Deep Reinforcement Learning for Autonomous SONAR Port Monitoring

Christian Kanarski[1,2], Bastian Kaulen[1], Frederik Kühne[1], Tim Owe Wisch[1], Karoline Gussow[1], Sören Christensen[3], Gerhard Schmidt[1]

[1]*Digital Signal Processing and System Theory, Kiel University, Kiel, Germany*

[2]*DeepSea Monitoring Working Group, GEOMAR Helmholtz Center for Ocean Research Kiel, Kiel, Germany*

[3]*Stochastics Research Group, Kiel University, Kiel, Germany*

Email: {chk, bk, frk, timw, kars, gus}@tf.uni-kiel.de[1], *christensen@math.uni-kiel.de*[3]

Abstract

The use of MIMO-SONAR systems to autonomously monitor a port environment requires a robust control of the system parametrization and its adaptation to changing environmental conditions in real-time. Deep reinforcement learning (DRL) can be used to implement a control-assisting artificial intelligence (AI) which adapts the system parametrization in relation to the observed environment scans. Through the design of a reward function, the controlling agent can learn the fulfillment of given sub-goals, such as the evaluation of security risks and the management of limited computational and energy resources. During training, the agent explores the unknown environmental dynamics in a trial-and-error fashion and improves its policy by exploiting the gathered experiences of agent-environment interactions. By retrospective analysis of the chosen system parametrization and the resulting scan observations, the agent learns to adapt its monitoring strategy to fulfill the main goal of reliably detecting unwanted intruders inside of the port. This work presents the design of the reward function, the training architecture, and the performance evaluation of the trained deep reinforcement learning agent for a simulated port environment.

Introduction

Port environments with open entrances carry the risk of unwanted intruders entering to cause damage to ships or port infrastructure. A SONAR system can be stationed inside of the port to hydroacoustically monitor the area for possible attackers. Reliable and fast attacker detection is required for timely interception by a coastal guard. This requires real-time adaptation to dynamic environmental conditions with robust control of the SONAR system parametrization to optimally scan the regions of interest. When using mobile systems with limited energy and computational resources, tactical management of these limited resources is required for long-term autonomous monitoring. Such an autonomous control task can be achieved by an artificial intelligence (AI) agent that sets the parametrization of the next scan in relation to the currently observed environment as in Fig. 1. A port security scenario is considered here where the scan modes of a MIMO-SONAR system are controlled by a reinforcement learning (RL) agent. A virtual port environment written in the Python programming language is used to train the agent with the TensorFlow Agents (TF-agents) [2] library. This work extends the results presented in [1].

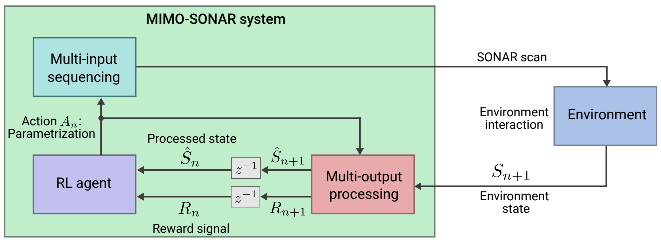
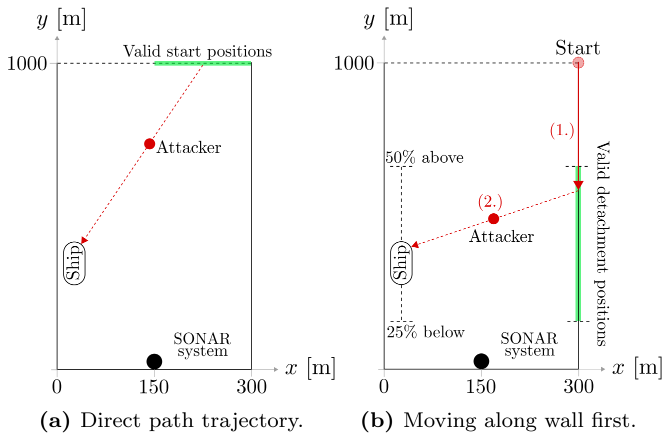


Figure 1: MIMO-SONAR parametrization control by a RL agent reacting to the observed environment.

Environment

The scenario presented is divided into unique episodes with the environmental conditions being redefined at the start of each episode. This allows the agent to learn a generally viable scanning strategy by seeing multiple different scenarios. The attacker appears with a delay uniformly sampled in the range of 2 s to 6 s. With a 50% chance, the attacker follows the strategy of moving directly towards the target ship as in Fig. 2a to reach it as fast as possible. Otherwise, the attacker first moves along the port walls as in Fig. 2b to be masked by the strong wall reflections visible in Fig. 3. For moving inside the water, the attacker uses an aqua scooter with a 50% chance resulting in a moving speed being uniformly sampled in the range of $2\,\frac{\mathrm{m}}{\mathrm{s}}$ to $5.5\,\frac{\mathrm{m}}{\mathrm{s}}$. Otherwise, the attacker dives with a uniformly sampled speed in the range of $0.5\,\frac{\mathrm{m}}{\mathrm{s}}$ to $1.5\,\frac{\mathrm{m}}{\mathrm{s}}$.



(a) Direct path trajectory. **(b)** Moving along wall first.

Figure 2: Attack strategies of the port intruder.

For each discrete time step n of the episode, the agent interacts with the environment by choosing an action A_n from the following options: Setting the scan parametrization and starting a near, intermediate or far field scan as

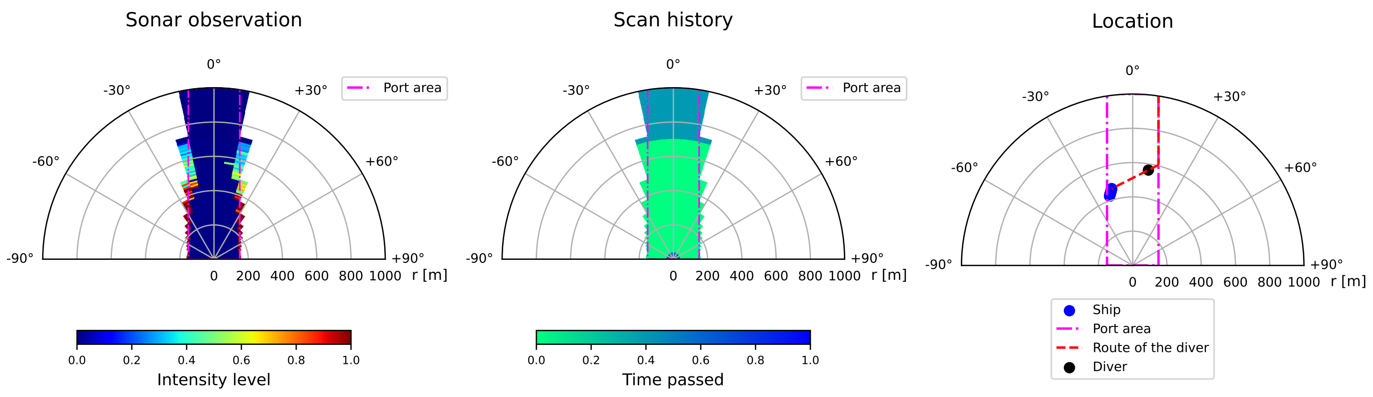


Figure 3: SONAR scan observations for the port training environment.

described in [1] to observe the port environment in the range of

$$\mathrm{range}(A_n) \in \begin{cases} [0\,\mathrm{m}, 375\,\mathrm{m}], & \text{Near field}, \\ [48\,\mathrm{m}, 700\,\mathrm{m}], & \text{Intermediate field}, \\ [675\,\mathrm{m}, 1000\,\mathrm{m}], & \text{Far field}, \end{cases} \tag{1}$$

or performing no scan to save energy. According to the chosen action, a SONAR scan observation as in Fig. 3 is generated to represent the observed environment state with a resolution of 30 beams and 100 cells per beam. The intensity of cells inside the port is set to Gaussian distributed environment noise. The active SONAR equation [3]

$$\underbrace{\mathrm{SE}}_{\text{Signal excess}} = \underbrace{\mathrm{SL}}_{\text{Source level}} + \underbrace{\mathrm{DI}}_{\text{Directivity index}} - 2\underbrace{\mathrm{TL}}_{\text{Transmission loss}} + \underbrace{\mathrm{TS}}_{\text{Target strength}} - \underbrace{\mathrm{NL}}_{\text{Noise level}} - \underbrace{\mathrm{DT},}_{\text{Detection threshold}} \tag{2}$$

is then used to calculate the intensity of the cells containing the attacker and the ship. To model the strong reflections at the port walls, the wall gain factor

$$w_{\mathrm{wall}} = G_{\mathrm{wall}} - 20\log_{10}(r_{\mathrm{wall}}), \tag{3}$$

with a quadratic intensity propagation loss term is added to the cells containing walls. Cells outside of the port area are masked with a value of -1 whereas cells inside the port are normalized to the maximum occurring intensity resulting in an intensity range of $[0, 1]$. The second generated observation is the scan history, storing the time elapsed since a cell was last scanned. This time is normalized and capped to a maximum of 23.5 s, the time needed to pass 12.5% of the port diagonal with an aqua scooter at maximum speed. This additional observation allows the agent to take its recent scanning behavior into account when evaluating the priorities of the port areas to scan. A detection is registered if the attacker's cell position matches the non-stationary maximum in the scan for 5 consecutive time steps. Stationary targets such as the walls and the ship are excluded from the detection by subtracting a recursively smoothed version of the last few scan observations before the maximum calculation. The attacker can therefore only be detected by appearing in a cell previously occupied by environment noise and if the corresponding area is scanned. An episode ends when the attacker reaches the ship or when a detection is registered. In the case of a detection, a successful interception by the coastal guard is assumed if it can reach the safety zone of 20 m around the ship's center before the attacker reaches that area.

Deep Reinforcement Learning

Deep reinforcement learning (DRL) is a deep learning method where a problem-facing agent, represented by a neural network, learns to achieve a desired goal by exploring an environment and learning from the interactions with it [4]. The goal is encoded by the reward function R_n, rewarding behavior which leads to the goal being achieved and punishing disadvantageous behavior. As in Fig. 1, the agent interacts with the environment by choosing an action A_n based on the currently observed environment state S_n and the environment returns the new environment state observation S_{n+1} and a reward R_{n+1} to the agent. The reward depends on how useful the state transition to S_{n+1} was with regard to reaching the goal from its successor states. These experience tuples

$$e_n = \langle S_n, A_n, R_{n+1}, S_{n+1} \rangle, \tag{4}$$

are stored in an experience replay memory for later training. In this work, the categorical deep Q network (C51) [5] is used to train a convolutional neural network (CNN) representing the RL agent. It is a distributional extension of the deep Q network (DQN) [6], where the underlying value probability distribution of the Q function

$$Q_\pi(s,a) = \mathbb{E}_\pi\left[\sum_{k=0}^{N_T} \gamma^k R_{n+k+1} \mid S_n = s, A_n = a\right], \tag{5}$$

is learned. This Q function represents the expected value of taking the action a in the state s by accumulating the rewards R_{n+k+1} discounted by the factor γ^k in the subsequent states. The agent's optimal policy

$$\pi^*(s) = \underset{a}{\mathrm{argmax}}\Big[Q(s,a)\Big], \tag{6}$$

aims to maximize the expected reward function by choosing actions that maximize the Q function, and therefore, the expected return.

Reward Design

The reward function

$$r_{\text{sum}}(n) = r_{\text{cost}}(A_n) + r_{\text{history}}(n) + r_{\text{time}}(n) + r_{\text{detect}}(n) + r_{\text{failure}}(n), \tag{7}$$

was designed to encourage the agent to learn a SONAR parametrization control strategy that leads to a reliable, fast, and energy efficient detection of the attacker. The energy cost term

$$r_{\text{cost}}(A_n) = \begin{cases} -P \cdot N_{T_x} \cdot t_{\text{sig}}(A_n), & A_n \,\widehat{=}\, \text{Scan}, \\ +\frac{1}{2} \cdot \min\left[P \cdot N_{T_x} \cdot t_{\text{sig}}(A_n)\right], & A_n \,\widehat{=}\, \text{Standby}, \end{cases} \tag{8}$$

represents the physical cost of performing a scan with transmit power P for N_{T_x} transmitting projectors and the signal duration $t_{\text{sig}}(A_n)$. It is negative for performing a SONAR scan, positive for the standby action to reward energy saving when appropriate and it is normalized to the maximum scan cost before scaling factors are applied. The history term $r_{\text{history}}(n)$ rewards the agent for how up-to-date the scan history is, being positive if more than half of the port area was recently scanned and negative otherwise. The negative time cost $r_{\text{time}}(n)$ increases for each time step passed in an episode to encourage a fast detection. The detection reward term $r_{\text{detect}}(n)$ rewards the agent for every consecutive scan of the attacker's area if the attacker is detected in this scan. This reward increases if the detection was fast enough for the coastal guard to intercept the attacker. The failure penalty term $r_{\text{failure}}(n)$ is the maximum negative reward if the attacker reaches the ship before an interception, even in the case of a preceding detection. This negative term tells the agent to avoid the ship attack from happening by all means.

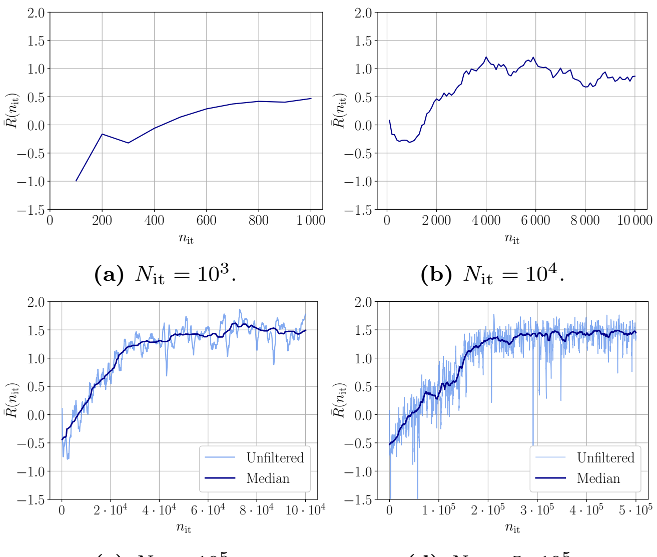
All of these terms can be multiplied with a scaling factor to increase or lessen their relevance for the control strategy of the agent. For example, when a stationary SONAR system with an active power supply is used, the relevance of the energy cost $r_{\text{cost}}(A_n)$ term can be decreased. Therefore, the agent can utilize more energy consuming scans to minimize the detection time and maximize the detection rate.

Training

The architecture of the CNN used to represent the agent is given in Tab. 1 with two convolutional and two fully connected (F.c.) layers, all followed by a rectified linear unit (ReLU) activation function. The outputs of the last layer are the discrete probability distributions of the Q functions from Eq. (5) for each action. As is common for C51 [5], 51 bins were used to model each distribution. The C51 implementation of the TF-agents library was used to train the CNN. Four agents were trained with the same network architecture and hyperparameters used but with different number of training iterations. The number of training episodes chosen were 10^2, 10^3, 10^4 and $5 \cdot 10^4$ with 10 training iterations per episode and an experience memory length of 10^5. The evaluation metric collected during training was the average collected episode reward over the last 10 episodes as seen in Fig. 4. For the longest trained agent with $N_{\text{it}} = 5 \cdot 10^5$ training iterations in Fig. 4d, the training took approximately 8 hours on a PC with an AMD Ryzen 5 3600 CPU, a Quadro RTX 4000 GPU and 16 GB of RAM.

Table 1: CNN architecture used to represent the agent.

Layer	Kernels	Size	Stride	Neurons
Conv.	32	8×8	4	/
Conv.	32	4×4	4	/
F. c.	/	/	/	512
F. c.	/	/	/	$51 \cdot N_{\text{a}}$



(a) $N_{\text{it}} = 10^3$. **(b)** $N_{\text{it}} = 10^4$. **(c)** $N_{\text{it}} = 10^5$. **(d)** $N_{\text{it}} = 5 \cdot 10^5$.

Figure 4: Agent's average collected reward for varying training iterations N_{it}.

Evaluation

As seen in Fig. 4, the maximum reward earned on average increases with rising amounts of training iterations. While the least trained agent in Fig. 4a reached a maximum reward of about 0.5, the longest trained agent in Fig. 4d managed to achieve a maximum average reward of approximately 1.5. As the reward is an indicator of how well the agent manages to achieve the given goals, longer training seems to lead to a better control strategy. To see the differences in the control-strategies of the agents, each agent was placed in the training environment for 1000 evaluation episodes. Averaged over all evaluation episodes, the chosen actions in Fig. 5, the episode steps taken and reward earned in Tab. 2 and the terminal states in Fig. 6 were recorded. The considered terminal states are the successful interception of the attacker by the coastal guard, a detection occurring but too late for a successful interception or a failed interception as described before.

The agent trained for $N_{\text{it}} = 10^3$ iterations learned to exclusively use the far field scan and therefore only detects attackers appearing in the far field, leading to a failure rate of 34.9% and a negative average reward of -2.00. This agent presumably did not explore the environment

Table 2: Average steps taken and reward earned per episode over 1000 evaluation episodes.

Training iterations	Average steps/episode	Average reward
1000	237	-2.00
10 000	358	$+0.56$
100 000	265	$+1.79$
500 000	198	$+1.46$

interactions enough to learn a satisfying control strategy due to the limited number of training iterations. With $N_{\text{it}} = 10^4$ training iterations, the next agent learned to utilize all actions such as the standby action in 21.4% of steps to save energy, achieving a higher average reward of $+0.56$. Due to a failure rate of 33.9%, this agent also did not learn a reliable interception scan strategy. Earning the highest average reward of $+1.79$, the agent trained for $N_{\text{it}} = 10^5$ iterations learned to utilize the standby action in 33.2% of steps, achieving a satisfactory total detection rate of 98.3% while saving energy resources. The longest trained agent with $N_{\text{it}} = 5 \cdot 10^5$ iterations achieves the highest successful interception rate with 93.0% and the highest total detection rate with 99.6%. This agent learned to prioritize a fast detection, with the lowest average steps per episode of 198, over collecting positive rewards by utilizing the standby action. This results in a lower average reward of $+1.46$. Thus, the scaling factor of reward terms strongly influences the agent's control-focus and not only the average reward should be taken into account when evaluating the agent's learned strategy. It is apparent that more training iterations lead to better agent policies in terms of the collected reward and fulfilling the goal of a reliable and energy efficient attacker detection.

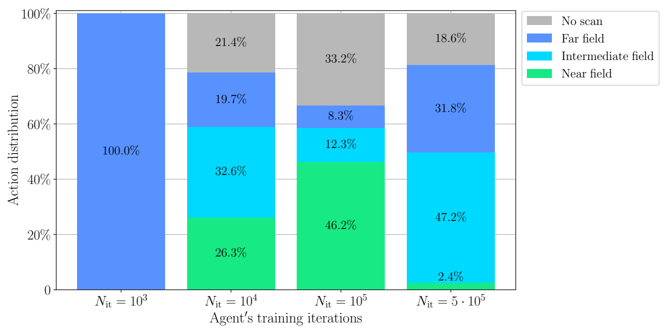


Figure 5: Action distribution in 1000 evaluation episodes for each trained agent.

The agents exported neural networks were imported in the Kiel Real-time Application Toolkit (KiRAT) [7], where the presented scenario is physically more accurately simulated. Therefore, the generated SONAR scan observations slightly differ from the training environment observations which led to a different agent behavior for all trained agents. For example, the longest trained agent only scanned the far field area, possibly falsely assuming an attacker being present in that area. The influence of observation differences on the agent's behavior should therefore be investigated and minimized if necessary.

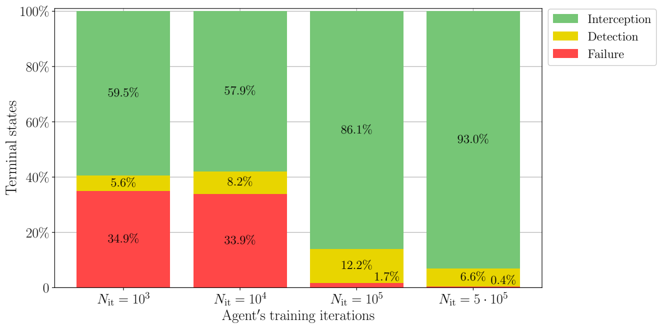


Figure 6: Terminal states in 1000 evaluation episodes for each trained agent.

Conclusion and Outlook

In this work, a port security scenario was implemented in a virtual training environment for the training of RL agents to learn a SONAR system parametrization control to enable a reliable and energy efficient attacker detection. With enough training iterations, the agents learned to reliably fulfill the given goals by learning from environment interactions and adapting their strategies to the observed environment to maximize the given reward function. For compatibility of the trained agents with differing deployment environments, differences between the environment observations have to be investigated. Accurate stochastic modelling of the deployment observations for the training environment could be investigated in future work to solve the presented difficulties.

Acknowledgement

The first author is funded through the Helmholtz School for Marine Data Science (MarDATA), Grant No. HIDSS-0005.

References

[1] Neumann, N., Kaulen, B., Christensen, S., Schmidt, G., 2021. Deep Reinforcement Learning for Cognitive SONAR Systems. In Proc. DAGA 2021, Wien.

[2] TF-Agents: A library for Reinforcement Learning in TensorFlow, 2018. `https://www.tensorflow.org/agents/`. Online, last access: 20.03.2023.

[3] Hodges, R. P., 2011. Underwater acoustics: Analysis, design and performance of sonar. John Wiley & Sons.

[4] Sutton, R. S., Barto, A. G., 2018. Reinforcement learning: An introduction. MIT press.

[5] Bellemare, M. G., Dabney, W., Munos, R., 2017. A distributional perspective on reinforcement learning. In International conference on machine learning, pp. 449-458. PMLR.

[6] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... Hassabis, D., 2015. Human-level control through deep reinforcement learning. In Nature, 518(7540), 529-533.

[7] KiRAT - Kiel Real-time Application Toolkit, 2023. `https://kirat.de/`. Online, last access: 20.03.2023.