not very “robot friendly"; it presented many challenges for both the robot and its perception system. Indeed, the experimental environment is not highly structured and presents many obstacles which the robot with its perception system struggles to detect. More precisely, there are obstacles which are too high or too low to be detected, or too narrow, e.g. chair and desk legs. However, despite these drawbacks, the mobile robot still was capable to safely and effectively perform the assigned task.

Future work foresees an extension to not only to consider bi-dimensional LiDAR sensors, but a robot equipping also a vision system. This will allow to perform sensor fusion with the LiDAR readings, gaining more information about obstacles which are hardly manageable with just with the bi-dimensional scanners. Moreover, the scenarios pool, with the simulated environment, will be further extended in order to include more critical scenarios such as mazes and narrow corridors. Moreover, the possibility will be investigated to apply the findings of this research in training a controller for a more complex mobile robot such as those with four steerable wheels in which each steer joint is subjected to joint limits constraints, as for instance the Archimede rover presented by Caruso et al. in [43].

Author Contributions: “Conceptualization, M.C., E.R., L.B. and S.S.; methodology, M.C., E.R., S.A.R.; software, M.C. E.R., F.J.C.V and S.A.R.; validation, M.C.; resources, L.B. and S.S.; data curation M.C., E.R.; writing—original draft preparation, M.C., E.R., F.J.C.V., S.A.R.; writing—review and editing, M.C., E.R., F.J.C.V., S.A.R., L.B. and S.S.; visualization, M.C., E.R.; supervision, L.B., S.S.; project administration, L.B., S.S.; funding acquisition, L.B., S.S. All authors have read and agreed to the published version of the manuscript.”

Funding: This work has been partially supported by the PRIN project “SEDUCE” n. 2017TWRCNB, by the internal funding program “Microgrants 2020” of the University of Trieste and from the "Regione Autonoma Friuli Venezia Giulia" through the Fondo Sociale Europeo (FSE) funding for "Dottorati di Ricerca 35° ciclo/P.S.89bis19 - budget ricerca 10%"

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Informed consent was obtained from all subjects involved in the study.

Data Availability Statement: Data and videoas, sample and codes are publicly available at the GitHub repositories https://github.com/EnricoReg/robot-sf, https://github.com/EnricoReg/asynch-rl and https://github.com/matteocaruso1993/crowd_nav_experimental

Conflicts of Interest: The authors declare no conflict of interest

Abbreviations

The following abbreviations are used in this manuscript:

RL	Reinforcement Learning
NN	Neural Network
DQL	Deep Q-Learning
A3C	Asynchronous Advantage Actor Critic
WMR	Wheeled Mobile Robot

References

1. Hercik, R.; Byrtus, R.; Jaros, R.; Koziorek, J. Implementation of Autonomous Mobile Robot in SmartFactory. *Applied Sciences* **2022**, *12*. https://doi.org/10.3390/app12178912.
2. Oyekanlu, E.A.; Smith, A.C.; Thomas, W.P.; Mulroy, G.; Hitesh, D.; Ramsey, M.; Kuhn, D.J.; Mcghinnis, J.D.; Buonavita, S.C.; Looper, N.A.; et al. A Review of Recent Advances in Automated Guided Vehicle Technologies: Integration Challenges and Research Areas for 5G-Based Smart Manufacturing Applications. *IEEE Access* **2020**, *8*, 202312–202353. https://doi.org/10.1109/ACCESS.2020.3035729.
3. Bøgh, S.; Gjøl Jensen, P.; Kristjansen, M.; Guldstrand Larsen, K.; Nyman, U. Distributed Fleet Management in Noisy Environments via Model-Predictive Control. *Proceedings of the International Conference on Automated Planning and Scheduling* **2022**, *32*, 565–573. https://doi.org/10.1609/icaps.v32i1.19843.
4. Caruso, M.; Gallina, P.; Seriani, S. On the modelling of tethered mobile robots as redundant manipulators. *Robotics* **2021**, *10*.
5. Lemasurier, G.; Bejerano, G.; Albanese, V.; Parrillo, J.; Yanco, H.A.; Amerson, N.; Hetrick, R.; Phillips, E. Methods for Expressing Robot Intent for Human–Robot Collaboration in Shared Workspaces. *J. Hum.-Robot Interact.* **2021**, *10*. https://doi.org/10.1145/3472223.
6. Trautman, P.; Krause, A. Unfreezing the robot: Navigation in dense, interacting crowds. In Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2010, pp. 797–803.
7. Trautman, P.; Ma, J.; Murray, R.M.; Krause, A. Robot navigation in dense human crowds: the case for cooperation. In Proceedings of the 2013 IEEE International Conference on Robotics and Automation, 2013, pp. 2153–2160. https://doi.org/10.1109/ICRA.2013.6630866.
8. Abdulov, A.; Abramenkov, A. Collision Avoidance by Communication for Autonomous Mobile Robots in Crowd. In Proceedings of the 2018 Eleventh International Conference "Management of large-scale system development" (MLSD, 2018, pp. 1–4. https://doi.org/10.1109/MLSD.2018.8551804.
9. Seriani, S.; Marcini, L.; Caruso, M.; Gallina, P.; Medvet, E. Crowded Environment Navigation with NEAT: Impact of Perception Resolution on Controller Optimization. *Journal of Intelligent & Robotic Systems* **2021**, *101*, 36. https://doi.org/10.1007/s10846-020-01308-8.
10. Stanley, K.O.; Miikkulainen, R. Evolving neural networks through augmenting topologies. *Evolutionary computation* **2002**, *10*, 99–127.
11. Dimitropoulos, K.; Hatzilygeroudis, I.; Chatzilygeroudis, K. A Brief Survey of Sim2Real Methods for Robot Learning. In Proceedings of the International Conference on Robotics in Alpe-Adria Danube Region. Springer, 2022, pp. 133–140.
12. Yang, S.; Li, T.; Gong, X.; Peng, B.; Hu, J. A review on crowd simulation and modeling. *Graphical Models* **2020**, *111*, 101081. https://doi.org/https://doi.org/10.1016/j.gmod.2020.101081.
13. Yang, S.; Li, T.; Gong, X.; Peng, B.; Hu, J. A review on crowd simulation and modeling. *Graphical Models* **2020**, *111*, 101081. https://doi.org/https://doi.org/10.1016/j.gmod.2020.101081.
14. Fraichard, T.; Levesy, V. From Crowd Simulation to Robot Navigation in Crowds. *IEEE Robotics and Automation Letters* **2020**, *5*, 729–735. https://doi.org/10.1109/LRA.2020.2965032.
15. Helbing, D.; Farkas, I.; Vicsek, T. Simulating dynamical features of escape panic. *Nature* **2000**, *407*, 487–490. https://doi.org/10.1038/35035023.
16. Helbing, D.; Johansson, A., Pedestrian, Crowd and Evacuation Dynamics. In *Encyclopedia of Complexity and Systems Science*; Meyers, R.A., Ed.; Springer New York: New York, NY, 2009; pp. 6476–6495. https://doi.org/10.1007/978-0-387-30440-3_382.
17. Karamouzas, I.; Skinner, B.; Guy, S.J. Universal Power Law Governing Pedestrian Interactions. *Phys. Rev. Lett.* **2014**, *113*, 238701. https://doi.org/10.1103/PhysRevLett.113.238701.
18. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *nature* **2015**, *518*, 529–533.
19. Mirhoseini, A.; Goldie, A.; Yazgan, M.; Jiang, J.; Songhori, E.; Wang, S.; Lee, Y.J.; Johnson, E.; Pathak, O.; Bae, S.; et al. Chip placement with deep reinforcement learning. *arXiv preprint arXiv:2004.10746* **2020**.
20. Bellemare, M.G.; Candido, S.; Castro, P.S.; Gong, J.; Machado, M.C.; Moitra, S.; Ponda, S.S.; Wang, Z. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature* **2020**, *588*, 77–82.
21. Degrave, J.; Felici, F.; Buchli, J.; Neunert, M.; Tracey, B.; Carpanese, F.; Ewalds, T.; Hafner, R.; Abdolmaleki, A.; de Las Casas, D.; et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature* **2022**, *602*, 414–419.
22. Ibarz, J.; Tan, J.; Finn, C.; Kalakrishnan, M.; Pastor, P.; Levine, S. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research* **2021**, *40*, 698–721.
23. Tai, L.; Paolo, G.; Liu, M. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2017, pp. 31–36.
24. Katyal, K.; Gao, Y.; Markowitz, J.; Wang, I.J.; Huang, C.M. Group-Aware Robot Navigation in Crowded Environments, 2020, [arXiv:cs.RO/2012.12291].
25. Chen, C.; Liu, Y.; Kreiss, S.; Alahi, A. Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning. In Proceedings of the 2019 International Conference on Robotics and Automation (ICRA). IEEE, 2019, pp. 6015–6022.

26. Regier, P.; Shareef, I.; Bennewitz, M. Improving Navigation with the Social Force Model by Learning a Neural Network Controller in Pedestrian Crowds. In Proceedings of the 2019 European Conference on Mobile Robots (ECMR), 2019, pp. 1–6. https://doi.org/10.1109/ECMR.2019.8870923.
27. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms, 2017. https://doi.org/10.48550/ARXIV.1707.06347.
28. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* **2013**.
29. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In Proceedings of the International conference on machine learning. PMLR, 2016, pp. 1928–1937.
30. Quigley, M. ROS: an open-source Robot Operating System. In Proceedings of the IEEE International Conference on Robotics and Automation, 2009.
31. Moussaïd, M.; Perozo, N.; Garnier, S.; Helbing, D.; Theraulaz, G. The Walking Behaviour of Pedestrian Social Groups and Its Impact on Crowd Dynamics. *PLOS ONE* **2010**, *5*, 1–7. https://doi.org/10.1371/journal.pone.0010047.
32. Helbing, D.; Molnár, P. Social force model for pedestrian dynamics. *Phys. Rev. E* **1995**, *51*, 4282–4286. https://doi.org/10.1103/PhysRevE.51.4282.
33. Johansson, A.; Helbing, D.; Shukla, P. Specification of the social force pedestrian model by evolutionary adjustment to video tracking data, 2008, [arXiv:physics.soc-ph/0810.4587].
34. Moussaïd, M.; Helbing, D.; Garnier, S.; Johansson, A.; Combe, M.; Theraulaz, G. Experimental study of the behavioural mechanisms underlying self-organization in human crowds. *Proceedings. Biological sciences / The Royal Society* **2009**, *276*, 2755–62. https://doi.org/10.1098/rspb.2009.0405.
35. Sutton, R.S.; Barto, A.G. *Reinforcement learning: An introduction*; MIT press, 2018.
36. Tsitsiklis, J.N. Asynchronous stochastic approximation and Q-learning. *Machine learning* **1994**, *16*, 185–202.
37. Tokic, M.; Palm, G. Value-difference based exploration: adaptive control between epsilon-greedy and softmax. In Proceedings of the Annual conference on artificial intelligence. Springer, 2011, pp. 335–346.
38. Yoo, A.B.; Jette, M.A.; Grondona, M. Slurm: Simple linux utility for resource management. In Proceedings of the Workshop on job scheduling strategies for parallel processing. Springer, 2003, pp. 44–60.
39. Gabriel, E.; Fagg, G.E.; Bosilca, G.; Angskun, T.; Dongarra, J.J.; Squyres, J.M.; Sahay, V.; Kambadur, P.; Barrett, B.; Lumsdaine, A.; et al. Open MPI: Goals, concept, and design of a next generation MPI implementation. In Proceedings of the European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting. Springer, 2004, pp. 97–104.
40. Moritz, P.; Nishihara, R.; Wang, S.; Tumanov, A.; Liaw, R.; Liang, E.; Elibol, M.; Yang, Z.; Paul, W.; Jordan, M.I.; et al. Ray: A distributed framework for emerging {AI} applications. In Proceedings of the 13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18), 2018, pp. 561–577.
41. Hwang, K.; Jotwani, N. *Advanced computer architecture, 3e*; McGraw-Hill Education, 2016.
42. Koenig, N.; Howard, A. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566), 2004, Vol. 3, pp. 2149–2154 vol.3. https://doi.org/10.1109/IROS.2004.1389727.
43. Caruso, M.; Bregant, L.; Gallina, P.; Seriani, S. Design and multi-body dynamic analysis of the Archimede space exploration rover. *Acta Astronautica* **2022**, *194*, 229–241. https://doi.org/https://doi.org/10.1016/j.actaastro.2022.02.003.