2018 IEEE International Conference on Robotics and Automation (ICRA)
May 21-25, 2018, Brisbane, Australia

Reinforcement Learning of Depth Stabilization with a Micro Diving Agent

Gerrit Brinkmann[1], Wallace M. Bessa[2], Daniel-A. Duecker[1], Edwin Kreuzer[1], and Eugen Solowjow[1,3]

Abstract— **Reinforcement learning (RL) allows robots to solve control tasks through interaction with their environment. In this paper we study a model-based value-function RL approach, which is suitable for computationally limited robots and light embedded systems. We develop a diving agent, which uses the RL algorithm for underwater depth stabilization. Simulations and experiments with the micro diving agent demonstrate its ability to learn the depth stabilization task.**

I. INTRODUCTION

Most robot control problems are solved with approaches based on feedback control theory, e. g. the PID regulator. However, for autonomy it is crucial that robots become capable in solving control tasks on their own. Reinforcement learning (RL) is a principled framework that allows agents to learn behaviours through interactions with the environment. In this work we study RL for performing a low-level control task on a highly embedded micro underwater robot.

The core idea of RL is to provide the robot with a high-level specification of what to do instead of how to do it. Reinforcement learning can be distinguished in value-function based methods and policy search [1]. In policy search robots learn a direct mapping from states to actions. In value-function-based approaches robots learn a value function, an intermediate structure, and derive actions from the value function. Both, policy search and value-function-based approaches can either be model-based or model-free. Model-free methods do not consider the dynamics of the world. Model-based methods incorporate a model of the world dynamics, which is learnt from data. While model-free methods are in general computationally cheaper, they require more data to solve the task.

Early work on reinforcement learning mainly focussed on value-function-based algorithms [2], [3]. However, most of the recent RL successes have been achieved with policy-search methods, e. g. acrobatics with helicopters [4] or controlling an inverted pendulum [5]. Value-function based RL methods have been successfully applied to e. g. control problems in robot-soccer [6], [7], [8]. It should be noted that the recent combination of deep neural nets and RL provides novel methods coined deep RL [9]. However, applications of deep RL are still not feasible on the limited hardware of micro robots. While RL with sufficient computational power has been proven feasible, the question remains to what extend computationally limited robots can perform RL.

Micro underwater agents have became a popular research topic in recent years [10], [11]. They can be deployed for oceanographic and industrial monitoring. An important problem in underwater robotics is depth stabilization. The problem resembles the inverted pendulum problem, because in both cases the desired state is an unstable equilibrium. The depth stabilization problem can be efficiently solved with feedback control [12]. However, this contribution explores if depth stabilization can also be achieved through RL. Unlike in aerial robotics [13], only few studies exist where RL has been applied to underwater robots, e. g. [14] or [15]. The task of depth stabilization has all challenges of the underwater domain, including hydrodynamic effects and slow actuator dynamics, while being simple enough for studying RL feasibility.

The contributions of this paper are two-fold. First, we introduce a value-function based RL algorithm for learning depth stabilization. The learning algorithm is studied in simulations and experiments. It is suitable for highly embedded micro robots with limited computational resources and memory. Second, we present the design of a micro diving agent, which can be used for studying a variety of control and learning problems in the underwater setting. It is an improved version of the diving agent presented in [12].

The remainder of this paper is structured as follows. Section II introduces the problem of depth stabilization. Section III describes the RL algorithm for learning depth stabilization. In Section IV the micro diving agent is introduced, which serves as a hardware testbed. Simulation and experimental results are presented in Section V. Section VI draws conclusions and presents and outlook.

II. PROBLEM STATEMENT

Consider a submerged diving agent whose state is defined by its depth z and velocity $\dot{z}$ as illustrated in Fig. 1. Figure 1 shows the forces acting on a dive agent of mass m_d. The forces are the gravitational force $F_{\mathrm{G}} = m_{\mathrm{d}}\, g$, the buoyancy force F_{B}, and the hydrodynamic force F_{D}, which consist of damping and added mass. The diving agent has a fixed volume $V_{\mathrm{d,0}}$ and an adjustable volume $V_{\mathrm{d,a}}$, which is used

Research supported by the German Research Foundation (DFG) under grant 250508151 (Kr 752/33-1). The work of W. M. Bessa was supported by the Alexander von Humboldt Foundation and the Brazilian Coordination for the Improvement of Higher Education Personnel.

[1]Authors are with the Institute of Mechanics and Ocean Engineering, Hamburg University of Technology, Germany. [2]Author is with the Department of Mechanical Engineering, Federal University of Rio Grande do Norte, Brazil. [3]Author is with Siemens Corporate Technology, Berkeley, CA, USA.

*Authors are listed in alphabetical order. gerrit.brinkmann@tuhh.de, wmbessa@ct.ufrn.br, daniel.duecker@tuhh.de, kreuzer@tuhh.de, eugen.solowjow@siemens.com.

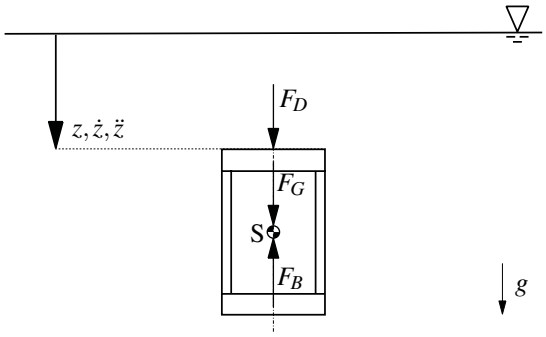


Fig. 1: Forces acting on the diving agent.

for actuation. The buoyancy force F_B can be computed as

$$F_\mathrm{B} = \rho_\mathrm{w}\, g\, (V_{\mathrm{d,0}} + V_{\mathrm{d,a}})\,, \tag{1}$$

where ρ_W is the density of water.

The goal of depth stabilization is to control $V_{\mathrm{d,a}}$ to maintain a desired depth z_d. We want to solve this control problem through reinforcement learning without prior knowledge of any system or actuator dynamics. The challenges are due to the unknown hydrodynamic effects and the slow actuator dynamics. The time constant of volume adjustance is in general similar to the system dynamics. Also, the adjustable volume is small compared to the fixed volume and hence the maximum control force is also small and constrained.

III. REINFORCEMENT LEARNING FRAMEWORK

This section describes the RL algorithm, which is deployed for training the diving agent to learn depth stabilization.

We treat the depth stabilization problem as a standard Markov decision process (MDP) consisting of states S, actions A, a reward function $R(s,a)$ and the state transition probability matrix $\boldsymbol{P}(s' \,|\, s,a)$. By taking an action $a \in A$ in state $s \in S$ the agent transitions to a new state s' according to $\boldsymbol{P}(s' \,|\, s,a)$ and receives a reward. The value of a state s is determined by the value function $v(s)$ while the value of a state-action pair is determined by the state-action-value function $q(a,s)$. By iterating over the Bellmann equation the optimal action-value function $q^*(a,s)$ can be obtained. The optimal policy $\pi(s)$ is then $\pi(s) = \arg\min_a q^*(a,s)$.

Our framework for learning depth stabilization is derived from model-based dynamic programming (DP) with initially unknown model and with generalized policy iteration (GPI). The value function is evaluated during the operation of the agent. After each episode parameters are updated and policy evaluation is performed. We refer to [2] for a concise introduction.

1) Reinforcement Learning Task: The state-space S consists of agent depth z and velocity $\dot{z}$. Figure 2 shows the discretized state-space. A zone around the target depth z_d, bounded by z_Tol, defines the learning region, which in this contribution always consists of 101 states. A finer discretization is not feasible due to the limited hardware of the micro robot deployed in the experiments. Every position outside of this area is considered to be the terminal state s_T. Reaching s_T leads to the termination of a learning episode and the start of a new episode.

Algorithm 1 Reinforcement Learning Algorithm

```
procedure RLALGORITHM
    while true do
        s =getState(z, ż)
        a =selectNextAction(s, v_k(s), P(s'|s,a), R(s,a))
        perform action a
        s' = getState(z, ż)
        get reward R
        updateSystemDynamicsModel(R, s, s', C(s,s',a),
    R(s,a), R_c(s,a))
        if s' = s_T then
            policyIteration(v_k(s), P(s'|s,a), R(s,a))
            re-initiate next episode
        end if
    end while
```

The action space A consists of two actions. Action a_1 reduces the adjustable volume $V_{\mathrm{d,a}}$ and hence the buyoncy force. Action a_2 increases the adjustable volume. The agent receives a negative reward $R(s',a)$, which punishes its current distance from the desired depth z_d:

$$R(s',a) = -|z - z_\mathrm{d}|\,. \tag{2}$$

Furthermore, a high negative reward is given for reaching the terminal state.

2) Learning Algorithm: The overall algorithm for learning the depth stabilization task is depicted in Alg. 1. Indices k and t refer to the k-th learning episode and the t-th timestep, respectively.

The algorithm runs in a loop with typical loop periods between 0.02 s and 0.2 s. At the beginning of each loop, the agent evaluates its current state s_t. Then, it selects an action a_t, which is executed. The action selection is based on the state s_t, the transition probability matrix $\boldsymbol{P}(s' \,|\, s,a)$ and the value functions $v(s)$. While performing a_t, the dive agent may change its state. Therefore, z and $\dot{z}$ are analyzed again to capture the new state s'_{t+1}. Additionally, a reward R_{t+1} is provided.

For updating the dynamics model, the agent keeps track of all transitions from s to s'. If s' is the terminal state of the system, $v(s)$, $\boldsymbol{R}(s,a)$, and $\boldsymbol{P}(s' \,|\, s,a)$ are updated and the next episode is initialized.

3) Action Selection: In each time step the agent selects an action. The decision is based on the state s of the system, as well as the dynamics model $\boldsymbol{P}(s' \,|\, s,a)$ and the reward matrix $\boldsymbol{R}(s,a)$. Algorithm 2 selects the action.

First, the action-value matrix $\boldsymbol{Q}(s,a)$ for both actions a_1 and a_2 is computed, whereby γ is the discount factor. Note here that instead of using separate rewards for every combination of s and s', the rewards are averaged over the successor states. The action a is determined by comparing the action-values $\boldsymbol{Q}(s,a_1)$ and $\boldsymbol{Q}(s,a_2)$. If the agent would strictly follow the computed action, it would receive the

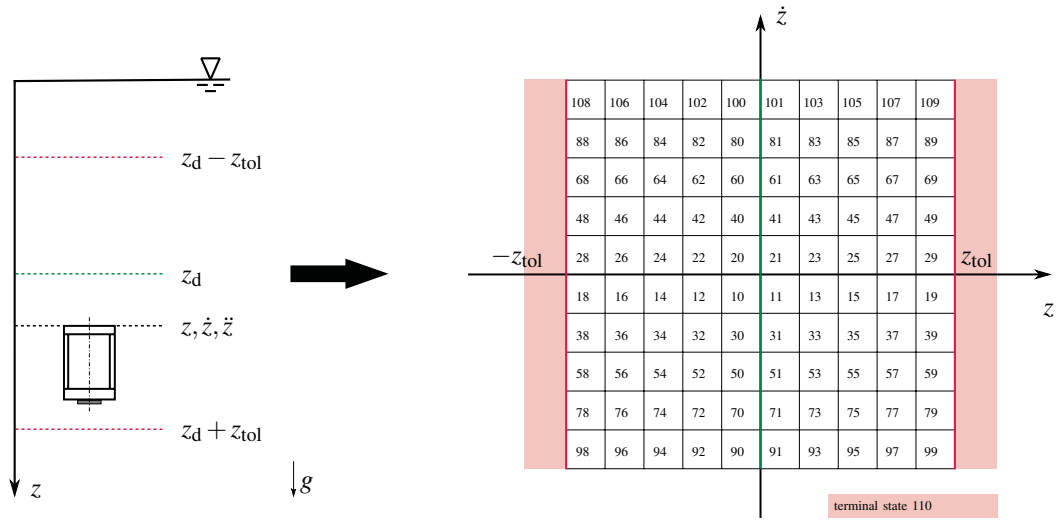


Fig. 2: State-space discretization in the learning region of the RL task. The state-space consists of 101 viable states and terminal states that are outside of the learning region.

Algorithm 2 Action Selection

procedure SELECTNEXTACTION(s, $v_k(s)$, $\boldsymbol{P}(s'\,|\,s,a)$, $\boldsymbol{R}(s,a)$)
  $\boldsymbol{Q}(s,a) = \boldsymbol{P}(s'\,|\,s,a)\,(\boldsymbol{R}(s,a) + \gamma \cdot v_k(s)),\ \forall\, a \in a$
  $p_{\text{rand}} :=$ random probability
  if $\boldsymbol{Q}(s,a_1) > \boldsymbol{Q}(s,a_2)$ **then**
    $a \leftarrow a_1$
  else
    $a \leftarrow a_2$
  end if
  if $p_{\text{rand}} > p_{\text{expl}}$ **then**
    act against action selection
  end if
return a

Algorithm 3 System Dynamics Model

procedure UPDATESYSTEMDYNAMICSMODEL(r, s, s', $\boldsymbol{C}(s,s',a)$, $\boldsymbol{R}(s,a)$, $\boldsymbol{R}_c(s,a)$)
  if $a = a_1$ **then**
    $\boldsymbol{C}(s,s',a_1) \leftarrow \boldsymbol{C}(s,s',a_1) + 1$
    $\boldsymbol{R}_c(s',a_1) \leftarrow \boldsymbol{R}_c(s',a_1) + r$
  else
    $\boldsymbol{C}(s,s',a_2) \leftarrow \boldsymbol{C}(s,s',a_2) + 1$
    $\boldsymbol{R}_c(s',a_2) \leftarrow \boldsymbol{R}_c(s',a_2) + r$
  end if
  if $s = s_T$ **then**
    $\boldsymbol{C}(s,a) \leftarrow \sum_{s'} \boldsymbol{C}(s,s',a)\ \forall\, a \in A,\ s \in S$
    $\boldsymbol{P}(s'\,|\,s,a) \leftarrow \boldsymbol{C}(s,s',a)\,/\,\boldsymbol{C}(s,a)\ \forall\, a \in A,\ s \in S$
    $\boldsymbol{R}(s',a) \leftarrow \boldsymbol{R}_c(s',a)$
    return $\boldsymbol{P}(s'\,|\,s,a), \boldsymbol{R}(s',a)$
  end if

highest returns according to its current dynamics belief. In order to explore the state-space, the action selection is altered with a certain probability p_{expl}. This allows to update $\boldsymbol{P}(s'\,|\,s,a)$ with new state transition counts.

4) Dynamics Model: The state transition probability $\boldsymbol{P}(s'\,|\,s,a)$ represents the dynamics model. Since the agent has no initial knowledge of its dynamics model, it has to learn $\boldsymbol{P}(s'\,|\,s,a)$ on-the-fly as well. The algorithm for learning the dynamics model is shown in Alg. 3.

After performing an action, the agent evaluates its new state s'. Thereafter, it stores the state transition with respect to the action in the transition count matrix $\boldsymbol{C}(s,s',a)$. Additionally, the received reward is averaged and stored in the reward count matrix $\boldsymbol{R}_c(s,a)$. The matrix differs from $\boldsymbol{R}(s,a)$ such that it holds the total averaged rewards from overall learning, while $\boldsymbol{R}(s,a)$ only holds the rewards of the current episode.

If the successor state is the terminal state s_T, all transitions stored in $\boldsymbol{C}(s,s',a)$ are evaluated with respect to a. Afterwards, $\boldsymbol{P}(s'\,|\,s,a)$ is updated by computing the probability of all state transitions initiated from s under action a. Besides updating the transition probability matrix, the reward matrix $\boldsymbol{R}(s,a)$ is updated by $\boldsymbol{R}_c(s,a)$.

5) Policy Iteration: An important part of the DP algorithm is GPI, for which the value function is evaluated and the policy is improved. After the termination of episode k, policy iteration is used to calculate an improved value function v_{k+1} with $\boldsymbol{P}(s'\,|\,s,a)$ and $\boldsymbol{R}(s,a)$. This value function is then used during the subsequent episode. The steps of this update are summarized in Alg. 4.

In order to improve the value function after the k-th episode, the agent computes the action value matrix $\boldsymbol{Q}(s,a)$ of the system with the current value function v_k. By choosing the maximal action value for each state, a successor value function $\hat{v}_k$ is determined. Due to the improved transition matrix, the values of possible successor states are backed up

more precisely to the value of each state, leading to a better approximation of an optimal value function. An accurate state transition model leads to fast convergence to an optimal policy.

In order to obtain a better approximation of the true state value, the successor value function is used in a new backup iteration as v_k. This step is repeated until the value function converges under the given transition and reward matrices. The iteration is terminated if the difference between v_k and $\hat{v}_k$ is smaller than a predefined tolerance Δv_{Tol}. Afterwards, the value of the state is updated according to v_{k+1}, which is then used for the action selection process of the next episode $k+1$.

Algorithm 4 Policy Iteration

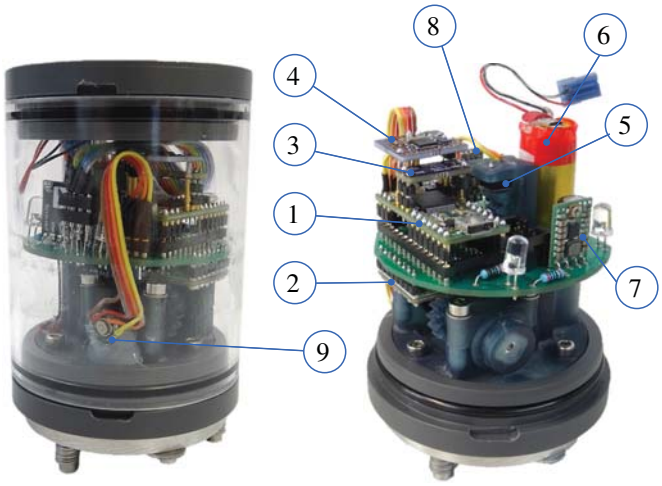
procedure POLICYITERATION($v_k(s)$, $\boldsymbol{P}(s'|s,a)$,$\boldsymbol{R}(s,a)$)
  while $\Delta v_{\max} > \Delta v_{Tol}$ **do**
    $v'_k(s,a) \leftarrow \boldsymbol{P}(s'|s,a)\, v_k(s) \;\forall\; a \in A$
    $\boldsymbol{Q}(s,a) = \boldsymbol{P}(s'|s,a)\,(\boldsymbol{R}(s,a) + \gamma v'_k(s)) \;\forall\; a \in A$
    $\hat{v}_k(s) \leftarrow \max_{\forall s \in S}(\boldsymbol{Q}(s,a_1), \boldsymbol{Q}(s,a_2))$
    $\Delta v_{\max} \leftarrow \max_{s}(\hat{v}_k(s) - v_k(s))$
    $v_k(s) \leftarrow \hat{v}_k(s)$
  end while
  $v_{k+1}(s) \leftarrow v_k(s)$
return $v_{k+1}(s)$

IV. MICRO DIVING AGENT DESIGN

For experimental evaluation of the presented RL algorithm a micro diving agent has been designed. The diving agent is a revised and improved version of the robot introduced in [12]. The diving agent measures the surrounding pressure in order to change its volume by means of a roller diaphragm. To make this contribution self-sufficient, we briefly introduce the design and main specs of the diving agent in what follows.

The diving agent is shown in Fig. 3. Its diameter is 70 mm and it weighs approx. 200 g It consists of a Teensy 3.2 microcontroller board ①, an MS5803-01BA pressure sensor (not visible), an external EEPROM board with two M24M02-DR EEPROM chips (512 kB total) ③, an ESP8266 wifi module ④, and a shortstroke button ⑤. The dive agent is powered by a 3.7 V lithium polymer battery with 1 200 mAh capacity ⑥, which is used with an adjustable S7V8A step-up/step-down voltage regulator ⑦, and a Micro BEC ⑧. The drive unit consists of a DRV8334 dual motor driver carrier ②, powering a Faulhaber AM1020-V-3-16 stepper motor combined with a planetary gear (256:1) ⑨, a roller diaphragm, and a self-designed 3d-printed gear/gear rack combination. The total adjustable volume amounts to 3.7% of the fixed volume.

The diving agent can be operated at up to 10 m water depth. The operational time on a single charge is 2 hours.



a) Diving Cell b) Bottom Part

Fig. 3: Illustration of the diving agent components.

V. RESULTS

This section presents results on underwater depth stabilization through reinforcement learning. Simulations and experiments are performed and analyzed.

A. Simulations

A simulation environment was set up to study the RL algorithm. The equation of motion of the diving agent are solved within the simulation environment considering actuator dynamics and contraints. For all simulated learning scenarios the boundaries of the state-space according to Fig. 2 are set to $z_{\text{Tol}} = 0.1\,\text{m}$ and $\dot{z}_{\text{Tol}} = 0.04\,\frac{\text{m}}{\text{s}}$.

In order to obtain a farsighted agent, the discount factor is set to $\gamma = 0.99$. The termination criterium for policy iteration is set to $\Delta = 0.0001$. Policy enforcement, which punishes the agent if it leaves the learning area, is ensured by allocating a negative reward of $\boldsymbol{R}(s_{\text{T}},) = -10$. Moreover, a linearly decreasing exploration probability defined by $p_{\text{expl}}(e=0) = 10\,\%$ and $p_{\text{expl}}(e=1000) = 0\,\%$ is chosen for the exploration of the state space. After the termination of an episode, the next attempt is initialized at a random position of the learning area. If the agent is able to remain in the learning area for 300 s, we consider the stabilization as successfull and the episode is terminated.

In order to evaluate the learning performance we analyze the duration during which the agent remains within the desired boundaries. We illustrate this time as a function over the completed episodes and call it learning curve (LC). An exemplary LC is depicted in Fig. (4). The learning task succeeds for the first time in episode 190 and succeeds every time from episode 400 on. A so called average learning curve (ALC) is obtained by calculating the moving average over 50 episodes of the LC. The ALC is illustrated in Fig. (4) as well.

In order to obtain statistical values of the algorithm performance we repeat the learning task 100 times, whereby

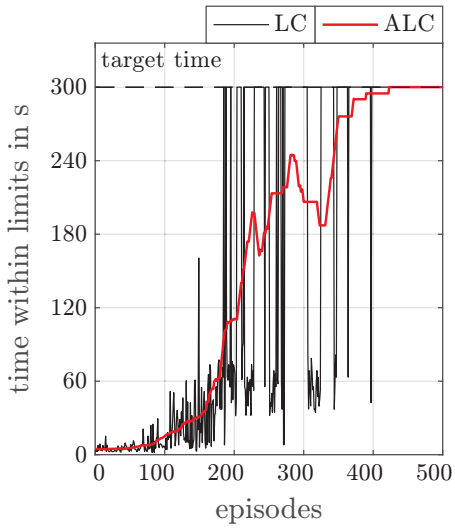


Fig. 4: Learning curve (in black) and moving average of learning curve (in red).

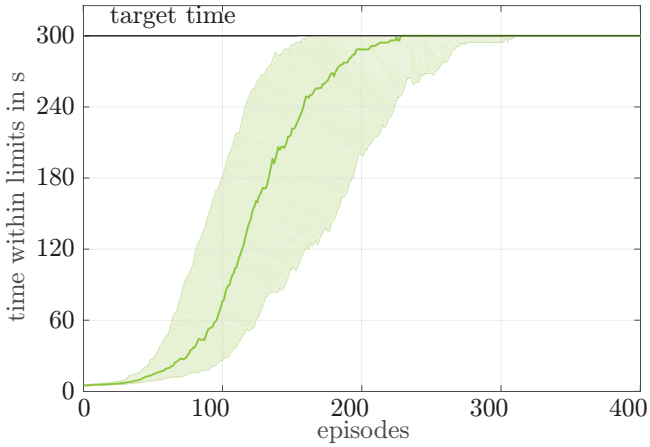


Fig. 5: Median and interquartile range (IQR) of the ALC obtained from 100 learning trials.

each time 400 episodes are run. The resulting ACL is shown in Fig. 5. Due to the inherent stochasticity of the dynamics and the learning algorithm, the learning curve is probabilistic. Hence, the median of the ALC and the interquartile range (IQR) of the ALC are illustrated. The median shows that the agent finds a policy after 230 episodes, after which it never fails the task again. For the 25-th percentile range the agent is even able to obtain such a policy after 150 episodes. Figure 6 shows the value function and Fig. 7 the policy in the state-space after the task has been learnt. Both are averaged over 100 performed simulations. As one would expect, the values of the states decrease with increasing distance from z_d. In order to maximize the expected return, the agent tries to avoid states which are located at the boundary of the state space. This is also represented by the policy. The agent chooses its actions to achieve small velocities. However, the

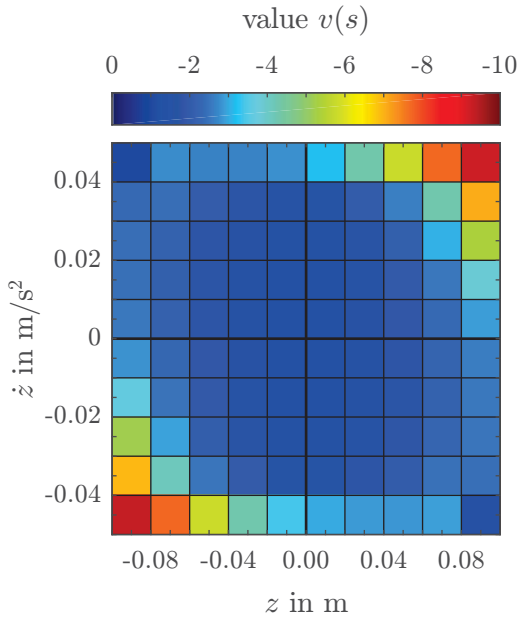


Fig. 6: Average value function obtained from 100 policies.

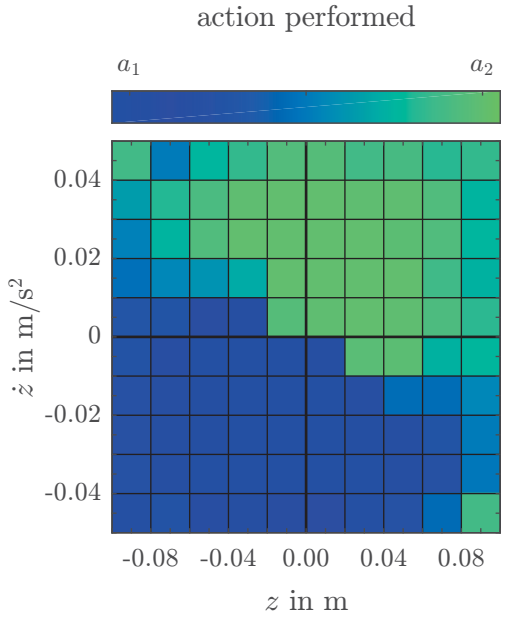


Fig. 7: Policy obtained from 100 policies.

action selection in regions with small velocities is altered in a way so that the agent tries to reduce the distance to the target depth.

B. Experiments

The RL algorithm is implemented on the diving agent, which was introduced in Section IV. The goal is to show that the agent is able to learn a policy for depth stabilization. The study is considered a success, if the diving agent achieves stabilization for 300 s, which demonstrates that it has learnt a policy for stabilization.

In order to implement the algorithm, the methods must be

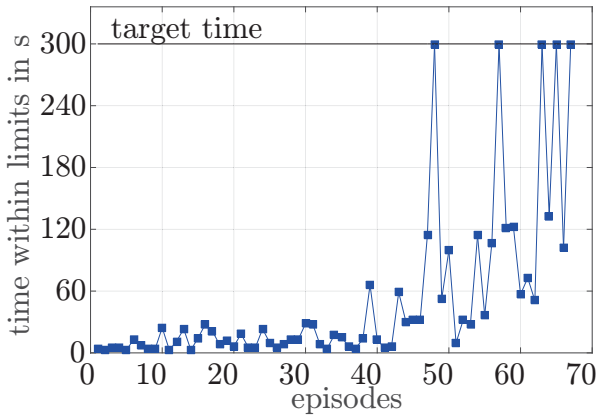


Fig. 8: Learning curve for the experimental validation of the RL algorithm.

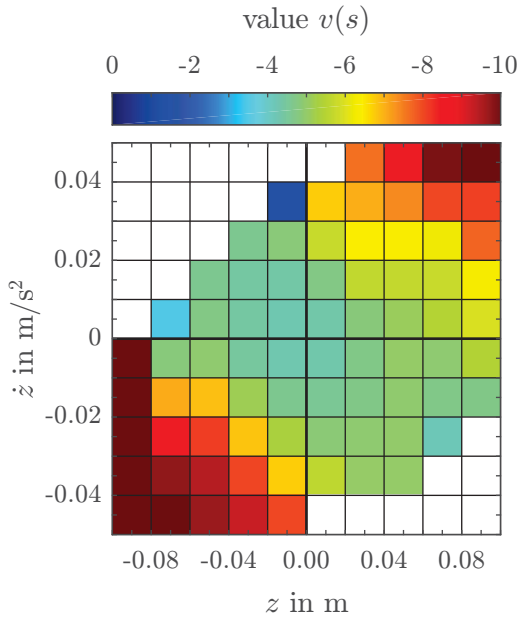


Fig. 9: Value function after the first success in the experiment in episode 47.

adjusted to cope with the operational limits of the embedded hardware. The microcontroller board is not able to store the matrices in its dynamic memory. Thus, a sparse matrix formulation is used to reduce the required memory. The agent can only measure its depth but not its velocity, a sliding mode observer is implemented for velocity estimation [16].

After each episode, the diving agent has to be guided back into the learning area before a new episode can be initiated. This is performed by a feedback controller, which is a regular sliding controller (SC) [17]. If the agent reaches a terminal state, the SC is activated in order to stabilize the agent at a random depth inside of the learning region, which is then used as the new initial depth for the subsequent learning episode. By starting in random positions, the agent explores the state-space faster, which leads to a more efficient development of the system dynamics model. After the agent is stabilized by the SC, the RL algorithm takes over with the next attempt of stabilizing the diving agent.

The desired target depth is set to be $z_\mathrm{d} = 0.4\,\mathrm{m}$. The position tolerance is chosen to $z_\mathrm{Tol} = 0.15\,\mathrm{m}$ and the velocity tolerance is set to $\dot{z}_\mathrm{Tol} = 0.03\,\frac{\mathrm{m}}{\mathrm{s}}$.

An LC for an experimental run is shown in Fig. 8. It demonstrates that the diving agent has learnt depth stabilization in episode 48. In order to show that the first success was no coincidence, the experiment was pursued until the target time was reached five times. The total learning time up to this point is 17 min. Note, that the agents remains up to 300 s in the learning region before a new episode begins, which is counted towards the learning time. After the first success, the agent shows a worse performance until the target time is reached again in episode 57. Afterwards, the target time is reached in episodes 63, 65, and 67 after which the experiment is terminated. This shows that the agent improves its policy with increasing number of performed episodes on the task. The value function of the agent after completing episode 48 are shown in 9. Not all states have been visited in the

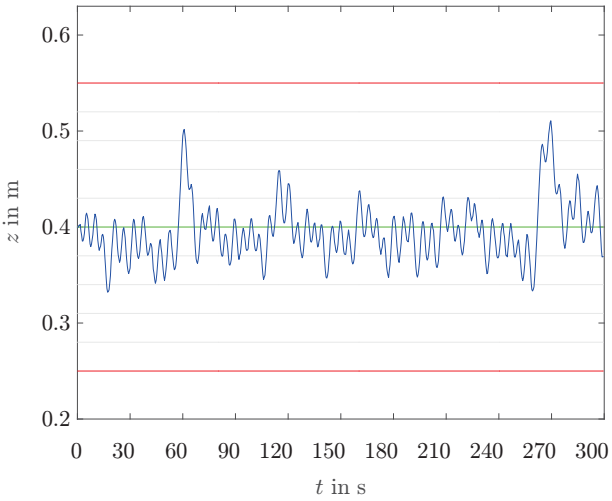


Fig. 10: Depth (in blue) of the diving agent during the episode of first successful stabilization. Desired depth (in green) and learning boundaries (in red) are shown as well.

experiment. This can have two reasons. First, the exploration of the state-space can be insufficient at this point of learning. Second, the system's dynamics can prevent the agent from reaching those states. As in simulations, two regions with low values emerge from the corners with high velocity directed to the terminal state. However, these regions are larger and fade out over more states. The qualitative structure of the value function for both experiments and simulations is equivalent.

Figure 10 visualizes the diving agent depth trajectory inside of the learning region for a succesfully learnt policy. Figure 10 demonstrates that the diving agent positions itself in the center of the learning region during the stabilization

and tends to return to the center. Additionally, the agent oscillates. This behavior is expected because a selected action is consistent within each state. Only when the state of the agent changes, a different action can be chosen by the agent. This leads to the oscillation of the agent between states that have different actions allocated in the policy. The effects of performing an action that counteracts the movement of the agent only occurs after some time delay. Hence, the oscillation amplitude can be reduced by adjusting the velocity and position ranges of the state-space or by a finite discretization. However, the limited memory of the given hardware setup prevents the implementation of more states in the state-space. In order to solve this problem, more dynamic memory is required for storing the state transition matrix.

In summary, the experiments show that the RL algorithm is able to learn a low-level control task on a highly embedded computationally constrained hardware platform. The agent finds a policy that provides stabilization within the learning zone.

VI. CONCLUSION AND FUTURE WORK

This paper investigated model-based value-function reinforcement learning (RL) with dynamic programming for embedded and computationally limited mechatronic systems. The task of underwater depth stabilization has been chosen for benchmarking purposes. It resembles the inverted pendulum problem because in both cases an unstable equilibrium is stabilized. A diving agent has been designed and built to evaluate the RL algorithm. The agent learns the system dynamics model in order to improve its value function, which is used for stabilizing the agent in a bounded area. Depth stabilization through learning was investigated in simulations as well as in water tank experiments.

The experiments showed that the agent is able to learn a policy for depth stabilization. However, it required 47 episodes to learn the task. The main reason for the slow convergence is that the state-space only includes depth and velocity, but not the state of the adjustable volume. However, since the volume adjustment is slow, neglecting its dynamics deteriorates the learning performance. The same action (increasing volume or decreasing volume) can have a completely different effect on the successor state for different states of the adjustable volume. As a consequence, the state transition matrices of both possible actions do only differ by a small probability fraction. Despite this missing information the agent was able to learn the stabilization task in experiments.

Future work will cover the improvement of the introduced RL algorithm. Different position and velocity discretizations of the state space will be investigated. Moreover, the state discretization of the task will be refined in order to enable the agent to counteract disturbances quicker. It is expected that this will result in smaller oscillations during the stabilization and increase the agent's proximity to the target depth. Furthermore, the dynamics model will be revised in order to enhance the action selection of the agent. The state-space will be augmented with the state of the adjustable volume. This should lead to a better performance for stabilization and a decrease of the total learning time for finding a control policy. The diving agent will be augmented with additional dynamic memory.

REFERENCES

[1] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[3] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[4] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," in *Advances in Neural Information processing Systems (NIPS)*, 2007, pp. 1–8.

[5] M. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on machine learning (ICML)*, 2011, pp. 465–472.

[6] T. Hester, M. Quinlan, and P. Stone, "Generalized model learning for reinforcement learning on a humanoid robot," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2010, pp. 2369–2374.

[7] ——, "RTMBA: A real-time model-based reinforcement learning architecture for robot control," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2012, pp. 85–90.

[8] T. Hester and P. Stone, "Intrinsically motivated model learning for developing curious robots," *Artificial Intelligence*, vol. 247, pp. 170–186, 2017.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[10] A. Hackbarth, E. Kreuzer, and E. Solowjow, "HippoCampus: A Micro Underwater Vehicle for Swarm Applications," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 2258–2263.

[11] A. Griffiths, A. Dikarev, P. R. Green, B. Lennox, X. Poteau, and S. Watson, "AVEXIS: Aqua Vehicle Explorer for In-Situ Sensing," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 282–287, 2016.

[12] W. M. Bessa, E. Kreuzer, J. Lange, M.-A. Pick, and E. Solowjow, "Design and Adaptive Depth Control of a Micro Diving Agent," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 1871–1877, 2017.

[13] A. Faust, I. Palunko, P. Cruz, R. Fierro, and L. Tapia, "Learning swing-free trajectories for UAVs with a suspended load," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2013, pp. 4902–4909.

[14] J. Yuh, "Learning control for underwater robotic vehicles," *IEEE Control Systems*, vol. 14, no. 2, pp. 39–46, 1994.

[15] C. Gaskett, D. Wettergreen, A. Zelinsky, *et al.*, "Reinforcement learning applied to the control of an autonomous underwater vehicle," in *Proceedings of the Australian Conference on Robotics and Automation (AuCRA)*, 1999.

[16] Y. Shtessel, C. Edwards, L. Fridman, and A. Levant, *Sliding mode control and observation*. Springer, 2014, vol. 10.

[17] J.-J. E. Slotine and W. Li, *Applied Nonlinear Control*. Prentice Hall Englewood Cliffs, NJ, 1991.