Quantum Deep Reinforcement Learning for Robot Navigation Tasks

HANS HOHENFELD, DIRK HEIMANN, FELIX WIEBE, AND FRANK KIRCHNER

Robotics Research Group, University of Bremen, 28359 Bremen, Germany
Robotics Innovation Center (RIC), 28359 Bremen, Germany
German Research Center for Artificial Intelligence (DFKI), 28359 Bremen, Germany

Corresponding author: Hans Hohenfeld (hans.hohenfeld@uni-bremen.de)

This work was supported in part by German Federal Ministry for Economic Affairs and Climate Action (BMWK) and German Aerospace Center e.V. (DLR e.V.) through the Project QINROS (DFKI) under Project 50RA2033, and in part by the University of Bremen under Project 50RA2032.

ABSTRACT We utilize hybrid quantum deep reinforcement learning to learn navigation tasks for a simple, wheeled robot in simulated environments of increasing complexity. For this, we train parameterized quantum circuits (PQCs) with two different encoding strategies in a hybrid quantum-classical setup as well as a classical neural network baseline with the double deep Q network (DDQN) reinforcement learning algorithm. Quantum deep reinforcement learning (QDRL) has previously been studied in several relatively simple benchmark environments, mainly from the OpenAI gym suite. However, scaling behavior and applicability of QDRL to more demanding tasks closer to real-world problems, e.g., from the robotics domain, have not been studied previously. Here, we show that quantum circuits in hybrid quantum-classic reinforcement learning setups are capable of learning optimal policies in multiple robotic navigation scenarios with notably fewer trainable parameters compared to a classical baseline. Across a large number of experimental configurations, we find that the employed quantum circuits outperform the classical neural network baselines when equating for the number of trainable parameters. Yet, the classical neural network consistently showed better results concerning training times and stability, with at least one order of magnitude of trainable parameters more than the best-performing quantum circuits. However, validating the robustness of the learning methods in a large and dynamic environment, we find that the classical baseline produces more stable and better performing policies overall. For the two encoding schemes, we observed better results for consecutively encoding the classical state vector on each qubit compared to encoding each component on a separate qubit. Our findings demonstrate that current hybrid quantum machine-learning approaches can be scaled to simple robotic problems while yielding sufficient results, at least in an idealized simulated setting, but there are yet open questions regarding the application to considerably more demanding tasks. We anticipate that our work will contribute to introducing quantum machine learning in general and quantum deep reinforcement learning in particular to more demanding problem domains and emphasize the importance of encoding techniques for classic data in hybrid quantum-classical settings.

INDEX TERMS Reinforcement learning, autonomous agents, robotics, quantum machine learning, quantum computing.

I. INTRODUCTION

Robotics research and applications pose various algorithmic challenges, ranging from large-scale optimization, processing of high-dimensional sensory input, planning the execution of complex tasks in demanding environments, and learning of autonomous, adaptable behaviors. On the latter, deep reinforcement learning is used to produce impressive results in tasks such as learning complex manipulation behaviors [1], reaching, tracking and, navigation [2], manipulation based on visual input [3] as well as dexterous hand movements [4] among many others. It constitutes a central aspect of modern robotics research, but its scaling behavior and applicability to more complex scenarios remain an open question.

The associate editor coordinating the review of this manuscript and approving it for publication was Wai-Keung Fung.
role on the path toward autonomous and life-long learning robots.

Quantum computing algorithms [5] present a novel way of approaching algorithmic problems and offer theoretical advantages over classical algorithms for specific problems like factoring numbers [6], unstructured search [7], and solving systems of linear equations [8]. With more development and further resources, quantum computing and, in particular quantum machine learning [9] may contribute to the development of artificial intelligence in general and the learning of autonomous behaviors for robots in particular [10].

The idea of robots controlled by quantum computers, interacting with an environment on the scale of individual quantum states has arguably first been hypothesized and described by quantum computing pioneer Paul Benioff in the late 1990s and early 2000s [11], [13], [14]. While those envisioned Quantum Robots are very different from typical mechanical robotic systems as they can be found in various practical applications today, the idea of a mobile system utilizing quantum computing hardware remains intriguing.

Quantum computing technology has not yet reached the state of mobile, embedded, and potentially battery-powered quantum hardware but has made remarkable progress over the last two decades. Research institutions and companies are building quantum computers with increasing capabilities, and while current Noisy Intermediate-Scale Quantum Computers (NISQ) are limited in the number of qubits, coherence times, and fidelity of operation [15], they already enable exploring solutions for various problems [16].

One potential application for NISQ devices is the hybrid training of parameterized quantum circuits (PQCs) as machine learning models [17]. While this technique has been studied in various domains of machine learning [18], deep reinforcement learning has only recently attracted substantial research interest in this context. Existing works (see Sec. II-D) demonstrate the applicability of hybrid quantum-classical approaches for reinforcement learning tasks, with performances similar to classical algorithms while learning notably more compact models. However, their scope is currently limited to relatively simple benchmark environments, mainly from the OpenAI gym suite [19].

Or main contributions, illustrated in Fig. 1, are as follows. We demonstrate the feasibility of quantum deep reinforcement learning in three simulated robotic navigation tasks of increasing size and difficulty. Thereby, we extend the scope of previously introduced methods to substantially more complex tasks in the robotic domain, as we show by comparative experiments with typical benchmark environments. Furthermore, we compare to different encoding strategies for the classical state of the robot into a quantum circuit and also analyze the scaling behavior of the quantum circuits relative to a classical baseline. To validate the robustness of the presented methods, we additionally demonstrate their application in a substantially larger, more demanding and dynamic environment. In comparison to previous works in the field of quantum deep reinforcement learning, we thereby increase the complexity of considered learning tasks and furthermore provide a systematic evaluation of the scaling behaviour of quantum models in this context. Finally, we discuss various challenges and limitations of quantum deep reinforcement learning in a robotic context, as well as potential areas of research for quantum machine learning to contribute to the future advancements in autonomous robotics.

The rest of this paper is outlined as follows: In Sec. II, we provide an overview of previous works regarding deep reinforcement learning with PQCs. Subsequently, we outline the quantum deep reinforcement learning framework underlying this work in Sec. III. Afterwards, the learning setup with regard to the simulated environments and learning methods is documented in Sec. IV. We present the training results of the suggested methods compared to a classical baseline in Sec. V before summarizing our main findings and discussing their implications and limitations in Sec. VI. Finally, we give an outlook toward potential future research directions in Sec. VII.

II. RELATED WORK

Introducing quantum algorithmic techniques and quantum mechanical effects into reinforcement learning (RL) methods is an active and growing field of research. Meyer et al. [20] give an overview over various proposed methods and applications in this area. In the following, we highlight important methods and results from this line of research.

A. QUANTUM RL AND QUANTUM INSPIRED RL

Quantum mechanics and quantum computing were introduced reinforcement learning by Dong et al. [21], who proposed Quantum Reinforcement Learning (QRL). In the QRL algorithm, the classical states and actions of the agent are expressed in the orthonormal eigenbasis of a Hermitian observable. Actions are chosen by measuring in that basis from a superposition state, where the amplitudes of that superposition state are modified during learning utilizing amplitude amplification, the essential building block of Grover’s algorithm [7]. The authors evaluate the QRL algorithm in a discrete maze world, comparing it to the tabular TD(0) RL algorithm [22], achieving convincing performance. Quantum-inspired Reinforcement Learning (QiRL) [23] is
using a quantum-inspired probabilistic sampling technique to address the exploration vs. exploitation [22] problem in RL and a classical technique inspired by amplitude amplification to control the sampling probabilities. The algorithm is demonstrated on a simulated grid world and real-world robot navigation task with a wheeled MT-R robot. A variant of QiRL with flexible rotation angles in the amplitude amplification step is proposed in [24], which shows better performance on a UAV navigation problem compared to tabular Q-learning [25] with two different exploration strategies. Hu et al. [26] apply QRL to the MountainCar and Cartpole-v0 problems from the OpenAI Gym [19] suite, focusing on the exploration vs. exploitation problem, finding better overall learning performance compared to the classical Q-learning algorithm with an $\epsilon$-greedy policy. Quantum-inspired Experience Replay (QER) [27] is an extension of the concepts of QiRL to the representation of experiences and sampling from the replay buffer in Deep Reinforcement Learning (DRL), which the authors evaluate in several Atari 2600 game environments [19] and compare to baseline experience replay and prioritized experience replay with several variants of the DQN [26] algorithm.

This line of work with QRL and QiRL emphasizes the expression of states and actions in RL problems in quantum states, efficiently updating measurement probabilities leveraging amplitude amplification, and expressing the same concepts in a classical learning setup. QER extends these ideas to experience replay in DRL. Our contribution is conceptually different, as we focus on substituting classical neural networks in DRL with parameterized quantum circuits while keeping the learning algorithm and representation of all aspects of the learning task unchanged.

### B. QUANTUM ENVIRONMENTS

Dunjko et al. [28] propose a quantum-enhanced framework that, in principle, covers supervised, unsupervised, and reinforcement learning but, in its formulation, is closest to the latter. In this framework, the agent and the environment interact and as the learning procedure are kept classical. We give a detailed account of the underlying theory in Sec. III. The focus in this relatively new field so far has mostly been on showing the feasibility of the methods, understanding their capabilities and limitations, as well as finding quantum-classical separations in learning tasks.

In several works the Q-function approximation in the DQN algorithm is implemented by a PQC. Chen et al. [36] use basis encoding [37] followed by CNOT entanglements and parameterized Pauli rotations without a data re-uploading structure for the FrozenLake [19] and a CognitiveRadio task [38] with discrete state and action spaces. Lockwood et al. [39] use a different encoding technique and combine the parameterized circuit with quantum pooling operations [40] and classical neural network layers without data re-upload. This setup is able to learn a Blackjack environment but do not successful learn Cartpole-v0 [19]. In [41] the circuit layout and encoding scheme is similar to the one that we employ in this work. The architecture also includes data re-uploading and enables learning on FrozenLake and Cartpole-v0.

In addition, PQCs have also been used in the policy gradient methods REINFORCE [22]. In [42] the PQC architecture also includes data re-upload scheme. Included

### C. PROJECTIVE SIMULATION

Projective simulation [33] is an extension of the RL learning framework by an episodic and compositional memory, which allows the agent to predict potential future events using random walks on that memory. In [34], the authors propose an extension of this learning method using quantum walk on quantum memory instead to achieve a quadratic speed-up, which was later demonstrated in a proof-of-principle experiment on an ion-trap based quantum system by Srijunothai et al. [35].

The deep reinforcement learning algorithm we use in our work does not utilize any form of episodic memory, hence the suggested techniques in this line of research are not immediately applicable.

### D. QUANTUM DEEP REINFORCEMENT LEARNING

In quantum deep reinforcement learning (QDRL), the line of research from which our contribution originates, one or multiple classical neural networks are replaced or extended by parameterized quantum circuits. In contrast, agent-environment interaction and as the learning procedure are kept classical. We give a detailed account of the underlying theory in Sec. III. The focus in this relatively new field so far has mostly been on showing the feasibility of the methods, understanding their capabilities and limitations, as well as finding quantum-classical separations in learning tasks.
in the REINFORCE algorithm, the setup is able to solve Cartpole-v1, Mountaincar-v0 and Acrobot-v1. Additionally, the authors demonstrate experimentally and formally that hybrid quantum deep reinforcement learning can solve environments based on the discrete logarithm problem [43] which are intractable for classical learning methods. A variant of the REINFORCE algorithm is used in [44] to optimize PQCs, which replace the classical attention head layers originally introduced in [45], to solve the vehicle routing problem and achieve similar results as the classical counterpart.

Furthermore, actor critic methods such as proximal policy optimization (PPO) [46] and soft actor-critic (SAC) [47] have also been adapted with PQCs. In [48] the PPO algorithm is augmented with PQCs by exchanging the actor approximation network. The PQC has no data re-uploading scheme and is trained on Cartpole-v0 without completely solving it. In [49] unentangled PQCs with a fully connected classical layer as post processing unit replace the classical estimator for the actor and critic. This setup solves OpenAI Gym environments Cartpole-v1, Acrobot-v1 and LunarLander-v2.

Nagy et al. [50] simulate a hybrid quantum version of PPO on a photonic processor which demonstrates that PQC equivalences on photonic quantum computers can be used for reinforcement learning as well. In [51] the author demonstrates that the critic network in SAC can be exchanged with a PQC followed by a classical neural network and still solve the Pendulum-v0 problem from OpenAI gym with continuous state and action spaces.

Several works introduce parameterized quantum circuits into a hybrid quantum-classical learning setup, without strictly falling into the category of deep reinforcement learning. Cherrat et al. [52] implement a quantum version of policy iteration to solve FrozenLake and the InvertedPendulum environment. Franken et al. [53] implement a gradient-free method based on evolutionary methods to optimize a PQC that receives input data encoded by a tensor network. This setup is able to solve MiniGrid worlds [54] with discrete state space.

Our contribution extends upon these previous works in the following way:

- We extend the scope of QDRL to considerably more complex learning tasks from the robotic domain. We establish that increase in complexity by comparative experiments (see Appendix A).
- We systematically evaluate the scaling behaviour of parameterized quantum circuits in QDRL across task complexity as well as model size, which has previously not been done.
- We compare different encoding strategies suggested in the literature for re-uploading circuits with regards to their performance in a QDRL scenario.

We thereby extend the understanding of the feasibility of QDRL from very simple benchmark environments towards more realistic application scenarios from the robotics domain and contribute to the understanding of the model scaling behaviour in this context.

III. QUANTUM DEEP REINFORCEMENT LEARNING

A. DOUBLE DEEP Q-NETWORKS

For all our experiments, we used the Double Deep Q-Network (DDQN) [55] algorithm as it performed slightly better on average compared to e.g., the basic Deep Q-Network algorithm (DQN) [56]. DDQN is a model-free, off-policy deep RL algorithm that uses a neural network to approximate the Q-function from the basic Q-learning algorithm [25].

RL is used to solve Markov Decision Processes (MDPs), that is, discrete-time, stochastic processes \((S,A,T,r,p_0)\) with

- \(S\): The state space, a set of all possible states of an environment
- \(A\): The action space, a set of all possible actions for an agent
- \(T:S \times A \times S \rightarrow [0,1]\): The possibly stochastic transition function with \(T(s,a,s') = p(s'|s,a)\) being the probability of transitioning to state \(s'\) after taking action \(a\) in state \(s\).
- \(r:S \times A \times S \rightarrow \mathbb{R}\): A reward function with \(r(s,a,s')\) denoting a numeric reward for taking action \(a\) in state \(s\) and transitioning to state \(s'\) and
- \(p_0\): A probability for each state to be a starting state of the MDP.

The general scheme of interaction for an agent in an environment governed by an MDP is illustrated on the left side of Fig. 2. At each time step \(t\), the agent observes a state \(s_t\) of the environment, takes an action \(a_t\), which causes a transition to state \(s_{t+1}\) and the agent to receive a reward \(r_t\). The agent’s action selection is governed by a policy \(\pi:S \rightarrow A\) and the goal is to maximize the total cumulative reward \(R\).
for a possibly infinite time horizon, given by
\[
R = \sum_{t=0}^{\infty} \gamma^t r_t
\]
with \(\gamma \in [0, 1]\), called discount factor, encoding the preference for immediate over long-term rewards.

In the DDQN algorithm, this is achieved by learning an optimal action-value function \(Q : S \times A \rightarrow \mathbb{R}\). The action-value function \(Q(s, a)\) expresses the expected total cumulative reward for taking action \(a\) in state \(s\)
\[
Q(s, a) := (R)_{s,a,\pi},
\]
and the greedy policy can be expressed in terms of \(Q(s, a)\) by
\[
\pi(s) := \arg\max_a Q(s, a).
\]

The action-value function, also referred to as Q-function, is approximated by an artificial neural network \(Q_{\theta}\) with parameters \(\theta\). The neural network takes a state \(s \in S\) as input and computes \(Q(s, a^0)\) for all \(a^0 \in A\) as output. During learning, an \(\epsilon\)-greedy policy is employed by the agent, that is at each time step \(t\) with probability \(\epsilon \in [0, 1]\), the agent takes a random action from \(A\) to further explore the environment and with probability \(1 - \epsilon\), it follows the greedy policy (3) to exploit its current knowledge. At the beginning of training, \(\epsilon\) is commonly chosen with a value close to 1 and gradually reduced towards 0 as learning progresses.

Interactions \((s_t, a_t, r_{t+1}, s_{t+1})\) are stored in a replay buffer [57] from which at a predefined interval e.g., at each time step, a mini-batch is sampled to update \(Q_{\theta}\) with stochastic gradient descent, minimizing the loss
\[
\mathcal{L}(\theta) = (y_t - Q_{\theta}(s_t, a_t))^2,
\]
with \(y_t\) given by
\[
y_t = r_{t+1} + \gamma Q_{\theta'}(s_{t+1}, \arg\max_{a'} Q_{\theta'}(s_{t+1}, a')).
\]

The target network \(Q_{\theta'}\) is used to stabilize the training process [56]. It has the identical structure as \(Q_{\theta}\) and is periodically updated with \(\theta' \leftarrow \theta\).

### B. QUANTUM COMPUTING

We give a short introduction to the common notation of quantum computing and refer the interested reader to [58] for a comprehensive explanation of basic and advanced concepts of this topic. The fundamental objects in quantum computing are qubits, analogous to bits in classical computing. Unlike classical bits, which can be in one of the two states, 0 and 1, a qubit can be in a state, which is a linear combination of those states. Using the bra-ket notation, a qubit state \(|\Psi\rangle\) can be written as
\[
|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle,
\]
where \(|0\rangle\) and \(|1\rangle\) are basis states of the underlying single-qubit Hilbert Space. During the probabilistic measurement process, the qubit will collapse to one of the two basis states, and \(|\alpha|^2\) and \(|\beta|^2\) can be interpreted as the probabilities for the respective basis states. Before the measurement, the state can be modified by applying quantum gates \(U\) to it,
\[
|\Psi'\rangle = U|\Psi\rangle,
\]
formally described by unitary operators \(U\). This formulation can be extended to multi-qubit systems by preparing an \(n\)-qubit quantum register. For quantum computers, this is commonly initialized in its computational basis state \(|0\rangle^\otimes n\).

### C. PARAMETERIZED QUANTUM CIRCUITS FOR DEEP REINFORCEMENT LEARNING

Variational quantum algorithms are a promising method to implement algorithms on current and near-term quantum computers as they are well suited for systems with a relatively small number of qubits, noisy operations, and limited coherence times [59]. Their basic principle of operation is the combination of a parameterized quantum circuit whose parameters are optimized by a classical optimizer toward the desired outcome while evaluating the quantum circuit with adjusted parameters at each optimization step [60]. First introduced in the context of variational quantum eigensolvers [61], they became a major research area in quantum machine learning [59].

A parameterized quantum circuit (PQC) is a series of unitary quantum gates \(U(\theta, x)\), which is applied to the computational basis of \(n\) qubits \(|0\rangle^\otimes n\). These gates are parameterized by variational parameters \(\theta\) and classical input data \(x\). Fig. 3 shows this general ansatz for a machine learning application.

The PQC’s quantum state \(|U(\theta, x)\rangle = U(\theta, x)|0\rangle^\otimes n\) is computed and measured for many repeated iterations to gather sufficient statistics for the expectation value
\[
\langle O \rangle \ prodspace{=} \langle U(x, \theta) | O | U(\theta, x) \rangle
\]
for an observable \(O\). \(|U(x, \theta)\rangle\) denotes the conjugate transpose of \(|U(\theta, x)\rangle\). The measured expectation value of the quantum computation can be interpreted as the computation of a parameterized function \(f_{O}(\theta, x)\) depending on the observable, circuit parameters, and input.

The parameters \(\theta\) are tuned with an appropriate method to fit a target function. E.g., in the domain of supervised machine learning, a loss function \(\mathcal{L}\) can be minimized by performing
using the function implementation of the layers. Fig. 4 depicts the circuit layout on the entanglement gates [66].

Various encoding methods for quantum machine learning tasks have been suggested [37], but recent results on the expressiveness of quantum circuits emphasize the advantages of repeated encodings, also referred to as data re-upload [65], [66]. Such an ansatz enables the circuit to compute functions of repeated encodings, also referred to as data re-upload [65], [66]. Such an ansatz enables the circuit to compute functions of repeated encodings, also referred to as data re-upload [65], [66].

In the following, $x_1$ denotes the set of all encoded and rescaled state features of the rescaled state $q$.

In the second case, we encode, in line with [65], three qubits $x_{i1}, x_{i2}, x_{i3}$ which are repeated $L$ times. The entire circuit is given by

$$U(\theta, x) = \prod_{l=1}^{L} \left( U_{\text{ent}} U_{\text{par}}(\theta_l) U_{\text{in}}(x_l) \right) U_{\text{ent}} U_{\text{par}}(\theta_0).$$

This operator is applied to the initial state $|0\rangle^n$ leading to the final state $|U(\theta, x)\rangle$, and the expectation value of an observable $\mathcal{O}$:

$$\langle \mathcal{O}_{\theta,s} \rangle := \langle \mathcal{O} | U(x(s), \theta) | \mathcal{O} \rangle.$$  

As observables, we choose Pauli-Z gates $\sigma_{z}^{(1)} \otimes \ldots \otimes \sigma_{z}^{(n)}$, each acting on another qubit to obtain $n$ different output values. The output values can either be directly interpreted as values for $Q(s, a)$ in the reinforcement learning scenario or combined, scaled, or further post-processed by any classical means including additional classical neural network layers.

Let $a_i$ be one action of the action space $A = \{a_0, \ldots, a_n\}$ with $n_a \leq n_s$. If $n_a < n_s$, the PQC output values can either be combined, e.g., by multiplying some of them [41], to reduce the number of output values to the number of possible actions $n_a$, or the first $n_a$ qubits are measured. Four our comparative experiments with the Cartpole-v0
FIGURE 5. The three simulated static navigation environments for the Turtlebot 2 robot. In each, the robot has to navigate from its starting position in the upper left corner to the position marked with a green circle in the lower right while avoiding collisions with the enclosing walls and any obstacles. With the configured control scheme, this takes about 20 steps in the $3 \times 3$ environment (left), 30 in the $4 \times 4$ (center), and 45 steps in the $5 \times 5$ environment (right) for a (near) optimal trajectory. Possible paths the robot can take to solve each environment are marked with a red dotted line.

FIGURE 6. Dynamic environment in which the robot is equipped with a front facing lidar, depicted with orange rays. The robot starts in the center of the environment, the goal position is sampled at random from either of the four corners at the start. While navigating to the goal, the robot has to avoid several static and moving obstacles. The trajectories of the moving obstacles is indicated by green arrows. Solving the environment takes between 60 and 70 individual steps, depending on the sampled goal, position of dynamic obstacles and path the robot takes.

In our experiments, we use four environments based on a simulated Turtlebot 2 robot.\footnote{https://www.turtlebot.com/turtlebot2/} We chose this robotic system as it enables relatively simple yet realistic navigation tasks while being a readily available and extensible system we can build upon in future work. The robot is controlled via two independent motors by setting target velocities for its two wheels.

The first three environments are static navigation tasks depicted in Fig 5. The $3 \times 3$ environment shown on the left is the smallest, the $4 \times 4$ environment (center) is of medium size, and the $5 \times 5$ environment (right) is the largest. In each environment, the robot starts at a fixed position in the upper left corner and has to navigate to a fixed goal position marked with a green sphere while avoiding collisions with the outer walls and the obstacles within the environment. The robot has a state space with three components for these tasks. The first two are its position in the plain in $s_x$ and $s_y$ coordinates, and the third is its orientation $s_\phi$ around the $z$-axis in radians. We use these environments to understand the scaling behavior of parameterized quantum circuits in the learning task, assess their behavior and performance for trajectories of increasing length and complexity and evaluate the effect of an increasing exploration demand.

We furthermore created a considerably more demanding environment to validate the robustness of the presented method with a higher dimensional state space and dynamic components in the learning task. In this environment, shown in Fig. 6, the robot is equipped with a simple, front-facing lidar that covers a range of $180^\circ$ in the plane. The robot’s state space contains ten distance measures in $20^\circ$ intervals as well as the current distance and orientation to the goal. The robot starts in the center of the environment and has to navigate to a goal position, which is sampled at random at the beginning of each episode to be in either of the four corners.

We created all environments with the pybullet\cite{67} real-time physics engine and set a control frequency of 100 Hz for collision detection and calculating forward dynamics.

The robot has three actions available (forward, turn left, turn right) to move in the environment. To move forward, the same target velocity is applied to both wheels, whereas for turning left and right, equal velocities but with opposing directions are set. Turning left or right causes a change in orientation between $40^\circ$ and $50^\circ$ depending on the current forward and angular velocity of the robot. Similarly, the

IV. METHOD

A. ENVIRONMENTS

In our experiments, we use four environments based on a simulated Turtlebot 2 robot.\footnote{https://www.turtlebot.com/turtlebot2/} We chose this robotic system as it enables relatively simple yet realistic navigation tasks while being a readily available and extensible system we can use the former strategy, for the dynamic robot navigation environment the latter.

In our learning scenario, the Q-values range exceeds the interval $[-1, 1]$ and thus needs additional post-processing. The authors of \cite{41} suggest rescaling each output value with an additional trainable parameter $a_j$:

$$Q(s, a_j) = \langle \sigma_z^{(j)} \rangle \theta \cdot w_j,$$

which adds $n_a$ trainable output variables to the model.
robot moves between 0.15 and 0.2 units in the direction of its current orientation, where one unit corresponds to the length of one square on the environment floor. An action is chosen every 50 simulation steps, corresponding to an execution time of 0.5 seconds. With this control scheme, the robot needs about 20 consecutive actions to reach the goal in the 3 × 3 environment on a near-optimal trajectory, 30 steps in the 4 × 4 environment, and 45 steps in the large 5 × 5 environment. Possible paths the robot can take to solve the static environments are marked with red dotted lines in Fig. 5. In the dynamic environment, where the robot is equipped with a lidar, a typical trajectory leading to the goal takes about 60 to 70 steps, depending on the current goal, position of dynamic objects and path taken by the robot.

We use the same simple yet informative reward function to train the robot in all environments. The agent receives a positive reward for decreasing the distance to the goal as well as for reaching it, whereas increasing or maintaining the distance as well as collisions are penalized. The reward function is given by:

\[
    r(s_t, s_{t+1}) = \begin{cases} 
    10.0 & \text{if } s_{t+1} \text{ is within the goal area} \\
    0.1 & \text{if } d_{\text{goal}}(s_{t+1}) < d_{\text{goal}}(s_t) \\
    -1.0 & \text{for any collision} \\
    -0.2 & \text{else} 
    \end{cases}
\]

(16)

where \(d_{\text{goal}} : S \rightarrow \mathbb{R}\) is the euclidean distance of the robot to the goal area. The penalties in the reward function ensure that shorter trajectories are preferred by the agent. We consider the static environments solved when a total reward of 10.5 (3 × 3), 11.0 (4 × 4), and 10.0 (5 × 5) is reached. Higher rewards are possible, as the two larger environments have more than one possible path to the goal and we furthermore allow some tolerance for the length of the trajectory and the exact route. Therefore, these thresholds are a lower bound based on several manually determined valid trajectories. For the dynamic environment we do not set a threshold and observe the average evaluation reward over the entire training time.

In all environments, an episode ends when the robot reaches the goal, collides with an object, or when a maximum of 200 steps were executed during training.

**B. LEARNING**

We trained the simulated robot using three different paradigms: A baseline with a classical neural network as approximator for the action-value function and two different parameterized quantum circuits, distinguished by their encoding strategy for the classical input data.

For the classical baseline agent, we employ a three-layer, fully connected neural network with rectified linear unit activation on all but the final layer, which has a linear activation. In the static environments, the network takes the three components of the robot’s state \(s = (s_x, s_y, s_g)\) as input, followed by two layers with \(u_1\) and \(u_2\) number of hidden units and three outputs corresponding to \(Q(s, a_i), i \in \{1, 2, 3\}\). For the dynamic environment, we use the same neural network architecture, albeit with a 12 dimensional input for the ten lidar distance measurements as well as the distance and orientation to the goal. The number of trainable parameters \(|\theta_{NN}|\) including weights and biases for the classical neural network is therefore given by:

\[
    |\theta_{NN}| = |s|u_1 + u_1u_2 + u_2 + 3u_2 + 3. 
\]

Here \(|s|\) is the dimensionality of the state space and \(u_i\) the units in the \(i\)-th layer.

In both quantum cases, we build our circuit on three qubits for the static environments, which aligns well with the dimensionality of the state space and the number of actions available to the agent. Both circuits follow the general approach depicted in Fig. 4 and are only different in their data encoding structure \(U_{\text{in}}\) and the number of layers \(L\). The circuit layout for a single layer \(L > 0\) is illustrated in Fig. 7, whereas the encoding strategies are shown in Fig. 8.

Our first data re-upload PQC model uses the encoding \(U_{\text{in}}^{(q)}\) on each qubit with the rotation gate \(R_x\) to encode one state feature on each qubit (PQC-1). For the second model, we use \(U_{\text{in}}^{(q)}\) for each qubit with rotation gates \(R_xR_yR_z\) to encode all three state features on each qubit (PQC-
TABLE 1. The configurations used for the classical baseline in all three static environments with the number of units \(u_1\) and \(u_2\) for the first two layers of the neural network and the number of trainable parameters \(|\theta_{NN}|\) for each configuration.

| Hidden units | Parameters | static \(|\theta_{NN}|\) |
|--------------|------------|------------------------|
| \(u_1\) \(u_2\) | \(258\) \(128\) | 34,307 |
| \(128\) \(128\) | 17,411 | 128 |
| \(64\) \(64\) | 8,963 | 64 |
| \(64\) | 2,435 | 64 |
| \(32\) \(32\) | 1,238 | 32 |
| \(32\) | 707 | 32 |
| \(16\) \(16\) | 387 | 16 |
| \(16\) | 227 | 16 |
| \(8\) \(8\) | 131 | 8 |

3). As introduced in Sec. III, we scale each feature with a trainable parameter that is individual for each encoding gate and furthermore apply an activation function for which we choose the arc tangent in all our experiments. The universal rotation \(U_{\text{par}}^{(q)}\) on each qubit is composed of three parameterized Pauli rotation gates \(R_x R_y R_z\) with trainable parameters.

For the large, dynamic environment with a 12 dimensional state space, we use circuits with 12 qubits. The PQC-1 as described above directly translate to this setting, whereas for the PQC-3 encoding we distribute all 12 features of the state space across four layers, each encoding three of the features, as outlined in Sec. III.

The number of trainable parameters for each quantum circuit \(|\theta_{\text{PQC}}|\) is the sum of variational parameters in the initial parameterized and the following \(L\) layers, the input scaling and the output scaling parameters, in total:

\[
|\theta_{\text{PQC}}| = 3Q(L+1) + Q n_{\text{enc}} L + 3, \tag{18}
\]

Here \(n_{\text{enc}} = 1\) for the PQC-1 and \(n_{\text{enc}} = 3\) for the PQC-3 encoding, \(Q\) is the number of qubits in the circuit.

Based on these three architectures, two quantum and one classical, we performed experiments with different sizes of each architecture for the static environment. For the classical neural network, we evaluated a total of ten configurations for the number of units \(u_1\) and \(u_2\) in the first and second hidden layers. The configurations and their number of trainable parameters \(|\theta_{\text{NN}}|\) are outlined in Table 1.

Likewise, we included ten configurations for each quantum encoding strategy with an increasing number of layers \(L\). As the different types of encoding lead to a different amount of trainable parameters for each layer, we arranged the number of layers to have an equal number of parameters between both. The number of layers \(L\) for both strategies, as well as the number of trainable parameters, including variational, input, and output scaling parameters are summarized in Table 2.

TABLE 2. The configurations used for both quantum encoding strategies while training the three static environments. The number of layers \(L\) as well as the number of trainable parameters \(|\theta_{\text{PQC}}|\), which include the variational, input, and output scaling parameters, are outlined.

| PQC-1 | PQC-3 | Parameters |
|-------|-------|-----------|
| \(L\) | \(L\) | static \(|\theta_{\text{PQC}}|\) |
| 12 | 8 | 156 |
| 15 | 10 | 192 |
| 18 | 12 | 228 |
| 21 | 14 | 264 |
| 24 | 16 | 300 |
| 27 | 18 | 336 |
| 30 | 20 | 372 |
| 33 | 22 | 408 |
| 36 | 24 | 444 |
| 39 | 26 | 480 |

With regard to the parameter scaling, we emphasize that the number of trainable parameters roughly doubles with each increase of the configuration size for the classical baseline, whereas the scaling for the quantum circuits is only linear. Thus, the largest neural network we employed has about two orders of magnitude (34,307) more parameters than the largest quantum circuits (480). With the dynamic environment, we only perform experiments with a single configuration for each architecture, due to the considerable computational effort involved in simulating very large quantum circuits. The neural network used as baseline has 256 and 128 hidden units (36,611 trainable parameters), the PQC-1 circuit has 24 layers, and the PQC-3 circuit 16 layers (bot 1,191 trainable parameters).

We set a learning rate for the stochastic gradient descent of \(10^{-3}\) for the classical baseline as well as for the variational parameters in both quantum circuit architectures. The input and output scaling parameters were trained with a learning rate of \(10^{-2}\) for both PQC-1 and PQC-3 as encoding.

For the hyper-parameters specific to the DDQN algorithm, we use the same values for all experiments. The replay buffer was set to a capacity of 20,000 experience samples and is initially filled with 5,000 samples from executing a fully random policy in the environment, before each training starts. The agent is trained after each step it executes in the environment with a mini-batch of 64 samples from the replay buffer. Exploration is handled with an \(\epsilon\)-greedy policy as introduced in Sec. III, starting at \(\epsilon = 1.0\) and setting \(\epsilon \leftarrow 0.99\epsilon\) every 250 training steps. Total training time is limited to 50,000 steps in all environments, except for the dynamic environment, in which we train 100,000 steps. We evaluate the current performance of the learned policy after every 100 training steps by performing 10 consecutive runs within the environment. Once the average total reward over those 10 runs surpasses the solution criterion for any of the static environment outlined above, the training is stopped early, whereas we do not stop the training early in the dynamic environment.
To gather sufficient data on the robustness and reproducibility of the learning procedure, we repeat the training for each combination of static environment, architecture, and configuration 20 times, each time with a different random seed. We do not set the random seeds to specific values but have them provided by the operating system’s randomness source instead. We consider a configuration successful if at least 15 of 20 training runs solve the environment. In the dynamic environment, we repeat each training 10 times and record the evaluation performance to evaluate the robustness of the presented methods in a considerably larger and more challenging environment and large quantum circuits.

All hyper-parameters were determined empirically before the actual experiments. Our main goal was to find a set of parameters that would enable reliable and robust training under mostly identical premises for all three architectures, their respective configurations, and for all three environments, as our main interest is not in absolute performance but in comparison of architectures and scaling behavior. An overview of all hyper-parameters can be found in Table 6 in App. B.

### C. HARDWARE, SOFTWARE AND COMPUTATIONAL RESOURCES

All experiments were conducted on a workstation equipped with an AMD Ryzen Threadripper Pro 3975WX 32 core/64 thread CPU, 128 GB of RAM, and an NVIDIA RTX A6000 GPU. On the software side, we used TensorFlow [68] as framework for all general and classical machine learning tasks, TensorFlow Quantum [69] for quantum machine learning specific tasks, as well as TensorFlow Agents [70] for all components related to Deep Reinforcement Learning and a stable DDQN implementation. TensorFlow Quantum integrates the Cirq [71] quantum computing framework for building and running quantum circuits, as well as the Qsim [72] quantum circuit simulator.

All quantum circuit simulations in Qsim were executed under idealized noise-free conditions. We compute the expected value of observables directly from the system’s state vector. If experiments were to be conducted in a shot-based simulation, a large enough number of circuit receptions would need to be chosen to estimate the expected values of observables with sufficient accuracy. Similarly, if experiments were to be reproduced on quantum hardware or with simulated hardware noise, appropriate measures for error mitigation would have to be taken into account, which is outside of the scope of this work.

All simulated robotic environments were built using the PyBullet [67] python bindings to the Bullet real-time physics SDK. For the baseline experiments described in App. A, we furthermore used the OpenAI Gym [19] suite.

We released our robotic environments as well as the entire experimental setup under an Open Source license for interested researchers to reproduce, verify, or build upon our work. Both can be found together with installation and usage instructions under the following addresses:

- Environments: https://github.com/dfki-ric-quantum/qdrl-turtlebot-env
- Experiments: https://github.com/dfki-ric-quantum/qdrl-turtlebot-eval

Concerning the computational resources and wall-clock time needed to conduct our experiments, we observe the following: For the classical baseline, training a single neural network within the range of configurations and across all environments requires 1.5 GB of RAM and 600 MB of VRAM, assuming TensorFlow uses GPU acceleration. Training the network for 1,000 steps takes on average 25 seconds wall-clock time with our hardware setup, which is relatively stable overall environments and network sizes.

For the three static environments, the number of qubits of a quantum circuit, which is the same in both our encodings and across all configurations, primarily determines the memory requirements for its simulation. Simulating the training of each quantum circuit requires about 2.2 GB of system memory and 500 MB of VRAM. The quantum circuit simulator imposes a substantial computational overhead, resulting in much longer execution in terms of wall-clock time and a nearly linear growth with respect to the number of layers. The average wall-clock time for 1,000 training steps in the 5 x 5 environment with the PQC-1 and PQC-3 encoding are summarized in Table 3.

Learning the dynamic environment with either encoding on 12 qubits for the number of layers we use, requires considerably more computational resources. A single run requires about 18 GB of system memory and 1.2 GB of VRAM. Training for 1,000 steps takes on average one hour, hence the training the full 100,000 steps is finished in about four days.

### V. RESULTS

#### A. OPENAI GYM ENVIRONMENTS

As we work with custom environments, we first compared their complexity to established OpenAI Gym environments, namely FrozenLake and Cartpole-v1. The results for

| Environment | PQC-1 | PQC-3 |
|-------------|-------|-------|
| FrozenLake  |       |       |
| Cartpole-v1 |       |       |

### TABLE 3. Average wall-clock time for 1,000 training steps with the PQC-1 and PQC-3 encoding in the 5 x 5 environment. The time necessary to train the model grows nearly linear in the number of layers.
both environments with classical neural networks and PQCs are described in Appendix A. With these comparative experiments, we can demonstrate that our navigation environments are indeed substantially more difficult to solve for the DDQN algorithm.

### B. STATIC ENVIRONMENTS

Performing experiments with the 10 classical neural network configurations and 10 configurations for both PQC input encoding variants provides insight into the scaling behavior and robustness across multiple training runs for each architecture in the given robotic reinforcement learning task. The complete statistics for all experiments in the static environments are outlined in Tables 7 to 9 in Appendix C and visualized in Fig. 9.

The first noteworthy result is that all three architectures, the classical neural network as well as both types of quantum circuits are capable of learning an optimal action-value function in all three environments in 20 out of 20 training runs with a sufficiently large configuration (see column Solved in the Tables 7 to 9). More precisely, for the 3 × 3 and 4 × 4 environments, all configurations of the architectures solve the environments, whereas in the 5 × 5, the three smallest neural networks, the two smallest PQC-1, and the smallest PQC-3 configurations were unable to solve the environment in at least 15 out of 20 runs. Also, we find that an increase of the model size in terms of the number of trainable parameters leads to a decreased median and mean in required training steps. This trend converges after the model size reaches a sufficient size. In our case, the two biggest configurations of all three architectures are similar in terms of median and mean of training steps. In addition, in most cases, the range and standard deviation decreases as the model size increases, resulting in our largest models being the best-performing and most stable configurations.

In the following, we focus on the two best PQC-1 and PQC-3 configurations and compare them with four different neural network configurations. From our data, we select the classical neural networks such that they have a similar number of parameters or one or two orders of magnitude more parameters than the PQC configurations. Table 4 summarizes the mean and standard deviation of the required training steps for these configurations. From this, we observe the following general trends with regard to the number of trainable parameters:

![Figure 9: Statistics on training time for all three static environments, architectures, and configurations. The results per environment are shown from the top to bottom row, whereas the classical baseline neural network architecture (NN), as well as both types of quantum circuits (PQC-1 and PQC-3) are arranged from left to right. For each combination of environment and architecture all related configurations, that is number of units \((u_1, u_2)\) for the classical baseline and number of layers \(L\) for both quantum encoding strategies, are reported. Each box shows the median training steps over 20 runs for each configuration with different random seeds as well as the lower and upper quartile, range and flier points.](image-url)
With about the same order of magnitude of parameters, the quantum circuits perform better and converge to an optimal solution faster. This is especially true for the PQC-3 architecture.

With about one order of magnitude more parameters for the classical neural network, its performance is about equal compared to the parameterized quantum circuits.

A further increase in the number of parameters up to two orders of magnitude more for the neural network puts it slightly ahead of both quantum circuit architectures in all observed metrics.

Furthermore, we can compare the results of the two best PQC-1 and PQC-3 configurations in Table 4. For all environments, the best PQC-3 architecture yields faster convergence to an optimal policy compared to the best PQC-1 encoding scheme.

This advantage is relatively stable across all three static environments, suggesting that for the navigation setup considered in this work, a larger number of encoding gates is beneficial. A larger variety of environments with regards to complexity and type of task to learn would need to be evaluated to make more definitive statements on this.

The evaluation performance for the best configuration for each architecture in all three environments is shown in Fig. 10. During training, we observed the agent’s performance with the trained policy every 100 steps for 10 consecutive runs and evaluated its mean reward. In all three environments, the classical baseline converges to an optimal policy faster, albeit with two orders of magnitude more trainable parameters. In the 3 × 3 environment, the PQC-3 architecture reaches a solution notably faster than the PQC-1 architecture. With increasing environment complexity, both types of quantum circuits perform increasingly similarly, whereas the classical neural network remains ahead of both. This finding emphasizes the trends discussed above.

C. DYNAMIC ENVIRONMENT

In the large, dynamic navigation environment, our main interest is the robustness of the presented method in a substantially more demanding task and employing considerably larger...
TABLE 4. Mean number of training steps and standard deviation in all three environments for the two largest configurations for both quantum circuit architectures in comparison to two classical baseline models with about the same order of magnitude of trainable parameters as well as two larger neural networks. We find, that with about the same order of magnitude of parameters, the two quantum architectures converge to an optimal solution in fewer training steps. With an order of magnitude more parameters, the classical neural network performs comparable or better and achieves better performance compared to both quantum architectures with further increasing number of trainable parameters.

| Arch. | Config. | Environment: | No. of training steps |
|-------|---------|--------------|-----------------------|
|       |         | 3 × 3        | 4 × 4                  | 5 × 5                  |
|       |         | Mean | Std. | Mean | Std. | Mean | Std. |
| NN    | (16,8)  | 21,075 | 5,050 | 35,910 | 7,672 | 49,515 | 1,684 |
|       | (16,16) | 16,405 | 3,129 | 24,565 | 13,736 | 46,570 | 5,237 |
|       | (64,32) | 10,635 | 2,290 | 15,055 | 4,315 | 32,135 | 8,049 |
|       | (256,128) | 7,495 | 1,359 | 11,480 | 3,859 | 22,220 | 4,122 |
| PQC-1 | 36      | 9,150 | 3,746 | 17,705 | 4,678 | 29,260 | 6,614 |
|       | 39      | 10,060 | 2,905 | 16,880 | 7,288 | 25,110 | 5,309 |
| PQC-3 | 24      | 6,850 | 3,146 | 14,665 | 6,068 | 25,757 | 7,334 |
|       | 26      | 9,515 | 3,347 | 15,875 | 5,164 | 23,635 | 7,535 |

TABLE 5. Statistics over the training in the dynamic navigation environment. For all three configurations the number of training steps to the best performing evaluation runs, the mean evaluation reward, the mean number of solved evaluation runs as well as their respective standard deviations are reported. Statistics are taken over 10 consecutive evaluation runs executed every 100 steps and 10 repetitions of the experiment with different random seeds.

| Arch. | Config. | Reward               | Solved               |
|-------|---------|----------------------|----------------------|
|       |         | Steps | Mean | Std. | Mean | Std. |
| NN    | (256,128) | 81,500 | 10.27 | 1.37 | 8.50 | 0.92 |
| PQC-1 | 24      | 94,200 | 5.50 | 3.12 | 6.70 | 1.79 |
| PQC-3 | 16      | 94,500 | 3.87 | 3.53 | 6.30 | 1.27 |

quantum circuits. To this end, we trained a classical baseline and two large quantum circuits with the two encoding strategies for 100,000 iterations on the environment. We evaluated the performance in 10 consecutive runs every 100 training steps. Fig. 11 shows the training progress regarding the mean evaluation reward and number of solved evaluation runs, with averages taken over 10 repetitions of the experiment with different random seeds.

The classical baseline neural network performs considerably better in this task than both employed quantum circuits, learns policies that achieve higher mean rewards, solves more evaluation runs on average and is more robust in the dynamic setting. Both quantum architectures perform about the same concerning to both metrics. The fact that the difference between the classical model and quantum circuits regarding the mean reward is larger than for the number of solved evaluation runs is explained by the observation that the environment allows for much larger negative rewards on failed runs than positive rewards on the successful ones. Hence, the negative rewards will dominate the result if several runs fail.

Table 5 summarizes the best results achieved by all three architectures. The classical baseline reaches its best average performance after 81,500 training steps, whereas both quantum circuits require more than 94,000 steps. Additionally, the mean evaluation reward of 10.27 for the classical neural network is considerably larger than 5.50 and 3.87 for the PQC-1 and PQC-3 architecture.

After this training duration, the robot can successfully navigate to the goal on average in 8.5 out of 10 evaluation runs over 10 repeated experiments. Solving 6.7 and 6.3 evaluation runs on average for the quantum architectures shows noteworthy training progress for both, but with considerably worse performance. We furthermore observe larger standard variations on both metrics for the quantum models compared to the classical baseline, suggesting less robust and less reliable training results.

VI. DISCUSSION

In this work, we investigated the potential and scaling of hybrid quantum deep reinforcement learning as a method to learn autonomous robotic behaviors. We systematically evaluated two different quantum circuit architectures in three simulated static environments of increasing difficulty and with increasing circuit sizes. These results were compared to a classical neural network baseline. Additionally, we tested the robustness of the presented method in a considerably more demanding learning task, using a dynamic navigation environment.

Both quantum architectures as well as the classical baseline yielded sufficient action-value functions for the simulated robot in all three static environments. Not considering the number of trainable parameters, the classical baseline models outperformed the quantum circuits in terms of training speed and stability. A noteworthy result, which is in line with previous findings from the quantum deep reinforcement learning research is that both best-performing quantum circuits were capable of solving the environments within a similar number of training steps as classical neural networks with about one order of magnitude more trainable parameters. This observation is consistent across all three environments. The best-performing quantum models have 444 and 480 trainable parameters, the classical baseline was sufficient to solve the 3 × 3 and 4 × 4 with a similar amount of parameters, albeit with substantially more training steps. At this model size, the neural network was unable to fit an optimal action-value function for the 5 × 5 environment in
most of the 20 training runs within the 50,000 training step threshold we set, whereas both quantum architectures still succeeded with only 300 parameters.

Comparing both quantum circuit architectures, we find that the PQC-3 embedding performs better than the PQC-1 embedding in all three environments, suggesting that in this context, having more encoding gates for the same data is beneficial, although the difference becomes less pronounced with increasing environment difficulty. Moreover, our experiments show that with increasing environment size, quantum circuits with more layers are needed to solve the tasks consistently, especially for the $5 \times 5$ environment. This finding is consistent with the results from [66], as adding more layers increases the expressiveness of the circuit, which makes it possible to approximate more complex action-value functions.

Testing the same learning methods in a more demanding, dynamic navigation environment, we find that both quantum circuit architectures get outperformed by the classical neural network regarding reward, solved evaluations, training duration and robustness. Given the limited scope of this experimental setup, it remains open, if this result can be improved by changes on the training procedure, circuit architecture, encoding strategies or by increasing the circuit size. We consider these questions to be out of scope for this work, but plan to address them in future.

Additionally, our results demonstrate that PQCs of this size are trainable in a quantum circuit simulator for a practical problem class, which does not necessarily follow from previous considerations on the expressiveness of PQCs [65], [66]. Beyond these results, we can confirm, similar to e.g., [73], that training PQCs is fairly unstable regarding changes in the hyperparameters compared to classical neural networks.
TABLE 7. Statistics on the experiments executed in the small 3 × 3 environment for all three architectures and their configurations. The best statistical values (mean, median, minimum, maximum, and standard deviation) in terms of number of training steps for each architecture are marked bold, the best overall configuration for each architecture is marked with a green background.

| Units | $\theta_{NN}$ | Solved | Mean | Median | Min | Max | Std. |
|-------|----------------|--------|------|--------|-----|-----|------|
| (8;8) | 131            | 17/20  | 29,390 | 26,800 | 18,200 | 50,000 | 9,558 |
| (16;8) | 227           | 20/20 | 21,075 | 20,500 | 13,700 | 33,200 | 4,921 |
| (16;16) | 387         | 20/20 | 16,405 | 16,800 | 10,500 | 21,900 | 3,049 |
| (32;16) | 707        | 20/20 | 14,620 | 14,650 | 8,900 | 20,000 | 2,853 |
| (32;32) | 1,283     | 20/20 | 12,625 | 12,400 | 8,700 | 17,000 | 2,143 |
| (64;32) | 2,435   | 20/20 | 10,635 | 10,250 | 7,000 | 14,100 | 2,231 |
| (64;64) | 4,611    | 19/20 | 12,790 | 10,100 | 6,700 | 50,000 | 10,200 |
| (128;64) | 8,963 | 20/20 | 11,535 | 9,700 | 6,800 | 47,600 | 8,414 |
| (128;128) | 17,411 | 20/20 | 8,855 | 8,650 | 5,800 | 11,900 | 1,746 |
| (256;128) | 34,307 | 20/20 | 7,495 | 7,550 | 5,200 | 9,800 | 1,324 |

| PQC-1 | Layers | $\theta_{PQC}$ | Solved | Mean | Median | Min | Max | Std. |
|-------|--------|----------------|--------|------|--------|-----|-----|------|
| 12    | 156    | 20/20         | 24,670 | 25,200 | 13,400 | 31,800 | 4,034 |
| 15    | 192    | 20/20         | 18,620 | 18,800 | 9,000 | 24,000 | 3,402 |
| 18    | 228    | 20/20         | 19,345 | 19,250 | 14,900 | 24,000 | 3,208 |
| 21    | 264    | 20/20         | 15,905 | 16,900 | 5,700 | 22,600 | 4,600 |
| 24    | 300    | 20/20         | 15,350 | 14,750 | 10,000 | 20,700 | 3,136 |
| 27    | 336    | 20/20         | 11,365 | 11,800 | 4,200 | 17,000 | 3,692 |
| 30    | 372    | 20/20         | 10,245 | 11,100 | 4,100 | 15,800 | 3,383 |
| 33    | 408    | 20/20         | 10,220 | 10,450 | 1,700 | 20,200 | 4,427 |
| 36    | 444    | 20/20         | 9,150  | 9,350  | 2,000 | 16,500 | 2,746 |
| 39    | 480    | 20/20         | 10,060 | 10,050 | 4,800 | 13,900 | 2,905 |

| PQC-3 | Layers | $\theta_{PQC}$ | Solved | Mean | Median | Min | Max | Std. |
|-------|--------|----------------|--------|------|--------|-----|-----|------|
| 8     | 156    | 20/20         | 20,270 | 20,000 | 14,300 | 26,300 | 1,608 |
| 10    | 192    | 20/20         | 14,930 | 16,750 | 3,900 | 22,900 | 5,231 |
| 12    | 228    | 20/20         | 16,530 | 15,850 | 7,100 | 25,000 | 4,155 |
| 14    | 264    | 20/20         | 10,995 | 11,750 | 2,600 | 22,100 | 5,188 |
| 16    | 300    | 20/20         | 9,300  | 8,790  | 1,200 | 20,600 | 4,175 |
| 18    | 336    | 20/20         | 10,385 | 9,450  | 2,600 | 19,300 | 4,553 |
| 20    | 372    | 20/20         | 10,395 | 10,400 | 1,800 | 18,000 | 4,164 |
| 22    | 408    | 20/20         | 9,490  | 8,400  | 2,300 | 20,300 | 5,181 |
| 24    | 444    | 20/20         | 6,850  | 6,650  | 1,700 | 12,800 | 3,146 |
| 26    | 480    | 20/20         | 9,515  | 8,800  | 4,400 | 15,300 | 3,347 |

Considering the best-performing PQC architectures in this work, we have to emphasize that this configuration is not to be considered efficient or even viable for current quantum hardware. The largest employed circuit using the PQC-3 architecture has almost 200 gates per qubit, not considering additional gates that could be introduced by transpiling it to a native gate set of any quantum hardware platform. Circuits with long execution times and more gates are more prone to noise on current quantum hardware. Hence, we would not expect meaningful results without substantial error mitigation efforts. Training the circuits directly on quantum hardware was also not a realistic option, given the total number of experiments we conducted and the limited availability and access to quantum computing hardware. Consequently, we limited our study to an idealized environment in a noise-free quantum circuit simulator.

VII. OUTLOOK

Understanding the characteristics of PQC is an ongoing research topic. For PQC to offer advantages over classical solutions, there are still some open questions that have to be addressed. Concerning expressiveness, the authors of [66] showed that PQC with the data-reupload technique can represent real-valued truncated Fourier series. While this could be considered a weak restriction on the expressiveness, it remains unclear if they are rich enough to approximate deep RL algorithm outputs for more complex behaviors. In [74], the authors leverage that PQC represent truncated Fourier series by showing that classical models can be obtained efficiently from trained PQC. They also report no advantage in the performance nor trainability of PQC over classical models for the problems they consider. The trainability of PQC is analyzed in more detail by Bittel et al. [75], who rigorously prove that classical training is NP-hard, and by the authors of [76], who found many sub-optimal local minima in the gradient landscape. Moreover, barren plateaus [77] are one additional hurdle for trainability. These works and our results indicate that PQC mark the beginning of quantum machine learning in general and quantum deep reinforcement learning specifically. These methods have to be developed further substantially to yield potential improvements over classical learning techniques.
Our results provide additional insight into the scaling behavior and applicability of hybrid quantum deep reinforcement learning based on PQCs, especially with regard to more demanding problems than previously considered. Our experimental setting is focused on three static environments and two different quantum circuit architectures. Furthermore, we studied the robustness of these methods in a more demanding, dynamic navigation task, although with limited scope. Hence more empirical research is needed to substantiate our findings further, and produce more conclusive results.

The second area is the applicability of quantum machine learning and quantum deep reinforcement learning in real-world applications, especially in the field of robotics. While we have demonstrated quantum deep reinforcement learning in a limited robotic scenario, actual advantages of the presented method over classical deep reinforcement learning have yet to be shown. While previous works demonstrated a quantum advantage for a certain class of problems [42] intractable for classical learning methods, it remains an open question if this advantage can be translated to problems from e.g., robotic domains.

Another crucial topic linked to real-world applications, is the encoding scheme of classical data into the quantum circuit. With the proposed methods, the required number of qubits and the operations per qubit scale linearly in the best case with the dimensionality of the state space. It will be interesting to see how different encoding techniques like e.g., amplitude encoding [78] would impact the learning behavior. Also, we limited our experiments to state spaces of small dimensionality to account for the computational demands of simulating quantum circuits on a classical computer. While this imposed no detriments on our learning scenarios, having high dimensional sensory data, e.g., high-resolution image data, is common in more complex robotic tasks. How to encode classical data with hundreds, thousands, or more dimensions efficiently onto quantum devices with their current limitations is an open question. Investigating

### TABLE 8. Statistics on the experiments executed in the medium sized $4 \times 4$ environment for all three architectures and their configurations. The best statistical values (mean, median, minimum, maximum, and standard deviation) in terms of number of training steps for each architecture are marked bold, the best overall configuration for each architecture is marked with a green background.

| Units | $|\theta_{\text{NN}}|$ | Solved | No. of training steps | Mean | Median | Min | Max | Std. |
|-------|----------------|--------|-----------------------|------|--------|-----|-----|------|
| (8,8) | 131            | 16/20  | 37,315                | 37,050 | 19,600 | 50,000 | 9,329 |
| (16,8) | 227            | 17/20  | 24,565                | 23,350 | 900    | 50,000 | 13,387 |
| (32,16) | 707            | 17/20  | 28,060                | 25,100 | 7,200  | 50,000 | 11,359 |
| (32,32) | 1,283          | 16/20  | 24,315                | 19,300 | 7,200  | 50,000 | 13,548 |
| (64,32) | 2,435          | 20/20  | 15,055                | 16,550 | 6,300  | 28,300 | 5,543 |
| (64,64) | 4,611          | 20/20  | 14,740                | 15,050 | 6,900  | 22,200 | 4,205 |
| (128,64) | 8,963         | 19/20  | 18,370                | 15,000 | 5,400  | 50,000 | 11,092 |
| (128,128) | 17,411         | 20/20  | 11,980                | 11,900 | 5,700  | 50,000 | 3,910 |
| (256,128) | 34,307       | 20/20  | 11,480                | 12,000 | 4,400  | 18,900 | 3,800 |

| PQC-1 |
|-------|--------|----------------|------------------|------|--------|-----|-----|------|
| Layers | $|\theta_{\text{PQC}}|$ | Solved | No. of training steps | Mean | Median | Min | Max | Std. |
| 12     | 156    | 18/20  | 41,395              | 42,400 | 28,700 | 50,000 | 5,260 |
| 15     | 192    | 20/20  | 29,905              | 31,700 | 9,300  | 46,100 | 9,254 |
| 18     | 228    | 20/20  | 27,055              | 30,950 | 2,400  | 39,500 | 11,105 |
| 21     | 264    | 20/20  | 24,815              | 26,750 | 4,000  | 36,000 | 8,303 |
| 24     | 300    | 20/20  | 27,795              | 27,250 | 22,200 | 36,000 | 3,913 |
| 27     | 336    | 20/20  | 22,335              | 24,400 | 11,800 | 31,700 | 6,131 |
| 30     | 372    | 20/20  | 20,560              | 21,600 | 4,800  | 30,700 | 6,542 |
| 33     | 408    | 20/20  | 20,135              | 20,750 | 4,400  | 29,900 | 5,640 |
| 36     | 444    | 20/20  | 17,705              | 18,800 | 7,000  | 26,000 | 4,678 |
| 39     | 480    | 20/20  | 16,880              | 18,600 | 3,400  | 29,300 | 7,288 |

| PQC-3 |
|-------|--------|----------------|------------------|------|--------|-----|-----|------|
| Layers | $|\theta_{\text{PQC}}|$ | Solved | No. of training steps | Mean | Median | Min | Max | Std. |
| 8      | 156    | 20/20  | 30,155              | 32,900 | 8,100  | 41,300 | 9,654 |
| 10     | 192    | 20/20  | 25,990              | 28,700 | 8,800  | 39,900 | 8,469 |
| 12     | 228    | 20/20  | 21,525              | 23,800 | 3,500  | 31,900 | 8,074 |
| 14     | 264    | 20/20  | 19,695              | 20,750 | 2,700  | 29,900 | 7,103 |
| 16     | 300    | 20/20  | 20,170              | 21,950 | 4,300  | 28,300 | 5,984 |
| 18     | 336    | 20/20  | 19,015              | 20,950 | 4,400  | 26,700 | 5,891 |
| 20     | 372    | 20/20  | 18,795              | 19,700 | 6,000  | 24,500 | 4,346 |
| 22     | 408    | 20/20  | 17,065              | 17,130 | 5,500  | 26,600 | 5,652 |
| 24     | 444    | 20/20  | 14,665              | 15,350 | 2,000  | 25,000 | 6,068 |
| 26     | 480    | 20/20  | 15,875              | 15,950 | 7,200  | 25,200 | 5,164 |
H. Hohenfeld et al.: Quantum Deep Reinforcement Learning for Robot Navigation Tasks

| TABLE 9. Statistics on the experiments executed in the large 5 × 5 environment for all three architectures and their configurations. The best statistical values (mean, median, minimum, maximum, and standard deviation) in terms of number of training steps for each architecture are marked bold, the best overall configuration for each architecture is marked with a green background. Configurations for which fewer than 15 runs succeeded are considered insufficient and are marked with a red background. |

| Units | $|\theta_{\text{NN}}|$ | Solved | No. of training steps |
|-------|----------------|--------|----------------------|
|       |                 |        | Mean | Median | Min | Max | Std. |
| (8;8) | 131             | 1/20   | 49,825 | 50,000 | 46,500 | 50,000 | 762 |
| (16;8) | 227            | 4/20   | 49,515 | 50,000 | 42,500 | 50,000 | 1,641 |
| (16;16) | 387        | 8/20   | 46,570 | 50,000 | 32,400 | 50,000 | 5,104 |
| (32;16) | 707         | 15/20  | 42,040 | 41,350 | 29,700 | 50,000 | 6,291 |
| (32;32) | 1,283        | 17/20  | 40,055 | 40,900 | 23,300 | 50,000 | 8,465 |
| (64;32) | 2,435        | 18/20  | 32,135 | 30,850 | 22,000 | 50,000 | 7,845 |
| (64;64) | 4,611        | 20/20  | 29,475 | 28,450 | 23,300 | 40,400 | 4,218 |
| (128;64) | 8,963       | 20/20  | 25,540 | 24,950 | 19,500 | 31,600 | 3,132 |
| (128;128) | 17,411     | 20/20  | 22,870 | 22,100 | 15,900 | 35,900 | 4,301 |
| (256;128) | 34,307      | 20/20  | 22,220 | 21,100 | 17,700 | 33,500 | 4,017 |

| PQC-1 | No. of training steps |
|-------|----------------------|
| Layers | $|\theta_{\text{PQC}}|$ | Solved | No. of training steps |
|       |                     |        | Mean | Median | Min | Max | Std. |
| 12     | 156                  | 4/20   | 48,005 | 50,000 | 31,400 | 50,000 | 4,644 |
| 15     | 192                  | 12/20  | 46,390 | 47,700 | 36,500 | 50,000 | 4,021 |
| 18     | 228                  | 18/20  | 40,410 | 39,830 | 25,900 | 50,000 | 6,182 |
| 21     | 264                  | 20/20  | 37,375 | 34,300 | 29,400 | 49,700 | 6,546 |
| 24     | 300                  | 20/20  | 35,190 | 33,600 | 27,400 | 47,200 | 5,216 |
| 27     | 336                  | 19/20  | 35,685 | 32,900 | 19,300 | 50,000 | 7,881 |
| 30     | 372                  | 20/20  | 30,995 | 31,400 | 8,500  | 46,600 | 9,123 |
| 33     | 408                  | 20/20  | 28,155 | 27,900 | 19,100 | 38,500 | 5,752 |
| 36     | 444                  | 20/20  | 29,260 | 29,300 | 15,800 | 47,000 | 6,614 |
| 39     | 480                  | 20/20  | 25,110 | 26,550 | 12,900 | 33,500 | 5,309 |

| PQC-3 | No. of training steps |
|-------|----------------------|
| Layers | $|\theta_{\text{PQC}}|$ | Solved | No. of training steps |
|       |                     |        | Mean | Median | Min | Max | Std. |
| 8      | 156                  | 13/20  | 43,925 | 45,700 | 32,100 | 50,000 | 6,303 |
| 10     | 192                  | 18/20  | 41,645 | 42,500 | 28,800 | 50,000 | 6,744 |
| 12     | 228                  | 20/20  | 35,675 | 35,000 | 27,100 | 47,600 | 6,372 |
| 14     | 264                  | 20/20  | 33,885 | 33,400 | 22,400 | 45,200 | 5,778 |
| 16     | 300                  | 20/20  | 32,625 | 33,700 | 23,000 | 43,300 | 5,588 |
| 18     | 336                  | 20/20  | 30,765 | 30,300 | 22,600 | 42,200 | 5,212 |
| 20     | 372                  | 20/20  | 29,895 | 31,250 | 20,500 | 39,300 | 5,037 |
| 22     | 408                  | 20/20  | 24,530 | 24,900 | 12,800 | 38,600 | 5,762 |
| 24     | 444                  | 20/20  | 27,575 | 28,650 | 9,400  | 38,800 | 7,344 |
| 26     | 480                  | 20/20  | 23,635 | 21,800 | 9,700  | 41,600 | 7,535 |

We understand our work as a contribution toward application-focused empirical research on quantum algorithms in a robotic context. We see this as a viable route to accelerate the development and understanding of quantum algorithms, quantum machine learning, and the application of quantum techniques in deep reinforcement learning. Looking forward, we believe that quantum algorithms, together with future hardware developments in the field of quantum computing, will contribute to the advancement of autonomous robotics.

APPENDIX A

COMPARISON TO BASELINE ENVIRONMENTS

We use our learning setup to solve the benchmark OpenAI gym [19] environments FrozenLake and CartPole-v1. This way, we underline our argument that the navigation tasks are indeed more complex and difficult to learn.

To learn the FrozenLake environment, we use binary encoding for the state features, adapt the circuit to four qubits,
and adapt the parameters for epsilon decay and max steps per episode. All other learning hyper-parameters are unchanged. We also use the arctangent activation function and trainable parameters on the input features as well as four trainable output parameters. A full list of the hyper-parameters is given in Table 6 in App. B. The left plot of Fig. 12 shows that 20 runs with a classical neuronal networks with (128, 64) hidden units learn an optimal policy in fewer than 1,250 training steps with a mean of 510 steps, a median of 513 steps, and a standard deviation of 204 steps. The classical architecture takes roughly 20 times as long to learn an optimal policy in our simplest navigation task. Similar results hold for PQC with one input encoding and 15 layers (PQC-1-15). Here, the training finishes on average in 593 steps, with a median of 613 steps and a standard deviation of 205 steps. This result is in alignment (slightly better) with [41]. We want to emphasize that we did not fine-tune the hyper-parameters for the FrozenLake environment but were still able to learn the task much faster than for our 3 × 3 navigation environment.

We obtained similar results for the Cartpole-v1 environment as depicted in the right plot of Fig. 12. For this environment, we adapted the PQC to 4 qubits and used the measurements $\sigma^z_1, \sigma^z_2$ and $\sigma^z_3, \sigma^z_4$ for the post-processing. We adapted the epsilon decay parameters and other hyper-parameters slightly, as shown in Table 6. Averaged over 20 runs, the classical network with (256, 128) hidden units is able to solve Cartpole-v1 with an average of 3,645 training steps (median: 2,350, standard deviation: 2,600) with slightly adapted hyper-parameters. That is approximately twice as fast as the same network architecture learns the 3 × 3 navigation environment. For the PQC, the configuration with one input encoding and five layers needed fewer than 10,000 training steps to learn the optimal policy, which is notably faster than reported in the literature (e.g., [41] for Cartpole-v0). The PQC-1-5 ansatz solves Cartpole-v0 in a similar time (mean: 4065, median: 4050, standard deviation: 1933) and thus solves it faster than larger PQC-1 configurations solve the 3 × 3 navigation environment.

Hence, we conclude that our navigation tasks are considerably more challenging than FrozenLake and Cartpole-v1 for the (hybrid quantum) DDQN algorithm.

**APPENDIX B**

**HYPER-PARAMETERS FOR EXPERIMENTS**

The hyper-parameters used in all environments and learning setups are outlined in Table 6.

**APPENDIX C**

**RESULT DETAILS**

Detailed statistics over all conducted experiments are reported in Tables 7, 8 and 9.

**ACKNOWLEDGMENT**

The authors would like to thank Elie Mounzer, Gunnar Schönhoff, Patrick Draheim, Melvin Laux, and Alexander Fabisch for their valuable feedback on this manuscript and also would like to thank the anonymous reviewers for their constructive remarks and recommendations to improve this article.

**REFERENCES**

[1] S. Gu, E. Holly, T. Lillicrap, and S. Levine, “Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates,” in *Proc. IEEE Conf. Robot. Autom.* (ICRA), May 2017, pp. 3389–3396.

[2] A. R. Mahmood, D. Korenkeyvych, G. Vasan, W. Ma, and J. Bergstra, “Benchmarking reinforcement learning algorithms on real-world robots,” in *Proc. 2nd Conf. Robot. Learn.*, 2018, pp. 561–591.

[3] D. Kalashnikov, A. Ipan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, “Scalable deep reinforcement learning for vision-based robotic manipulation,” in *Proc. Int. Conf. Robot. Learn.*, 2018, pp. 651–657.

[4] M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, “Learning dexterous in-hand manipulation,” *Int. J. Robot. Res.*, vol. 39, no. 1, pp. 3–20, 2020.

[5] A. Montanaro, “Quantum algorithms: An overview,” *NPJ Quantum Inf.*, vol. 1, no. 1, Jan. 2016.

[6] P. W. Shor, “Algorithms for quantum computation: Discrete logarithms and factoring,” in *Proc. 35th Annu. Symp. Found. Comput. Sci.*, 1994, pp. 124–134.

[7] L. K. Glover, “A fast quantum mechanical algorithm for database search,” in *Proc. 28th Annu. ACM Symp. Theor. Comput. (STOC)*, New York, NY, USA, 1996, pp. 212–217.

[8] A. W. Harrow, A. Hassidim, and S. Lloyd, “Quantum algorithm for linear systems of equations,” *Phys. Rev. Lett.*, vol. 103, no. 15, Oct. 2009, Art. no. 150502.

[9] L. Alchieri, D. Badalotti, P. Bonardi, and S. Bianco, “An introduction to quantum machine learning: From quantum logic to quantum deep learning,” *Quantum Mach. Intell.*, vol. 3, no. 2, pp. 1–30, Dec. 2021.

[10] F. Kirchner, “AI-perspectives: The Turing option,” *AI Perspect.*, vol. 2, no. 1, pp. 1–12, Dec. 2020.

[11] P. Benioff, *Quantum Computers and Quantum Computers*, 1997.

[12] P. Benioff, “Quantum robots and environments,” *Phys. Rev. A. Gen. Phys.*, vol. 58, no. 2, pp. 893–904, Aug. 1998.

[13] P. Benioff, “Space searches with a quantum robot,” 2000, arXiv:quant-ph/0003006.

[14] P. Benioff, “Quantum robots plus environments,” in *Quantum Communication, Computing, and Measurement 2*, Springer, 2002, pp. 3–9.

[15] J. Preskill, “Quantum computing in the NISQ era and beyond,” *Quantum*, vol. 2, p. 79, Aug. 2018.

[16] M. Brooks, “Beyond quantum supremacy: The hunt for useful quantum computers,” *Nature*, vol. 574, no. 7776, pp. 19–21, Oct. 2019.

[17] E. Farhi and H. Neven, “Classification with quantum neural networks on near term processors,” 2018, arXiv:1802.06002.

[18] M. Benedetti, E. Lloyd, S. Sack, and M. Fiorentini, “Parameterized quantum circuits as machine learning models,” *Quantum Sci. Technol.*, vol. 4, no. 4, Nov. 2019, Art. no. 045001.

[19] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “OpenAI gym,” 2016, arXiv:1606.01540.

[20] N. Meyer, C. Uferecht, M. Periyasamy, D. D. Scherer, A. Plinge, and C. Mutschler, “A survey on quantum reinforcement learning,” 2022, arXiv:2202.03464.

[21] N. M. P. Neumann, P. B. U. L. de Heer, and F. Phillipson, “Quantum computation, Computing, and Measurement 2”, Springer, 2002, pp. 3–9.

[22] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction* (Adaptive Computation and Machine Learning Series). Cambridge, MA, USA: MIT Press, 2018.

[23] D. Dong, C. Chen, J. Chu, and T.-J. Tarn, “Robust quantum-inspired reinforcement learning for robot navigation,” *IEEE/ASME Trans. Mechatronics*, vol. 17, no. 1, pp. 86–97, Feb. 2012.

[24] Y. Li, A. H. Aghvami, and D. Dong, “Intelligent trajectory planning in UAV-mounted wireless networks: A quantum-inspired reinforcement learning perspective,” *IEEE Wireless Commun. Lett.*, vol. 10, no. 9, pp. 1994–1998, Sep. 2021.

[25] C. J. C. H. Watkins, “Learning from delayed rewards,” Ph.D. thesis, King’s College, Cambridge, U.K., 1989.
G. D. Paparo, V. Dunjko, A. Makmal, M. A. Martin-Delgado, and T. Sriarunothai, S. Wölk, G. S. Giri, N. Friis, V. Dunjko, H. J. Briegel, H. J. Briegel and G. De las Cuevas, “Projective simulation for artificial intelligence,” in Proc. IEEE Int. Conf. Syst., Man. Cybern. (SMC), Oct. 2017, pp. 282–287.

V. Saggio, B. E. Asenbeck, A. Hamann, T. Strömberg, P. Schiansky, V. Dunjko, J. M. Taylor, and H. J. Briegel, “Quantum-enhanced machine learning,” Proc. Workshop Quantum and Renewable Energy, Vol. 55, 2021.

A. Hamann and S. Wölk, “Performance analysis of a hybrid agent for quantum-accessible reinforcement learning,” New J. Phys., vol. 24, no. 3, Mar. 2022, Art. no. 033044.

D. Nagy, Z. Tabi, Z. Kallus, and Z. Zimborás, “Photonic quantum policy learning in OpenAI gym,” in Proc. IEEE Int. Conf. Quantum Comput. Eng. (QCE), Oct. 2021, pp. 123–129.

Q. Lan, “Variational quantum soft-actor-critic,” 2022, arXiv:2112.11921.

E. A. Cherrat, I. Kerenidis, and A. Prakash, “Quantum reinforcement learning via policy iteration,” 2022, arXiv:2203.01889.

V. Muih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, Adam Trask, I. Antonoglou, H. King, D. Kumaran, D. War Issue, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, pp. 529–533, 2014.

J.-Y. Hsiao, Y. Du, W.-Y. Chiang, M.-H. Hsieh, and H.-S. Goan, “Untangled quantum reinforcement learning agents in the OpenAI gym,” 2022, arXiv:2203.14348.
HANS HOHENFELD received the B.Sc. and M.Sc. degrees in computer science from FernUniversität in Hagen, in 2014 and 2017, respectively. He is currently pursuing the Ph.D. degree with the University of Bremen. Before joining the Robotics Research Group, University of Bremen, in 2017, he has been a Software Engineer, since 2008. His research interests include hardware efficient quantum algorithms for machine learning and quantum information and quantum algorithms in learning and decision problems for autonomous agents.

DIRK HEIMANN received the B.Sc. and M.Sc. degrees in physics from Leibniz University Hannover, in 2014 and 2017, respectively. He is currently pursuing the Ph.D. degree with the University of Bremen. Before joining the Robotics Research Group, University of Bremen, in 2020, he was a Software Consultant in the industry. His research interests include quantum algorithms for machine learning and quantum optimal control.

FELIX WIEBE received the B.Sc. and M.Sc. degrees in physics from Georg-August University Göttingen, Göttingen, in 2014 and 2018, respectively. He is currently pursuing the Ph.D. degree in optimal control for underactuated robots. In 2018, he joined the Mechanics and Control Team, German Research Center for Artificial Intelligence (DFKI), and Robotics Innovation Center (RIC), Bremen. His research interests include robotics, optimal control, model predictive control, optimization algorithms, and quantum optimal control.

FRANK KIRCHNER received the degree in computer science and neurobiology and the Dr.rer.nat. degree in computer science from the University of Bonn, in 1994 and 1999, respectively. Since 1994, he has been a Senior Scientist with the Gesellschaft für Mathematik und Datenverarbeitung (GMD), Sankt Augustin. He has been a Senior Scientist with the Department for Electrical Engineering, Northeastern University, Boston, MA, USA, since 1998, where he was appointed as an Adjunct Professor and then a Tenure Track Assistant Professor, in 1999. Since 2002, he has been a Full Professor with the University of Bremen. Since December 2005, he has been the Director of the Robotics Innovation Center (RIC), Bremen.