Journal of Intelligent Manufacturing
https://doi.org/10.1007/s10845-023-02197-y

Predictive reinforcement learning: map-less navigation method for mobile robot

Dmitrii Dobriborsci[1] · Roman Zashchitin[1] · Mikhail Kakanov[1] · Wolfgang Aumer[1] · Pavel Osinenko[2]

Received: 24 March 2023 / Accepted: 10 August 2023

Abstract
The application of reinforcement learning in mobile robotics faces the challenges of real-world physical environments, in contrast to playground setups like video games. In a mobile robot motion control, it is not always possible to perform episodes of pre-training in large amounts due to time, resource limitations or other concerns. Control methods that rely on a prior explicit map may be impractical or even impossible to use for new dynamic environments. In this paper, we present a method of local navigation approach for driving a robot to a desired position without relying on an explicit map of the environment. Only the laser scan measurements were used to determine the obstacles. We focus in this work on online methods of reinforcement learning which do not require running the robot in full episodes until success or failure. However, the price for such an online capability is that some model knowledge about the environment has to be utilized. Here, we propose an algorithm called *stacked Q-learning*, which unifies aspects of standard reinforcement learning techniques with model-based predictive agents. We compare this algorithm to a classical *model predictive controller*. The comparison focuses on the accumulated cost of parking the robot avoiding obstacles. The results look promising as the stacked Q-learning beat its counterpart, model predictive control, yet being of the same computational complexity. The suggested agent design of stacked Q-learning can thus be taken as a foundation for a class of predictive reinforcement learning methods.

Keywords Reinforcement learning · Mobile robotics · Predictive control · Mapless navigation

Introduction

The mathematical apparatus underlying reinforcement learning (RL) allows autonomous learning of control policies. One of the most distinguishable features of RL is the process of learning by trial-and-error (Sutton & Barto, 2018). However, despite recent developments and advances in RL (Mnih et al., 2015; Silver et al., 2016; Yahya et al., 2017), the main limitation of current RL approaches is associated with data usage effectiveness. The required number of interactions with the environment is quite large, which can be impractical in real-world applications. For example, many RL approaches, even for low-order systems and relatively simple dynamics, require thousands of training iterations. For this reason, in problems of control of technical systems (in particular, robotics), the use of such approaches becomes almost impossible, let alone more complex cases.

Over the past decade, RL methods have made remarkable progress in several tasks and competitions such as Go, chess, shogi, StarCraft II, and Rubik's Cube (Akkaya et al., 2019; Silver et al., 2016, 2018; Vinyals et al., 2019). While some of these tasks are now easily beatable by RL agents, recently, RL has started moving towards technically-sound settings like robot manipulation (Kumar et al., 2016; Borno et al., 2013;

✉ Dmitrii Dobriborsci
dmitrii.dobriborsci@th-deg.de

Roman Zashchitin
roman.zashchitin@th-deg.de

Mikhail Kakanov
mikhail.kakanov@gmail.com

Wolfgang Aumer
wolfgang.aumer@th-deg.de

Pavel Osinenko
p.osinenko@skoltech.ru

[1] Technology Campus Cham, Deggendorf Institute of Technology, Badstrasse 21, 93413 Cham, Bavaria, Germany

[2] Center for Digital Engineering, Skolkovo Institute of Technology, Bolshoy Boulevard 30, bld. 1, Moscow, Russia 121205

Published online: 31 August 2023

Table 1 RL: model-based vs. model-free

	Model-based	Model-free
Descr	An agent uses a system model	An agent does not require a system model
Pros	Stability guarantees could be obtained (García, 2015)	Does not require to know a system model
	Recognised by industry (Polydoros & Nalpantidis, 2017)	
Cons	An accurate model of the system should be known	Has limited application in industry
	The prediction error grows with the horizon (Xiao et al., 2019)	Data collection is expensive (Sun et al., 2019)
		Guaranteeing safety and performance are complicated

Fig. 1 Robot with obstacles in the test environment

Tassa et al., 2012) and mobile robot navigation (Surmann et al., 2020; Dobrevski & Skočaj, 2021; Guldenring et al., 2020).

Here, we can note the similarity of the proposed approach to *model predictive control* (MPC), which has already been well studied and widely developed and implemented. The use of predictive models in RL is already gaining popularity in real-world applications. Thus, RL can be divided into two large classes: model-free (Degris et al., 2012; Gullapalli et al., 1994; Davari e tal., 2017) and model-based (Szita and Szepesvári, 2010; Huang et al., 2020; Lambert et al., 2020).

Each approach has its own merits and demerits (see Table 1).

In this paper, we study the effectiveness of online RL-based autonomous navigation of a mobile robot (Fig. 1).

In this regard, the embedding of a priori known information about the system model of the control object looks promising. However, it is often difficult to identify the dynamic model in itself and can lead to limited steady-state errors due to inaccuracies. Nevertheless, a well-identified model will allow the agent to learn online much faster, thus minimizing or eliminating trial runs.

Related works The goal of the motion control is to drive the robot to a desired position avoiding obstacles. This problem can, of course, be solved using the classical methods like simultaneous localization and mapping method (SLAM). It solves this problem by prior environment mapping using a laser rangefinder. However, this approach is time-consuming to build and update the environment map and highly depends on the accuracy of the laser rangefinder used to map and construct the local cost-map. Despite the abundance of approaches to the navigation problem it is still a challenge to rapidly generate appropriate navigation behaviours for mobile robots without an obstacle map and based on pure range sensor data or depth cameras.

RL-based map-less navigation There are several methods for achieving map-less navigation, and this section focuses on laser range finder-based approaches, which are closely related to the methods under study. While there are also vision-based approaches (Mirowski et al., 2018; Kulhánek et al., 2021; Zhu et al., 2017), they are not within the scope of this work.

The motion planners considered in this section are map-free and unify the global planner and local planner. This approach eliminates the need to create and maintain a prior geometric map of the environment, which can have a significant impact on the final outcome of the motion planning. Additionally, compared to supervised learning-based map-less motion planning methods, RL-based methods can learn and evolve directly from the interaction data between robots and external environments, avoiding the need for the construction of complex labeled expert data-sets.

One example of such an RL-based approach is the exploration of deep RL methods for learning navigation problems for wheeled mobile robots using intrinsic reward, as investigated in Zhelo et al. (2018). The approach was tested in different scenarios without an explicit occupancy map of the environment. Another example is the deep RL-based navigation strategy proposed in Shi et al. (2020), which uses an intrinsic curiosity module as a more general end-to-end motion planner based on the A3C framework with sparse Lidar data input. The proposed planner was successfully deployed in a realistic mixed scene.

While these approaches have their advantages, they require a pre-training procedure, which may not always be feasible or practical. To address this issue, a hybridization of RL methods with the optimal control theory's well-established apparatus, such as MPC, is studied in this paper.

This approach combines the benefits of RL-based learning with the explicit computations of MPC, allowing for efficient and effective map-less navigation without the need for pre-training.

Fusion of RL and MPC Research into the hybridization of reinforcement learning methods is attracting more and more attention from scientists worldwide. There are attempts to combine functional properties of RL, such as accumulating prior knowledge and the computational features of MPC (see e.g. Berkenkamp et al. (2017), Song and Scaramuzza (2007), Napat et al. (2020), Hoeller et al. (2020)).

There are also some advances not only in theoretical research. For example, applications of new online methods in the context of complex engineering problems are studied in Lenz et al. (2015), Drews et al. (2017). More details on the prospects for hybridizing these two general approaches and their benefits are outlined in several papers, e.g., here Zanon and Gros (2021), Bhardwaj et al. (2020), Primbs et al. (1999).

The primary focus of this work is to investigate the performance of the recommended agents in experiments involving a mobile robot. Previous research has already addressed the issue of integrating safety and stability constraints, as demonstrated in publications such as Osinenko et al. (2020), Beckenbach et al. (2020, 2018), Osinenko et al. (2023).

Contribution This paper presents the results of experimental studies of previously proposed model-based RL algorithms for the mobile robot navigation problem with obstacle avoidance. The peculiarity of the approach is that there is no need to construct the environment map in advance, which allows applying these approaches in an unknown environment. Moreover, the approach under study does not require prior training of the agent, which also removes the energy-consuming pre-training and sim2real transfer problems. As a payment for such a feature, prior knowledge of the models' structure is required. The difference from our previously published works is the implementing of an obstacle recognition module based on laser rangefinder measurements. Based on the sensory data constraint equations were formulated and incorporated into the controller structure. At the same time, the computational complexity of the proposed RL-based *Stacked Q-learning* (SQL) algorithm is the same as that of the classical MPC. This is explained by the identity of the optimality property of SQL and conventional dynamic programming. Thus, SQL is a legitimate Hamiltn–Jacobi–Bellman optimal control method. Whereas in the paper Osinenko and Dobriborsci (2021) exclusively simulation studies on the effects of prediction horizon and sampling were carried out. In the paper Dobriborsci and Osinenko (2022), experimental studies without obstacle avoidance were considered. Thus, this paper presents a generalization of previously obtained results in the context of the map-less navigation problem for mobile robots. Note that the work confirms the previously obtained results, which are summarized below. All necessary code and instructions for reproducing the results are provided as an open-source framework (see Osinenko (2020)). This repository is a kind of snapshot of the *Rcognita* Osinenko (2020) framework with an additional ROS-preset [*Robot Operating System* (Quigley et al., 2009)]. This framework *Rcognita* has wide versatility and allows algorithms benchmarking for different models, whether a wheeled robot, a walking robot, or any other dynamical system.

The main results can be summarized as follows. Both methods (MPC as a baseline, and proposed Stacked RL) showed better performance at a longer prediction horizon, which was expected. While SQL beat the baseline MPC in terms of the cost. This shows potential of stacked approaches as viable predictive RL modifications.

Preliminaries and background

Kinematic model of a nonholonomic mobile robot

Our study aims to validate our predictive RL-based motion controller using a nonholonomic wheeled mobile robot (WMR) Robotis© Turtlebot3 Burger through real-world and simulation experiments.

At time t, the origin of the local coordinate frame (base) of the robot is located at the point $(x(t), y(t))$ of the world coordinate frame. The robot's linear velocity $v(t)$ along the axes determines the direction of motion, while the angular velocity $\omega(t)$ determines the rotation (refer to Fig. 2a). The state vector $\mathbf{x} \in \mathbb{R}^3$ and the control input vector $\mathbf{u} \in \mathbb{R}^2$ are defined as

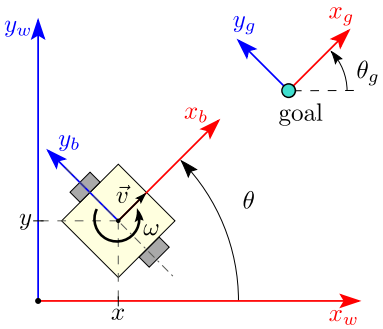
$$\mathbf{x} := \left[x(t)\ y(t)\ \theta(t)\right]^T, \mathbf{u} := \left[v(t)\ \omega(t)\right]^T. \tag{1}$$

The kinematic model of the WMR can be represented in a state-space form as

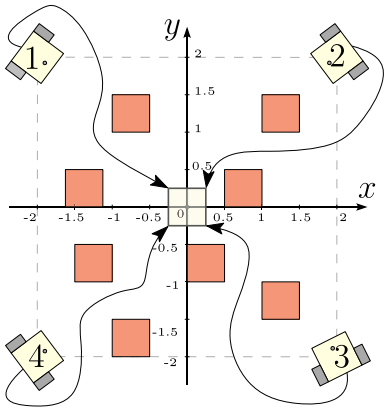
$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) \leftarrow \begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = \omega, \end{cases} \tag{2}$$

where $f(\cdot)$ denotes a nonlinear function of states and control inputs, x is the x-coordinate [m], y is the y-coordinate [m], θ is the orientation of the robot [rad], v is the velocity [m/s], ω is the angular velocity [rad/s]. Two advantages of a WMR robot are its simplicity (typically the motor is attached directly to the axle of each wheel) and high maneuverability (the robot can spin in place by rotating the wheels in opposite directions). Casters are often not appropriate for outdoor use, however.

Fig. 2 Experimental system overview



(a) Robot kinematic scheme.



(b) Graphical interpretation of the control goal.

The robot's states and control inputs may have specific constraints. For instance, if the robot operates in a bounded environment, its state vector should remain within the boundaries' coordinates defined by $\mathbf{x_{min}}$ and $\mathbf{x_{max}}$. Additionally, the control inputs in real-world scenarios are constrained by $\mathbf{u_{min}}$ and $\mathbf{u_{max}}$, which are determined by the actuators' capabilities.

The task is to design such control law, which ensures the fulfillment of the following condition

$$\lim_{t\to\infty} |e(t)| \leq \Delta, \tag{3}$$

where $\mathbf{e(t)} = \mathbf{x}(t) - \mathbf{x_{des}}(t)$ is position error, Δ is admissible accuracy.

In dynamic positioning, the main objective is to stabilize the robot in a desired pose. This work addresses this challenge using reinforcement learning, which is a particular case of an optimal control problem.

Notation For any z, denote $\{z_{i|k}\}_i^N = \{z_{1|k}, \ldots, z_{N|k}\} = \{z_k, \ldots, z_{k+N-1}\}$, if the starting index k is emphasized; otherwise, it is just $\{z_i\}^N = \{z_1, \ldots, z_N\}$. If N in the above is omitted, the sequence is considered infinite. The notation [N] will mean the set {1, 2,…, N}.

Methodology

This section presents the algorithms used in the experimental study. It begins with a general (digital) optimal control and transitions to MPC followed by the stacked Q-learning.

In general, predictive control algorithms are designed to improve the overall performance of the control system and to improve quality indicators by using the future states of the system. Evidently, higher prediction horizons lead to higher computational complexity. At the same time, too short horizon may lead to a failure to even stabilize the system. Traditionally, MPC, which is discussed in the next section, is considered as a standard predictive controller.

Model predictive control

One typical instance of an optimal control problem involves a predictive digital controller that maintains constant control actions throughout sampling intervals. The objective is to minimize a cost function J in the following manner

$$\begin{aligned} &\min_{\kappa} J\left(x_0|\kappa\right) = \min_{\kappa} \sum_{i=1^H} \gamma^{i-1} \rho\left(\hat{x}_{i|k}, \kappa(x_{i|k})\right), \\ &\text{s.t. } \mathcal{D}^+ x = f(x, u^\delta), x(0) = x_0 \\ &\quad x_{i|k} := x((k+i-1)\delta), k \in \mathbb{N} \\ &\quad u^\delta(t) \equiv u_{i|k} = \kappa(x_{i|k}), t \in [k\delta, (k+i)\delta], \\ &\quad \hat{x}_{i+1|k} = \Phi(\delta, \hat{x}_{i|k}, u_{i|k}) \end{aligned} \tag{4}$$

where x is the state, u is the action, ρ is the running cost (also called reward, utility, etc.), κ is the control policy, H is the horizon length, x_k is the state at time-step k, γ is the discounting factor, $\mathcal{D}^+$ is the a suitable differential operator, $\delta > 0$ is the digital controller sampling time, Φ describes state prediction which can be , e. g.,a numerical integrator, say, via Euler method:

$$\Phi(\delta, \hat{x}_{i|k}, u_{i|k}) = \hat{x}_{i|k} + \delta f(\hat{x}_{i|k}, u_{i|k}). \tag{5}$$

In terms of horizon, H depends on the context of the problem and can be finite ($H := N$) or infinite ($H := \infty$), whence care should be taken of prediction inaccuracy accumulation (a more suitable system description could be pure time-discrete where Φ produces future state exactly). When $H := \infty$, J is also called "cost-to-go". Based on the described variants, two main optimal control formalisms are commonly known, namely Euler–Lagrange and Hamilton–Jacobi–Bellman (Primbs et al., 1999). The Euler–Lagrange

formalism is the foundation of MPC and describes the case when H is finite. The resulting policy depends on the current state. The Hamilton–Jacobi–Bellman, in contrast, is used to describe globally optimal policies which only depend on the initial state. In turn, an infinite horizon can be interpreted as an open horizon—a situation in which the user is not sure of the exact specification of the horizon. The optimal control problem with a finite horizon can be interpreted as an approximation to the problem with an infinite horizon, whereas the latter may well appear intractable. MPC, in one of its simplest variants, is precisely (4) with a finite horizon. One of the main features of it is that it has established convergence analyses (Mayne, 2014). As for sub-optimality, increasing the horizon reduces the mismatch between the factual cost-to-go under MPC and the value function $V = \min_\kappa J^\kappa$ (the optimized cost-to-go) (Grüne & Rantzer, 2008).

In contrast to MPC, RL has the HJB formalism as the groundwork. It seeks to approximate the optimal cost-to-go (or something related like Q-function or advantage function). While starting with different formalisms, integration of predictive, "MPC-esque" elements into RL may actually be viable.

Stacked Q-learning

A basic actor-critic, value-iteration, on policy QL reads:

$$\begin{aligned} u_k &:= \arg\min_u \hat{Q}(x_k, u; \vartheta_k), \\ \vartheta_k &:= \arg\min_\vartheta \tfrac{1}{2}\big(\hat{Q}(x_k, u_k; \vartheta) \\ &\quad -\hat{Q}(x_{k-1}, u_k; \vartheta^-) - \rho(x_k, u_k)\big)^2, \end{aligned} \tag{6}$$

where ϑ is vector of the critic neural network weights to be optimized, ϑ^- is the vector of the weights from the previous time step, $\hat{Q}(\bullet, \bullet; \vartheta)$—Q-function approximation parameterized by ϑ. The latter approximation is effectively the temporal difference (TD) in the value iteration form. It may be generalized to a custom size experience replay. Let $e_k(\vartheta) = \vartheta\varphi(x_{k-1}, u_{k-1}) - \gamma\vartheta^-\varphi(x_k, u_k) - \rho(x_{k-1}, u_{k-1})$ denote the temporal difference at time step k. Then, a more general critic cost function may be formulated as

$$J_k^c(\vartheta) = \frac{1}{2}\sum_{i=k}^{k+M-1} e_i^2(\vartheta), \tag{7}$$

where M is the buffer size.

In stacked Q-learning (Osinenko et al., 2017; Beckenbach et al., 2018), single Q-function approximation is substituted with the sum of multiple Q-functions, which resulted in a finite stack of Q-functions over a finite number of time steps (see Alg. 1). One may seek to approximate not only the Q-function for the current state, but also for a sequence of future Q-functions. As a result, it is necessary to establish some basic concepts for SQL. The generic Q-function under a policy κ reads, for a state x_k:

$$Q^\kappa(x_k, u_k) = \rho(x_k, u_k) + J^\kappa(x_{k+1}). \tag{8}$$

The value function J^* and the Q-function are related as follows:

$$J^*(x_k) := \min_{u_k} Q(x_k, u_k) = \rho(x_k, u_k) + V(x_{k+1}). \tag{9}$$

A Q-function stack can be defined as Osinenko et al. (2016):

$$\bar{Q}\left(x_k, \{u_{i|k}\}_i^N\right) := \sum_{i=1}^N Q(x_{i|k}, u_{i|k}). \tag{10}$$

Under generic policies, the stack can be expressed as:

$$\begin{aligned} \bar{Q}^{\{\kappa_{i|k}\}_i^N}(x_k, \{u_{i|k}\}_i^N) &= \sum_{i=1}^N \rho\left(x_{i|j}, u_{i|k}\right) \\ &\quad + \sum_{i=1}^N \sum_{j=k+1}^\infty \rho\left(x_{i|j}, \kappa_i(x_{i|j})\right), \end{aligned} \tag{11}$$

where i is a horizon index and j is an index for starting state k update, $\{\kappa_{i|1}\}_i^N$ – stack of policies.

Compare it to a single Q-function case:

$$Q^\kappa(x_k, u_k) = \rho\left(x_k, u_k\right) + \sum_{j=k+1}^\infty \rho\left(x_j, \kappa(x_j)\right). \tag{12}$$

For a theoretical analysis of the SQL optimality, please refer to Beckenbach et al. (2018).

Finally, a practical, neural network-based SQL actor reads:

$$\begin{aligned} &\min_{\{u_{i|k}\}_i^N} J_{\text{SQL}}^a\left(x_k | \{u_{i|k}\}_i^N; \vartheta_k\right) = \sum_{i=1}^N \hat{Q}(\hat{x}_{i|k}, u_{i|k}; \vartheta_k), \\ &\text{s.t.} \quad \hat{x}_{i+1|k} = \Phi(s\delta, \hat{x}_{i|k}, u_{i|k}), \end{aligned} \tag{13}$$

In the final part of this section let us summarize the main differences between the algorithms. *Stacked Q-learning (SQL)* is a modification of MPC method by hybridizing it with the core RL concept—Q-learning. The main idea is in evaluating the finite part of the infinite time horizon by stacking Q-function approximations instead of stage cost. This can be characterized as generalized predictive controller. This option provides an opportunity to evaluate the influence of the current action to not only N future steps but to all successive steps, due to the Q-function properties.

For the reader's convenience, these differences are summarized in the Table 2. The following sections describes the obstacle recognition module and experimental part of this work.

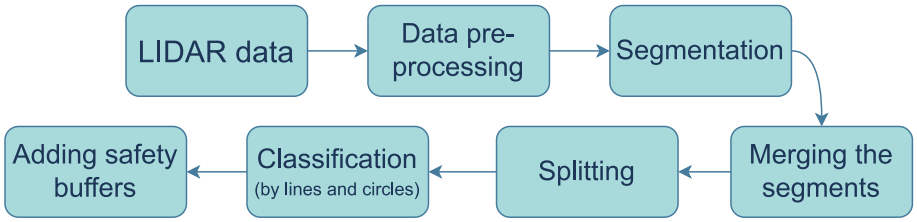


Fig. 3 The overall scheme of the algorithm

Table 2 Controller algorithms comparison

Controller	MPC	SQL
System dynamics	$x_{k+1} = f(x_k, u_k)$	
Optimal problem	$\min_{a_k} \sum_{i=k+1}^{k+N} \gamma^{i-1} r(x_i, u_i)$	$\min_{a_k} \sum_{i=k+1}^{k+N} Q(x_i, u_i)$

Algorithm 1 A practical implementation of stacked Q-learning

Input: System state x
while $t < \tau$ **do**
Critic update: $\vartheta_k^* := \arg\min_{\vartheta_k} J^c = \frac{1}{2} \sum_{k=1}^{N_c} e_k^2(\vartheta_k)$ See eq. (7)
Actor update: $\{u_{i|k}^*\}_i^N = \arg\min_{\{u_{i|k}\}_i^N} J_{SQL}^a \left(x_k | \{u_{i|k}\}_i^N; \vartheta_k^*\right) := \sum_{i=1}^{N} \hat{Q}(\hat{x}_{i|k}, u_{i|k}; \vartheta_k^*)$
Apply $u_{1|k}^*$ to the system
end while

Obstacle recognition

The main purpose of this section is to implement safe obstacle avoidance mechanism into the *Rcognita* framework. Therefore, the algorithms of obstacle detection and avoidance should be easily integrated into the functional sequence composing pipeline and can be used out-of-box (Fig. 3).

The input data for the algorithm is the laser scan data from robot's LIDAR sensor received with some frequency. This data contains information about the surrounding of the robot. Then it undergoes the transformation consisting of the following steps: preprocessing, segmentation, merging, splitting and classification. As a result of this procedure, the set of circular and rectangular obstacles is obtained. This set is further used for constructing functions having a special form so that they could be passed to the conditional optimization problem as constraints imposed on the state space. The optimization problem consists in finding the minimum of the target function (cost function) on the available part of space in order to define the next action that would lead the robot closer to the goal point whereas saving it from going into the restricted area. The techniques to apply these constraints to the state space can be various and will be discussed later.

Data processing

At this step, the stages of sensor readings processing are performed, starting with obtaining data from LIDAR sensor. These processes are performed according to the algorithms taken from Dong et al. (2021) and Przybyła (2017), that have been combined here, slightly modified and with hyperparameters adjusted.

Lidar scans the entire space around robot with a 360° view, so the date comes in the form of a set of N_R subsequent geometric points depicted in polar coordinates and successively equidistant in angular distance from each other. This set is ordered so that the position of any point is defined by the angle counted from the direction of the robot, equal to the serial number, multiplied by a constant unit angle, which is the angular distance between two adjacent points.

In order to describe the process of obstacle recognition, let us introduce the following definition:

Definition 1 *An obstacle* is a set of n points in some coordinate system that represents an object located in the environment.

In this work, two types of obstacles will be considered:

1. linear obstacle—an object with a flat surface facing the robot. It is characterized by two corner points.
2. circular obstacle—an object with a curved surface facing the robot. It is characterized by a radius and a center of circle approximating it.

A set of obstacles recognized in this work is defined as

$$\mathbb{O} \triangleq \mathbb{L} \cup \mathbb{C},$$

where

- $\mathbb{L}$ is a set of linear obstacles.
- $\mathbb{C}$ is a set of circular obstacles.

As it was discussed above, this simple classification is more multipurpose than using the convex hull of the point set, which provides more freedom in the choice of integration mechanism of obstacle avoidance method into RL algorithms.

The first step of the environment map construction is data pre-processing and segmentation.

Here, the set of points becomes more compact and filtered from extra data points. Lidar scanner has a bounded range of distances therefore in case of obstacles at some angular distance being too far the corresponding point may be assigned to infinity. Therefore, these points with incorrect values have to be replaced with some quite large finite value while maintaining the order of other points from lidar scanner.

Then, the set of points should be segmented into separate blocks having strong connection acquired from practical meaning. Two categories of segmentation process should be introduced which is a complete segmentation and a partial segmentation. An obstacle constructed from only one segment presents the process of complete segmentation. If an obstacle is combined from several segments it is considered to be partial segmentation (for example, wheels of the automobile). This option is more extensive so it will be applied in the presented algorithm.

In order to describe a segmentation process, a distance between two consecutive points i and $i+1$ expressed in polar coordinates should be introduced:

$$d(x_i, x_{i+1}) = \sqrt{x_i^2 + x_{i+1}^2 - 2x_i \cdot x_{i+1} \cdot \cos \Delta\phi}, \tag{14}$$

where

- x_i and x_{i+1} present the ranges of points i and $i+1$, consequently,
- $\Delta\phi$ is the default angle between two measurements determined by the equipment characteristics (equal to 1 deg in this work)

Let us introduce W as the linear size of the robot used in the experiments ($W = 0.178$ m). Then, the factor k representing dynamic amplification is defined in the following form:

$$\begin{cases} k = \frac{W \cdot R_i \cdot R_{i+1}}{100(R_i + R_{i+1})}, \text{ if } \frac{R_i + R_{i+1}}{2} > T_d; \\ k = 0.15, \text{ if } 0 < \frac{R_i + R_{i+1}}{2} \leq T_d; \end{cases}$$

where T_d (set to 10 in the work) is a threshold of the measured distance.

In case of d being bigger than $k \cdot W$, the point-cloud instance is segmented into two smaller ones. The meaning of the factor k is to consider the size of the robot when segmenting points to set the threshold for the distance between individual segments. For instance, there is no necessity in separating point cloud into two segments if the distance between them is too narrow for robot to pass through. Therefore, it is possible to simplify the map without distortion of the results. The following principle defines the process of unifying some segments into one after the segmentation is complete. Since the robot in unable to pass through the gap of size L: $L < W$, the merging of segments divided by this gap would improve the algorithm and simplify the map. In order to perform this process, these segments are artificially unified by adding a compensating points between them (let us name the segment corner points as p and q). Then the polar coordinates for the range and angle for i-th compensated point can be defined as follows:

$$r_i := r_p + i \cdot \frac{r_q - r_p}{N} \tag{15}$$

$$\theta_i := \theta_p + i \cdot \frac{\theta_q - \theta_p}{N} \tag{16}$$

where

- N—required number of points is defined by using the average distance d_{mean} of the consequent points in both segments: $N = \frac{L}{d_{mean}}$
- r_p, θ_p, r_q, θ_q—ranges and angles of two corner points p and q between which the artificial points are placed, respectively

The process of segments splitting, followed by merging, is performed according to the Algorithm 2.

After the described procedures a set of point clouds describing obstacles is obtained the classification of obstacles is performed. The types of obstacles utilized in this work were described above. The choice of two simplest shapes—line and circle—is stipulated by several factors. The first one is resource economy accompanied by computational complexity reduction. The shape of obstacles affect conditional optimization complexity significantly. Another one is maintaining the size of the available area since complicated shapes would increase the restricted area and narrow the admissible action space.

The process of obstacles classification is described in Algorithm 2. If

$$N_{mod} = \lfloor 0.8 \cdot R_{avg} N \rfloor > T_N = 10$$

the line segment or segments is constructed and added to the set $\mathbb{L}$. If $N_{mod} < T_N$, which means a relatively small sized point cloud, it is considered to be a circular obstacle. To obtain the radius of the circle extracted from a laser scan point cloud block, the center of the circle as the midpoint of line segment connecting the first and the last points of the block is obtained and the distance between the center and the farthest point in the block is defined as radius. The formed circle-type obstacle is appended to the set $\mathbb{O}$. Considering that robot equipment is unable to detect obstacles located too close to the sensors and the time required to change the trajectory, accompanied by the waiting time for odometry and

lidar readings a safe buffer was being added to every obstacle by increasing the radius of circular objects and sizes of linear objects in both dimensions. Therefore, the circular obstacles expanded in size whereas linear obstacles transformed into rectangles.

In Fig. 4 the realization of all the described operations can be observed on the example. In this example, two linear (rectangular) and two circular obstacles are located in the environment in which robot operates. In the next section, we continue with constraint functions design.

Considering that robot equipment is unable to detect obstacles located too close to the sensors and the time required to change the trajectory, accompanied by the waiting time for odometry and lidar readings a safe buffer was being added to every obstacle by increasing the radius of circular objects and sizes of linear objects in both dimensions. Therefore, the circular obstacles expanded in size whereas linear obstacles transformed into rectangles.

Algorithm 2 Segments splitting into subsets

Require: N—the number of points in a block $B \triangleq \{p_i\}, i \in [1, N]$;
D_m—the maximum value of the distances between each point and the line fit by two endpoints, S - the distance between two endpoints;
d_{split}—the threshold for splitting (a hyperparameter, set customly);
d_p—the distance proportion (a hyperparameter, set customly);
R_{avg}—the average range of point ranges in a laser scan point cloud block;
V_{result}, V_c—the vectors storing a laser scan point cloud block for which $\lfloor 0.8 \cdot R_{avg} N \rfloor < T_n$ and $\lfloor 0.8 \cdot R_{avg} N \rfloor > T_n$, correspondingly, where T_n - a threshold;
N_c - the size of V_c, N_{c_0}—the size of $V_c[0]$;
Place B into V_c
while $N_c \neq 0$ **do**
  if $0.8 \cdot R_{avg} N_{c0} > T_n$ **then**
    Calculate D_m of $V_c[0]$, Obtain p_k and R_k corresponding to D_m
    if $D_m > d_{split} + R_k \cdot d_p$ **then**
      Split $V_c[0]$ into $B_1 \triangleq \{p_1, p_2, \ldots, p_k\}$, $B_2 \triangleq \{p_k, p_{k+1}, \ldots, p_{N_{c_0}}\}$
      if $\lfloor 0.8 \cdot R_{avg_1} k \rfloor > T_n$ **then**
        Place B_1 to V_c
      else
        Place B_1 to V_{result}
      end if
      if $\lfloor 0.8 \cdot R_{avg_1} (n - k + 1) \rfloor > T_n$ **then**
        Place B_2 to V_c
      else
        Place B_2 to V_{result}
      end if
      Delete $V_c[0]$
    else
      Delete $V_c[0]$
    end if
  else
    Delete $V_c[0]$ in V_c and place $V_c[0]$ to V_{result}
  end if
end while
V_{result}

In Fig. 4 the realization of all the described operations can be observed on the example. In this example, two linear (rectangular) and two circular obstacles are located in the environment in which robot operates. In the next section, we continue with constraint functions design.

Constraint functions design

In order to perform constraint minimization of the cost function, the special form of the constraints function is required. The required form of the conditions in the optimization problem is usually as follows: the function takes non-positive values in the permitted area of state space and positive values in the restricted area with zero value on borders of obstacles.

Due to the parsing algorithm, the obstacles are perceived as a set of lines and circles. Let us consider these geometrical shapes in the plane (x, y).

If the coordinates of the characteristic points of all the obstacles are given it is possible to construct a general constraints function $g(x)$ which takes negative values in the permitted area, and, vice versa, takes non-negative values in the area containing obstacles, i.e. in the restricted part of the environment. In general we considered the convex case here since the concave parts of the figure would complicate the supposed trajectory of the robot and may increase the risk of robot's collision with obstacles.

First, a simple example is provided to show the simple idea behind the construction of the function. Suppose, we are given a square with its sides parallel to the coordinate axes, with corners in points (a, c), (b, c), (a, d), (b, d).

The permitted area can be defined as the following logical disjunction (in set theory equivalent to the union):

$$(x \leq a) \mid (x \geq b) \mid (y \leq c) \mid (y \geq d).$$

Each inequality describes an area of space, combining these areas gives all the allowed space. The disjunction means that, in order to be allowed point, it has to lie in at least one of the described areas.

In order to pass the function as the optimization constraint, it has to take real values and differ in sign in the permitted and prohibited areas. Therefore, it must be transformed into the following form (since the fulfillment of one of the inequalities will give the required value of the minimum function-less than 0):

$$\min(x - a, b - x, y - c, d - y) \leq 0$$

If there are several figures given, their common permitted area is defined as intersection of the permitted areas. This is similar to taking the maximum of the minimums, in order to

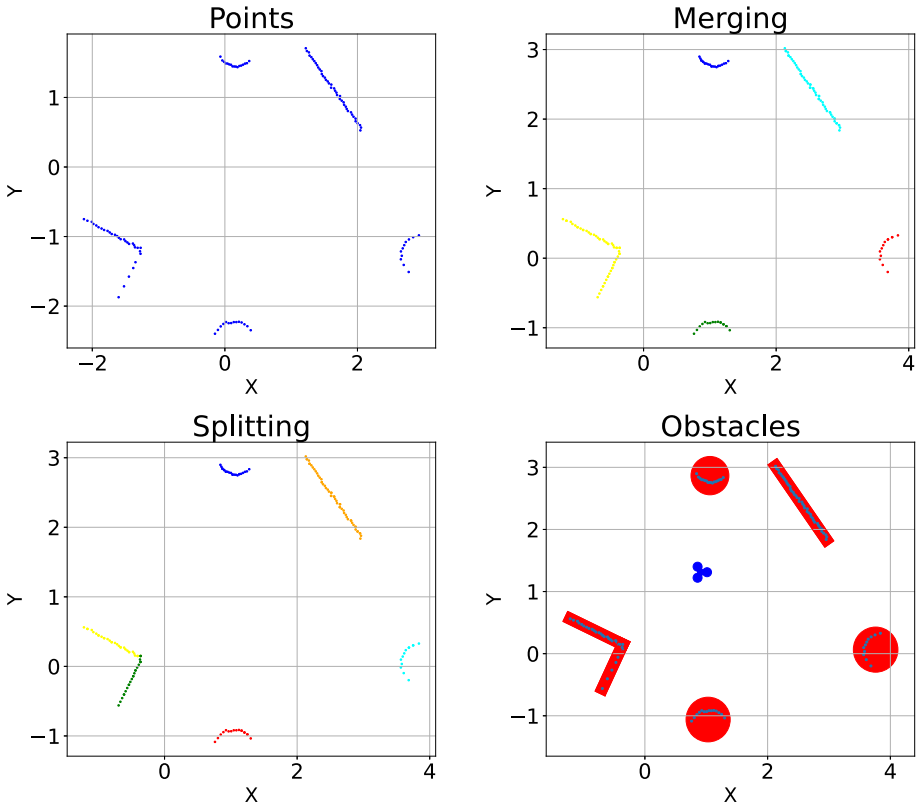


Fig. 4 Steps of lidar data processing. First, the points are merged into blocks, Output of the algorithm is the array of obstacles: lines and circles

provide meeting all conditions simultaneously:

$$\max(\min(\cdot), \min(\cdot), \ldots \min(\cdot)) \leq 0.$$

In the case of polygons with sides that in general are not parallel to the axes, the extrapolation consists of the following sequence of steps:

1. for each of m_j sides of each polygon j to find the equation for the straight line using 2 corners given (x_1, y_1) and (x_2, y_2):

$$(y_2 - y_1)x + (x_2 y_1 - x_1 y_2) - (x_2 - x_1)y = 0$$

2. to define the allowed half-plane with respect to this line (for example, by checking on which side of this line another corner of the polygon is located) and to turn the equation from the previous step into inequation (for side with number i): $g_{ij}(x, y) \leq 0$
3. to gather all inequalities into one minimum function:

$$g_j(x, y) = \min(g_{1j}(x, y), \ldots, g_{m_j j}(x, y)) \leq 0$$

4. to unite the inequalities for all polygons with taking the maximum:

$$g(x, y) = \max(g_1(x, y), \ldots, g_k(x, y)) \leq 0$$

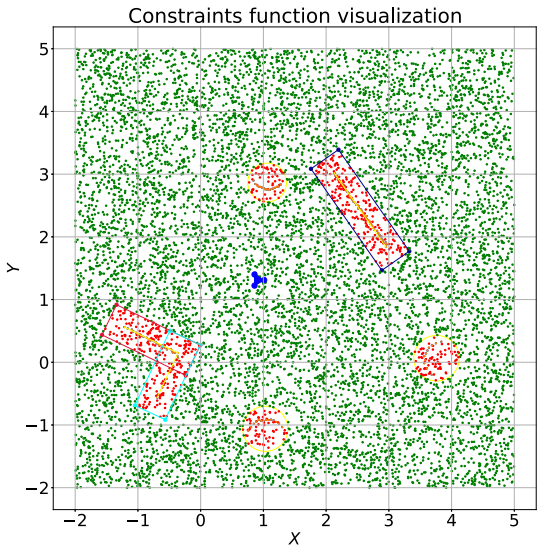


Fig. 5 The constraint function visualization—the negative values (green points) in the permitted area, the positive values (red points) in the forbidden area (Color figure online)

In this form, the function takes the positive values only in the prohibited area (inside a buffer surrounding an obstacle). It is the required form for the optimization method utilized in the algorithm. An example of this function work in practice can be observed in the Fig. 5.

Remark 1 We note that the presented graphs in the Figs. 4 and 5 are visualizations of the obstacle segmentation. The output of this module are arrays of lines and circles in the robot's environment. Thus, the given data represents a kind of "forbidden zones" incorporated into the controller structure. At the same time, the controller does not require the map to be explicitly stored and updated. This distinguishes the considered approach from SLAM.

Different ways of setting obstacle avoidance

In the function describing actor-cost minimization, the input is actions that will be taken under the given policy predicted on N (number of actor steps) time steps into the future. However, constraints can be expressed only in the state space due to the problem set. Therefore, the states are obtained from actions given the policy being followed, and these states are further evaluated for being in the forbidden region. This approach requires accuracy since there is an ambiguous connection between these spaces, and the algorithm's parameters must be selected thoroughly.

The question here is how to implement constraints into actor cost function optimization so that it would be reliable and effective. There are two main options possible:

1. integrating a penalty that is proportional to the proximity to any of the obstacles (for example, $\frac{1}{x}$, where x is the distance to an obstacle) into the cost function. This approach is risky because the cost function becomes non-continuous, and the high-cost gradients in this position are unpredictable in directions. The gradient descent starting at the acceptable cost value point may drop in several possible directions, leading to local minima or gradient trajectory oscillations without convergence. In order to contrive this crisis, the cost function may be modified to become continuous and differentiable at any point. This option requires a significant change in the target function. At the same time, the safety constraint should always be satisfied because, under any state of the system, the agent can not decide to pass through the obstacle due to the more favorable cost value.
2. to apply constraint optimization of the cost function by limiting the state and action spaces. This option does not require changes in the structure of the cost function. It is easier to implement since the classical optimization problem is complemented with imposed constraints, equations, or inequalities. This option is also more reliable than the previous one because it guarantees to avoid the dangerous area of state space. However, one drawback is considered in "Optimization process" section. Nevertheless, due to the universality of the method and its reliability, this method of obstacle avoidance is applied in this work.

Optimization process

In order to provide a safe trajectory for the robot, it is necessary to check for entry into the forbidden area not only the state at the next moment but also several subsequent states (the number of steps for which future planning takes place is equal to N, the number of predicted steps for actor-network). Therefore, a chain of states should be constructed based on the current one. It is performed according to the following rule:

$$x_{t+1} := x_t + \Delta t \cdot x \cdot f(x_t, x_t), \tag{17}$$

where $f(x_t, u_t) = A \cdot x_t$ is the transition function describing dynamics of the system, leading to the state x_{t+1} from state x_t while taking action u_t, Δt is the time step size. For the next time step the update rule becomes more expanded:

$$\begin{aligned} x_{t+2} :&= x_{t+1} + \Delta t \cdot f(u_{t+1}) \\ &= x_t + \Delta t \cdot x \cdot A \cdot u_t + \Delta t \cdot s \cdot A \cdot u_{t+1}. \end{aligned} \tag{18}$$

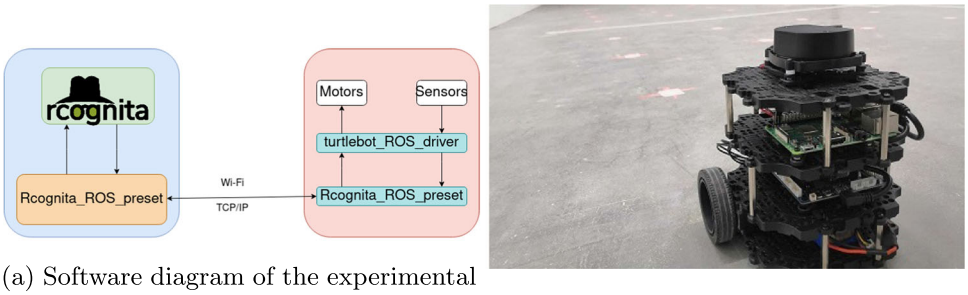
The optimization process should be applied for cost function optimization that would not perform straightforward calculations. Instead, that would build the computational graph of the function in order to minimize the number of computations required.

The actor cost objective depends not on the state but on the action performed. The constraints, however, are imposed on the state space by construction. In order to resolve this duality, the action sequence defined by the policy followed is employed for finding the predicted state sequence. This state sequence is further subjected to a check for entry into the restricted area of state space. If none of the states from this sequence are in this area, then the corresponding action sequence is assumed to be admissible. The optimization is performed on the space of all admissible action sequences. This approach may only sometimes lead to the optimal choice of action because of the absence of mutual uniqueness between actions and states. Therefore, hyper-parameters tuning is required to provide favorable optimization conditions.

Experimental setup

The study's aim is to evaluate RL-based algorithms for mobile robot motion control. To test the software in a real environment, the robot is utilized as a testing medium. MPC and SQL are implemented using a custom Python package

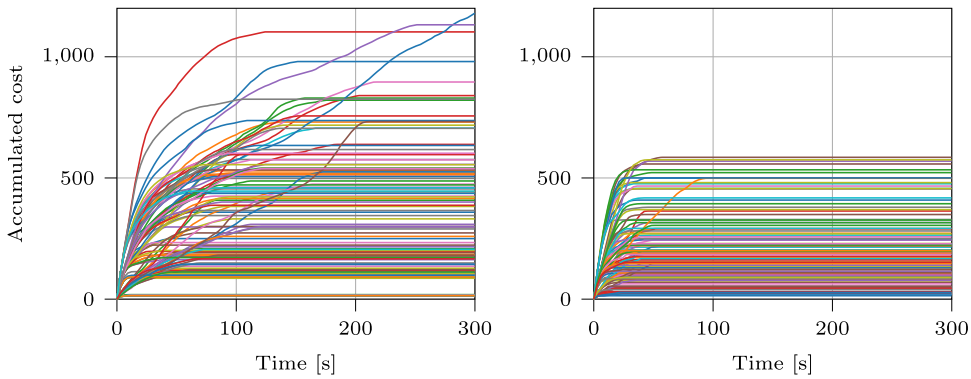
Fig. 6 Experimental system overview



(a) Software diagram of the experimental setup.

(b) Robotis TurtleBot3.

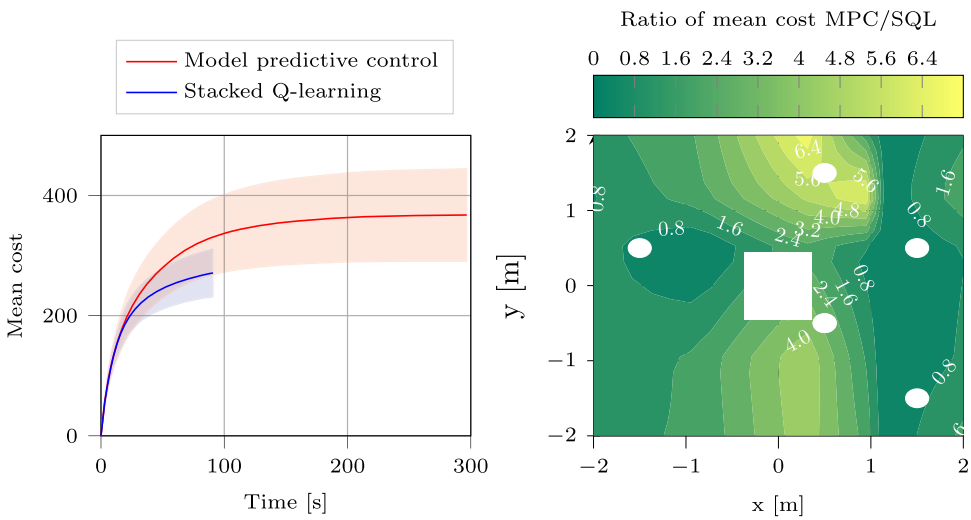
Fig. 7 Accumulated cost per each simulation for MPC and SQL algorithms



(a) Model predictive control

(b) Stacked Q-learning

Fig. 8 Contour and confidence plots



(a) Confidence plot

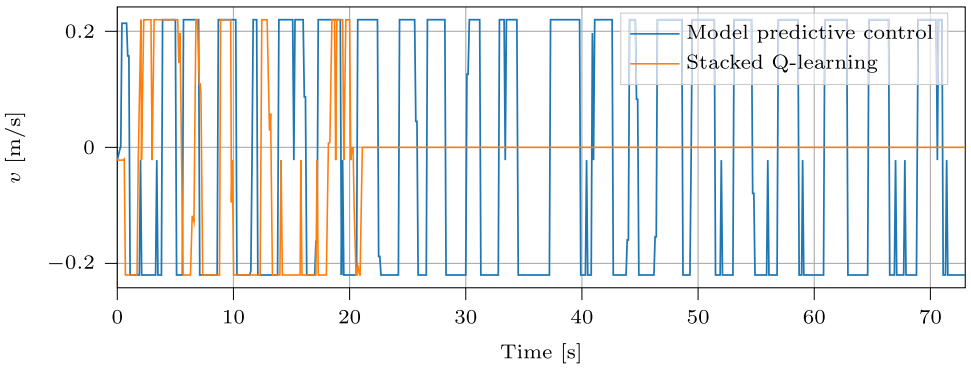
(b) Contour plot

developed explicitly for hybrid simulation of RL agents.[1] This package was further developed to integrate with the Robot Operating System (ROS), a widely-used tool for rapid prototyping in robotics (Fig. 6).

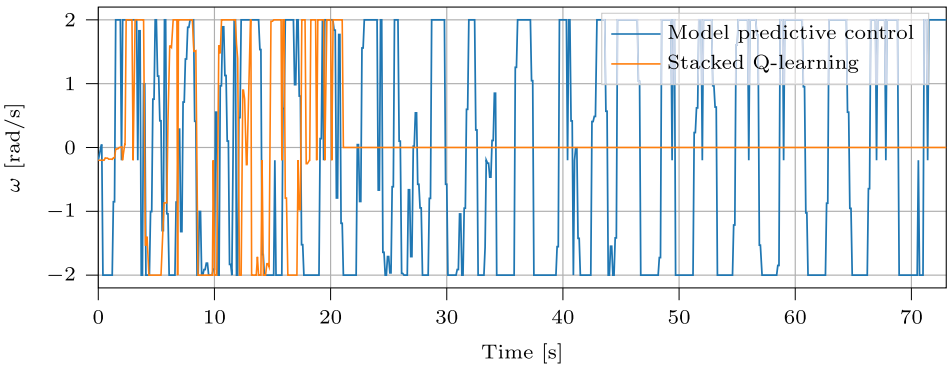
Robotis Turtlebot3

The TurtleBot3 Burger is a popular nonholonomic wheeled mobile robot developed by Robotis. It is a compact and affordable robot that can perform various tasks such as mapping, navigation, and object detection. The robot is equipped with a powerful onboard computer, a 360-degree LIDAR sensor for obstacle avoidance and mapping, and various other sensors for tracking its position and orientation. Its compact

[1] https://github.com/pavel-osinenko/rcognita.



(a) Linear velocities



(b) Angular velocities

Fig. 9 Linear and angular velocity for one of the tests

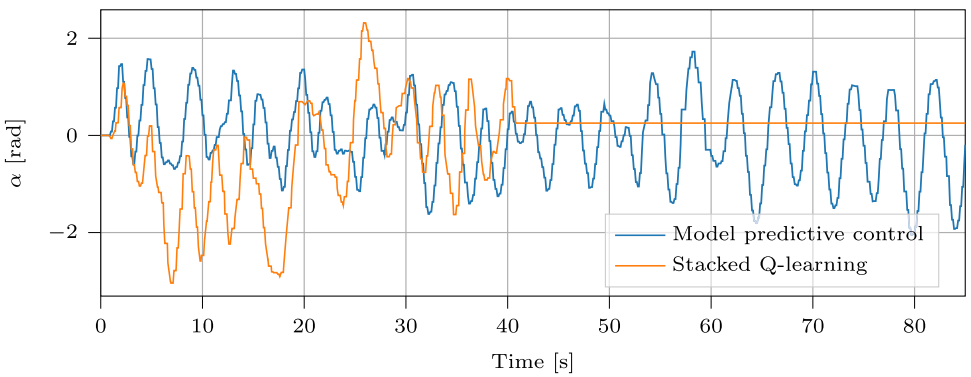


Fig. 10 Angle of yaw of the robot according to one of the tests

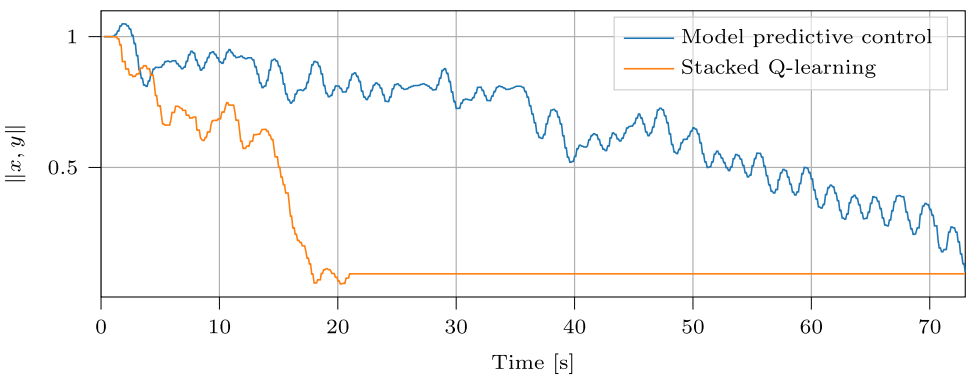


Fig. 11 Position error to one of the tests

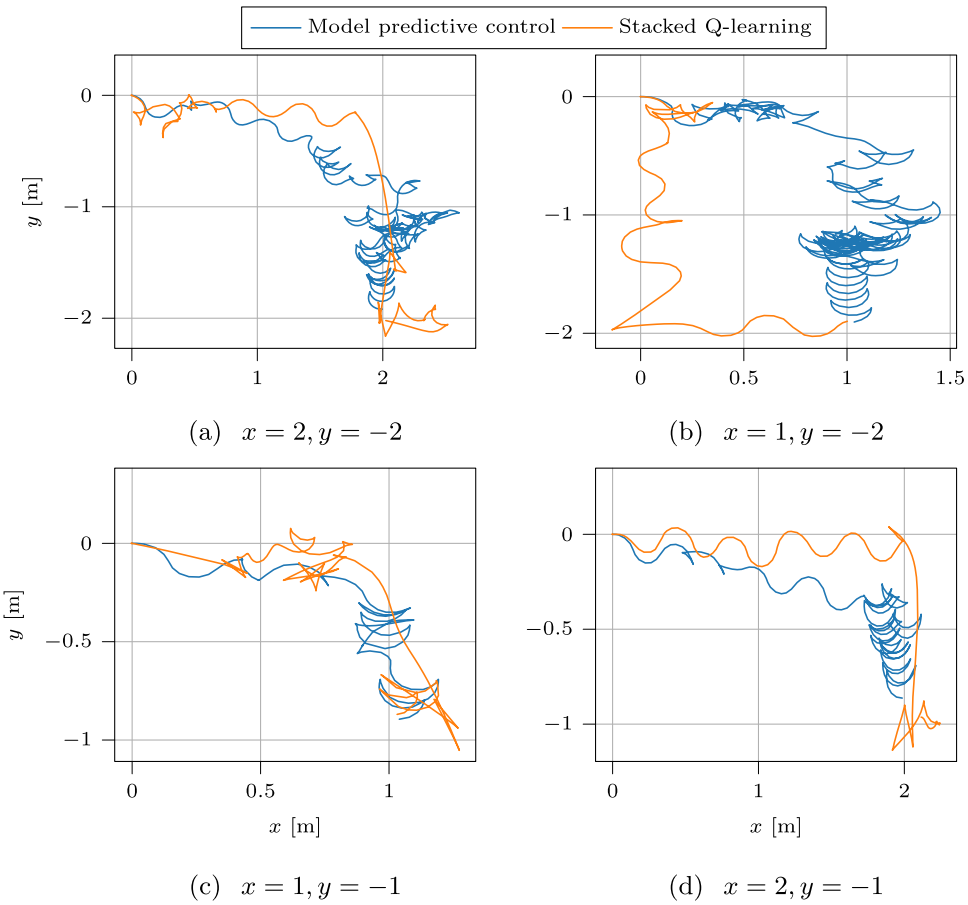


Fig. 12 Trajectories for different target points

size enable it to move smoothly in tight spaces and navigate around obstacles. It is often used for educational and research purposes due to its ease of use and versatility.

Results and discussion

The running cost was considered in the following quadratic form:

$$\rho = \chi^\top R \chi, \tag{19}$$

where $\chi = [x, u]$, R diagonal, positive-definite.

The critic structure was chosen quadratic as follows:

$$\begin{aligned} &\hat{Q}(x, u; \vartheta) := \vartheta \varphi^\top(x, u), \\ &\varphi(x, u) := \text{vec}\left(\Delta_u \left([x|u] \otimes [x|u]\right)\right), \end{aligned} \tag{20}$$

where ϑ is the critic weights, φ is the critic activation function, Δ_u is the operator of taking the upper triangular matrix, vec is the vector-to-matrix transformation operation, $[x|u]$ is the stack of vectors x and u, $\otimes$ is the Kronecker product.

Prediction of the robot's future states is carried out using the model (2). For the verification of considered algorithms, a number of experimental runs for different setup were carried out. Experiments for four various starting positions were done, see Fig. 2b.

Initially, position of the robot $p_b^w = \begin{bmatrix} x & y & \theta \end{bmatrix}$ and goal $p_g^w = \begin{bmatrix} x_g & y_g & \theta_g \end{bmatrix}$ are given in the same frame (see Fig. 2a). It was required to express the coordinates of the robot in the target frame (Spong et al., 2020). At this step we use the transformation matrix between the robot's frame and the target's frame in the following form

$$T(\theta_g) = \begin{bmatrix} \cos\theta_g & -\sin\theta_g & 0 & x_g \\ \sin\theta_g & \cos\theta_g & 0 & y_g \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Therefore, frame transformation takes the form

$$p_g^b = T p_b^w.$$

Experiments of the mobile robot parking problem with obstacle avoidance were performed. For MPC and SQL algorithms, 120 tests were performed on $x = [-2, 2]$ and $y = [-2, 2]$ regions with five obstacles each. The hyperparameters of the algorithms were chosen as follows: $\delta = 0.1$,

$H = 4$. When the plant reaches the set boundary relative to the desired position $||x, y|| < \Delta$, $\Delta = 0.1$, the test is terminated.

Figure 7 shows the accumulated running costs for each simulation. SQL results in a lower cost distribution compared to MPC. This is more clearly presented in Fig. 8a, which shows the average costs obtained by the algorithm with confidence intervals. Note that the transition times of SQL are significantly lower compared to MPC. To construct a contour diagram for each passed target position, the ratio of average accumulated costs at that point was calculated by the ratio $J_{MPC/SQL} = \frac{J_{MPC}}{J_{SQL}}$. The result is shown in Fig. 8b. As can be seen from the figure, MPC loses to SQL in the preponderance of target positions.

For a clearer picture of the processes in the experiments, Figs. 9, 10, 11 and 12 show samples of trajectories, control signals, yaw angles, and error rates. Figure 9 shows the control signals, representing the robot's angular and linear velocities. In the transient plots, it can be seen that SQL method provides faster convergence compared to MPC. It also provides a stable fulfillment of the control goal, while MPC, allows high oscillations in the transient time diagram. Such effects can also lead to undesirable consequences. The norm for the linear coordinates of the robot's position is shown in Fig. 11. Here we can observe that MPC provides convergence of the error to a comparable value as SQL (note the longer time and oscillatory nature of the convergence), but the robot orientation does not converge to the desired value. These same effects can be seen in the plots of the resulting trajectories Fig. 12. It is due to the short prediction horizon setup. Increasing the prediction horizon lead to better MPC performance. However, we note that as the prediction horizon increases, the computational complexity also increases, as noted, for example, in Osinenko and Dobriborsci (2021).

Finally, based on the data presented, we can conclude that the SQL agent is superior to the MPC for a shorter control horizon in the given task of parking a mobile robot with obstacle avoidance.

Conclusion

This study demonstrates the promise of implementing RL-based algorithms in robotics applications. The work extends the previously obtained and simulation-validated results to the level of experimental studies with added value. In particular, the effectiveness of constrained SQL is shown and compared to the well-known MPC. The proposed solution does not require a known map and path planning, which may be promising for dynamic obstacles. Note that the proposed approach works online (no pre-training is required), significantly reducing the computational load and completely removing the requirements for datasets' availability. However, implementing the algorithm requires a mathematical model of the control object. The proposed method performed better regarding accumulated cost results than the baseline while maintaining the same computational complexity.

Author contributions All authors contributed to the study conception and design. Material preparation and analysis were performed by DD, PO and WA. Data collection, simulation runs as well as experimental research and analysis of the results were done by RZ and MK. The first draft of the manuscript was written by DD and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Funding This work was supported by the Deggendorf Institute of Technology, Germany.

Data availability All data including source code and instructions are given in our public repository (Dmitrii Dobriborsci, 2023). Readers can reproduce all results both simulations and real-world experiment.

Declarations

Conflict of interest The authors have no conflicts of interest to declare that are relevant to the content of this article.

References

Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., et al. (2019). Solving rubik's cube with a robot hand. arXiv preprint arXiv:1910.07113

Beckenbach, L., Osinenko, P., Gohrt, T., & Streif, S. (2018). Constrained and stabilizing stacked adaptive dynamic programming and a comparison with model predictive control. In *2018 European control conference (ECC)* (pp. 1349–1354). IEEE.

Beckenbach, L., Osinenko, P., & Streif, S. (2020). On closed-loop stability of model predictive controllers with learning costs. In *2020 European control conference (ECC)* (pp. 184–189). IEEE

Berkenkamp, F., Turchetta, M., Schoellig, A. P., & Krause, A. (2017). Safe model-based reinforcement learning with stability guarantees. *Advances in Neural Information Processing Systems, 30*, 909–919.

Bhardwaj, M., Choudhury, S., & Boots, B. (2020). Blending MPC & value function approximation for efficient reinforcement learning. arXiv preprint arXiv:2012.05909

Borno, M., de Lasa, M., & Hertzmann, A. (2013). Trajectory optimization for full-body movements with complex contacts. *IEEE Transactions on Visualization and Computer Graphics, 19*, 1405–14. https://doi.org/10.1109/TVCG.2012.325

Davari, M., Alipour, K., Hadi, A., & Tarvirdizadeh, B. (2017). Learning a model-free robotic continuous state-action task through contractive q-network. In *2017 Artificial intelligence and robotics (IRANOPEN)* (pp. 115–120). https://doi.org/10.1109/RIOS.2017.7956453

Degris, T., Pilarski, P. M., & Sutton, R. S. (2012). Model-free reinforcement learning with continuous action in practice. In *2012 American control conference (ACC)* (pp. 2177–2182). https://doi.org/10.1109/ACC.2012.6315022

Dmitrii Dobriborsci, P. O. (2023). PredRL: Mapless navigation for mobile robot. https://github.com/thd-research/PredRL-robot-navigation

Dobrevski, M., & Skočaj, D. (2021). Deep reinforcement learning for map-less goal-driven robot navigation. *International Journal of Advanced Robotic Systems, 18*(1), 1729881421992621. https://doi.org/10.1177/1729881421992621

Dobriborsci, D., & Osinenko, P. (2022). An experimental study of two predictive reinforcement learning methods and comparison with model-predictive control. *IFACPapersOnline, 55*(10), 1545–1550.

Dong, H., Weng, C.-Y., Guo, C., Yu, H., & Chen, I.-M. (2021). Real-time avoidance strategy of dynamic obstacles via half model-free detection and tracking with 2d lidar for mobile robots. *IEEE/ASME Transactions on Mechatronics, 26*(4), 2215–2225. https://doi.org/10.1109/TMECH.2020.3034982

Drews, P., Williams, G., Goldfain, B., Theodorou, E. A., & Rehg, J. M. (2017). Aggressive deep driving: Combining convolutional neural networks and model predictive control. In Levine, S., Vanhoucke, V., & Goldberg, K. (Eds.), *Proceedings of the 1st annual conference on robot learning. Proceedings of machine learning research* (Vol. 78, pp. 133–142). PMLR. http://proceedings.mlr.press/v78/drews17a.html

García, J., & Fernández, F. (2015). A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research, 16*(42), 1437–1480.

Grüne, L., & Rantzer, A. (2008). On the infinite horizon performance of receding horizon controllers. *IEEE Transactions on Automatic Control, 53*(9), 2100–2111.

Guldenring, R., Görner, M., Hendrich, N., Jacobsen, N. J., & Zhang, J. (2020). Learning local planners for human-aware navigation in indoor environments. In *2020 IEEE/RSJ International conference on intelligent robots and systems (IROS)* (pp. 6053–6060). https://doi.org/10.1109/IROS45743.2020.9341783

Gullapalli, V., Franklin, J. A., & Benbrahim, H. (1994). Acquiring robot skills via reinforcement learning. *IEEE Control Systems Magazine, 14*(1), 13–24. https://doi.org/10.1109/37.257890

Hoeller, D., Farshidian, F., & Hutter, M. (2020). Deep value model predictive control. In *Conference on robot learning* (pp. 990–1004).

Huang, Y., Xie, K., Bharadhwaj, H., & Shkurti, F. (2020). Continual model-based reinforcement learning with hypernetworks. arXiv:2009.11997

Kulhánek, J., Derner, E., & Babuška, R. (2021). Visual navigation in real-world indoor environments using end-to-end deep reinforcement learning. *IEEE Robotics and Automation Letters, 6*(3), 4345–4352.

Kumar, V., Todorov, E., & Levine, S. (2016). Optimal control with learned local models: application to dexterous manipulation. In *2016 IEEE International conference on robotics and automation (ICRA)* (pp. 378–383). https://doi.org/10.1109/ICRA.2016.7487156

Lambert, N. O., Wilcox, A., Zhang, H., Pister, K. S. J., & Calandra, R. (2020). Learning accurate long-term dynamics for model-based reinforcement learning. In *2021 60th IEEE Conference on decision and control (CDC)* (pp. 2880–2887). IEEE.

Lenz, I., Knepper, R. A., & Saxena, A. (2015). *Deepmpc: learning deep latent features for model predictive control*. In Robotics: Science and systems.

Mayne, D. Q. (2014). Model predictive control: Recent developments and future promise. *Automatica, 50*(12), 2967–2986.

Mirowski, P. W., Grimes, M. K., Malinowski, M., Hermann, K. M., Anderson, K., Teplyashin, D., Simonyan, K., Kavukcuoglu, K., Zisserman, A., & Hadsell, R. (2018). Learning to navigate in cities without a map. In *NeurIPS*.

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. (2015). Human-level control through deep reinforcement learning. *Nature, 518*(7540), 529–533.

Napat, K., Valls, M. I., Hoeller, D., & Hutter, M. (2020). Practical reinforcement learning for MPC: learning from sparse objectives in under an hour on a real robot. In *2nd Annual conference on learning for dynamics and control (L4DC 2020)*.

Osinenko, P. (2020). Rcognita: A framework for hybrid agent-environment simulation. Retrieved from https://github.com/AIDynamicAction/rcognita

Osinenko, P., Beckenbach, L., Göhrt, T., & Streif, S. (2020). A reinforcement learning method with closed-loop stability guarantee. *21th IFAC World congress*. Accepted manuscript arXiv:2006.14034

Osinenko, P., & Dobriborsci, D. (2021). Effects of sampling and prediction horizon in reinforcement learning. *IEEE Access, 9*, 127611–127618. https://doi.org/10.1109/ACCESS.2021.3112498

Osinenko, P., Dobriborsci, D., Yaremenko, G., & Malaniya, G. (2023). A generalized stacked reinforcement learning method for sampled systems. *IEEE Transactions on Automatic Control*. https://doi.org/10.1109/TAC.2023.3250032

Osinenko, P., Göhrt, T., Devadze, G., & Streif, S. (2017). Stacked adaptive dynamic programming with unknown system model. *IFAC-PapersOnLine, 50*(1), 4150–4155. https://doi.org/10.1016/j.ifacol.2017.08.803

Osinenko, P., Göhrt, T., Devadze, G., & Steif, S. (2016). Stacked model-free adaptive dynamic programming using kalman-filter estimation (submitted manuscript). In *Proceedings of the 20th IFAC World congress*.

Polydoros, A. S., & Nalpantidis, L. (2017). Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent and Robotic Systems: Theory and Applications, 86*(2), 153–173. https://doi.org/10.1007/s10846-017-0468-y

Primbs, J. A., Nevistić, V., & Doyle, J. C. (1999). Nonlinear optimal control: A control Lyapunov function and receding horizon perspective. *Asian Journal of Control, 1*(1), 14–24. https://doi.org/10.1111/j.1934-6093.1999.tb00002.x

Przybyła, M. (2017). Detection and tracking of 2d geometric obstacles from lrf data. In *2017 11th international workshop on robot motion and control (RoMoCo)* (pp. 135–141). https://doi.org/10.1109/RoMoCo.2017.8003904

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., & Ng, A. (2009). Ros: an open-source robot operating system. In *Proceedings of the IEEE international conference on robotics and automation (ICRA) workshop on open source robotics*.

Shi, H., Shi, L., Xu, M., & Hwang, K.-S. (2020). End-to-end navigation strategy with deep reinforcement learning for mobile robots. *IEEE Transactions on Industrial Informatics, 16*(4), 2393–2402. https://doi.org/10.1109/TII.2019.2936167

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature, 529*(7587), 484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2018). A general reinforcement learning algorithm that masters Chess, Shogi, and go through self-play. *Science, 362*(6419), 1140–1144.

Song, Y., & Scaramuzza, D. (2007). Learning high-level policies for model predictive control. arXiv:2007.10284

Spong, M. W., Hutchinson, S., & Vidyasagar, M. (2020). *Robot modeling and control*. Wiley.

Sun, W., Jiang, N., Krishnamurthy, A., Agarwal, A., & Langford, J. (2019). Model-based RL in contextual decision processes: PAC bounds and exponential improvements over model-free approaches. In *Conference on learning theory* (pp. 2898–2933). PMLR.

Surmann, H., Jestel, C., Marchel, R., Musberg, F., Elhadj, H., & Ardani, M. (2020). Deep reinforcement learning for real autonomous mobile robot navigation in indoor environments.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. A Bradford Book.

Szita, I., & Szepesvári, C. (2010). Model-based reinforcement learning with nearly tight exploration complexity bounds. In *ICML* (pp. 1031–1038). https://icml.cc/Conferences/2010/papers/546.pdf

Tassa, Y., Erez, T., & Todorov, E. (2012). Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International conference on intelligent robots and systems* (pp. 4906–4913). https://doi.org/10.1109/IROS.2012.6386025

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., et al. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature, 575*(7782), 350–354.

Xiao, C., Wu, Y., Ma, C., Schuurmans, D., & Müller, M. (2019). Learning to combat compounding-error in model-based reinforcement learning.

Yahya, A., Li, A., Kalakrishnan, M., Chebotar, Y., & Levine, S. (2017). Collective robot reinforcement learning with distributed asynchronous guided policy search. In *2017 IEEE/RSJ International conference on intelligent robots and systems (IROS)* (pp. 79–86). IEEE.

Zanon, M., & Gros, S. (2021). Safe reinforcement learning using robust MPC. *IEEE Transactions on Automatic Control, 66*(8), 3638–3652. https://doi.org/10.1109/TAC.2020.3024161

Zhelo, O., Zhang, J., Tai, L., Liu, M., & Burgard, W. (2018) Curiosity-driven exploration for mapless navigation with deep reinforcement learning. https://doi.org/10.48550/ARXIV.1804.00456

Zhu, Y., Mottaghi, R., Kolve, E., Lim, J. J., Gupta, A., Fei-Fei, L., & Farhadi, A. (2017). Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *IEEE International conference on robotics and automation*.