Draft version, to appear in IWANN 2021: Advances in Computational Intelligence

Autonomous Docking of Mobile Robots by Reinforcement Learning Tackling the Sparse Reward Problem

A.M Burgueño-Romero, J.R. Ruiz-Sarmiento, and J. Gonzalez-Jimenez

Machine Perception and Intelligent Robotics group (MAPIR). Dept. of System Engineering and Automation. Biomedical Research Institute of Malaga (IBIMA). University of Malaga. Spain.
{ambrbr,jotaraul,javiergonzalez}@uma.es

Abstract. Most mobile robots are powered by batteries, which must be charged before their level become too low to continue providing services. This paper contributes a novel method based on Reinforcement Learning (RL) for the autonomous docking of mobile robots at their charging stations. Our proposal considers a RL network that is fed with images to visually sense the environment and with distance measurements to safely avoid obstacles, and produces motion commands to be executed by the robot. Additionally, since the autonomous docking is in essence a sparse reward task (the only state that returns a positive reward is when the robot docks at the charging station), we propose the usage of reward shaping to successfully learn to dock. For that we have designed extrinsic rewards that are built on the results of a Convolutional Neural Network in charge of detecting the pattern typically used to visually identify charging stations. The experiments carried out support our design decisions and validate the method implementation, reporting a $\sim$ 100% of success in the docking task with obstacle-free paths, and $\sim$ 93% when obstacles are considered, along with short execution times (10s and 14s on average, respectively).

Keywords: Autonomous docking · Reinforcement Learning · Mobile Robots · Pattern detection · Reward Shaping · CNN · Unity

1 Introduction

We all take care about the level of charge of our phones, hungrily looking for a charging point when they are running out battery so we can keep doing productive things like chatting or sending emails. The same is the case with mobile robots, whose landing in fields like education, housekeeping, health care or entertainment is becoming more and more evident, and which require to keep their batteries charged in order to reliably and autonomously provide their services [1, 2, 3]. For this purpose, a fundamental task is that of docking at the charging station. During the robot operation, this occurs when the robot's battery is running out of charge, or when it is estimated that the battery level is insufficient

to perform the remaining tasks. At that moment, the robot must initiate the docking process, which typically consists of the navigation to the area where the charging station is located, the identification of said station, and the approach of the robot towards it until docking takes place.

Over the past few years, different techniques have been proposed to detect the charging station in the environment, typically based on visual information (images coming from cameras), as well as to guide the robot towards it. In this way, it is common to see charging stations incorporating some sort of fiducial marker or pattern to ease their identification (see Fig. 1). Initial works relied on traditional computer vision techniques for edge segmentation, feature extraction, template matching, etc., aiming to detect such pattern [4, 5]. However, these approaches suffer from their high parametrization [6], which turns them into inflexible techniques prone to fail in presence of changing lighting conditions, occlusions, etc. Recent visual docking techniques made the move to Convolutional Neural Networks (CNN) [7, 8] in order to detect the pattern. Although these methods are more robust against challenging conditions, they require the collection of a vast amount of training data in order to be properly fitted. This, besides being a tedious and highly time-consuming task, can be difficult to carry out in specialized domains, as it is the case of pattern detection.

In spite of the technique used to detect the pattern, in order to dock at the charging station, it is typically implemented a simple algorithm: the robot is instructed to rotate and align the pattern in the center of the image, and then to move forward until docking is complete, making the strong assumption that the path is obstacle-free. An alternative to this approach is the utilization of Reinforcement Learning (RL) algorithms, which have been explored in recent works to automate robots' navigation towards a certain goal using visual information (intensity images) as input data [9, 10]. A reward function, needed for the fitting of RL algorithms, evaluates the action performed by the robot in a specific state, and returns a number which sign depends on whether the action it has performed is right or wrong for the task it has been assigned. However, a common issue that recurrently appears in RL problems is solving sparse reward tasks, that is, tasks where the amount of states that return a positive reward is very limited, as is the case of docking at the charging station. Sparse rewards cause the robot not to acquire the information needed to solve the problem, resulting in an unreliable operation.

This work contributes a novel method for the autonomous docking of mobile robots by means of Reinforcement Learning (RL) that deals with the previous issues. Concretely, this method addresses the sparse rewards problem by considering a CNN to detect the charging station pattern in RGB images, and the utilization of those detections to provide extrinsic rewards. This way, the proposed method performs docking by instantiating a RL network that is fed with RGB images and pattern detection results, and that produces actions to move the robot (translations and rotations) towards the charging station. We also consider an additional input to this RL network: distance measurements collected by a radial laser scanner, a sensor typically found in robotic platforms and that

Fig. 1. Left, charging station for a domestic robot composed of three circles [4]. Middle, station for a drone consisting of a circle with a cross [11]. Right, a charging station identified by two QR codes [5].

provides valuable information about the obstacles in the robot surroundings. By doing so we avoid an obstacle-free path assumption and permit the method to successfully operate in more challenging environments.

The training of a RL network requires the deployment of the robot in its working environment to learn by trial-and-error, which demands a long time to achieve robust results, being crashes also possible. To handle this, we have resorted to Unity [12], a video game development framework that permitted us to design realistic virtual environments including robots and charging stations, and to perform a faster RL network training. Regarding the CNN for pattern detection, as previously introduced, its training requires a vast amount of data for fitting a reliable model. In this regard, we propose the utilization of transfer learning [13] and the fine-tuning of a general model for the detection of the particular pattern used in each docking scenario (recall Fig. 1). To carry out such fine-tuning, Unity is also used to generate synthetic training samples in the form of images including the pattern at hand with different sizes, orientations, and lighting conditions.

To validate our proposal, we have carried out extensive tests with a replica of the Giraff Robot [14] (see Fig.2). The robot was commanded to dock at the charging station, starting from different relative distances and orientations w.r.t. said station. The performance of our proposal has been compared with other methods following other typical RL techniques such as behavior cloning [15], reporting a higher performance. The method implementation is publicly available at `https://github.com/AmbroxMr/UnityMLDocking`.

2 The Proposed Method for Autonomous Docking

Fig. 2 shows the pipeline of our proposal. Once the robot triggers the *docking at the charging station* task, we resort to a RL network, described in detail in Sec. 2.1, to produce the motion commands to reach such goal. In this work we assume that the robot is located in the same room as the charging station, otherwise a prior navigation to said room would be needed. At a certain time instant while executing the docking task, the RL network processes the following input data: an image from an RGB camera for visually sensing the environment,

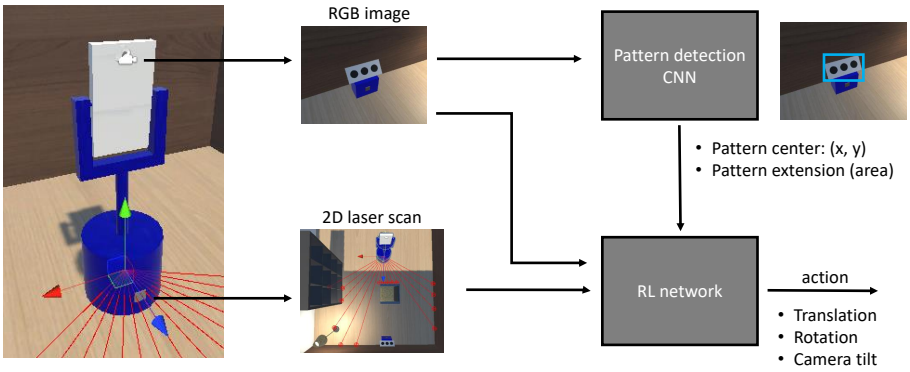
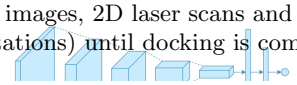


Fig. 2. Pipeline of the proposed method, where the RL network is in charge of processing images, 2D laser scans and pattern detections to produce actions (translations and rotations) until docking is complete.



distance measurements in the form of a 2D laser scan useful for avoiding obstacles, and the results from a pattern detection CNN (see Sec. 2.3) for facing the sparse rewards problem (see Sec. 2.2). This way, we consider a common robotic platform equipped with an RGB camera and a 2D laser scanner. The Giraff robot, recreated in our experiments, is capable of tilting the camera, but this is not a requirement for our method to work. The 2D laser scanner could be also replaced by other sensor or technique that provides distance measurements such as RGB-D cameras, sonars, stereo-vision systems, etc. With such inputs, the RL network infers an action (motion command), and the process is repeated until docking is completed. Specifically, the possible robot actions are translation (forward/backward), rotation (left/right) and tilt (increase/decrease).

Since we are dealing with RL techniques, it is worth mentioning at this point the learning environment used for the design and validation of the proposed method. Typically, for the sake of generality it is preferred the utilization of environments that randomly change between method executions (different objects and at different locations, different lighting conditions, different charging station and initial robot positions, etc.). For doing so we have resorted to the Robot@VirtualHome tool[1] (see Fig. 3), which provides interesting mechanisms to facilitate working with domestic environments.

2.1 Designing the Reinforcement Learning Network

We have leveraged the Unity ML-Agents Toolkit[2] for the design, training and execution of the RL network. This open-source toolkit allows us to integrate Unity and Python, providing implementations of state-of-the-art Reinforcement Learning algorithms. Concretely, ML-Agents offers an implementation of two popu-

[1] `https://github.com/DavidFernandezChaves/RobotAtVirtualHome`
[2] `https://github.com/Unity-Technologies/ml-agents`

Fig. 3. Randomized virtual scenarios built by the Robot@VirtualHome tool, used for the design and validation of the proposed method.

lar RL algorithms: Proximal Policy Optimization [16] (PPO) and Soft Actor-Critic [17] (SAC).

On the one hand, PPO is an on-policy algorithm which trains a stochastic policy $\pi_\theta(A|S)$, meaning that the policy π is the probability of taking an action $a \in A$, at state $s \in S$, and the network parameters are θ. It explores by sampling actions according to the latest version of this policy, which is updated using intrinsic and extrinsic rewards (adding them). An intrinsic reward determines the current objective function for the learning agent (a successful docking in our case) while extrinsic rewards encourage the agent to achieve that goal. In PPO, new policies use to be close to previous ones, becoming progressively less random during the training phase.

This policy is trained by means of both intrinsic and extrinsic rewards (r_t) that come from the environment at time step t. This way, the value of the policy π_θ, denoted $J(\pi_\theta)$, is the expected discounted sum of rewards obtained by the robot:

$$J(\pi_\theta) = \mathbb{E}_{\pi\theta}\left[\sum_t \gamma^t r_t\right] \tag{1}$$

where γ is the discount factor, a hyperparameter that quantifies how much importance is given to the rewards.

Thereby, having the rewards r_t from all time steps t within an episode (time steps between a initial state and a terminal one), it is possible to update the policy π_θ computing the gradient of the value J with respect to the policy parameters θ:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\pi\theta}\left[\sum_t (\gamma^t r_t) \nabla log \pi_\theta(a_t|s_t)\right] \tag{2}$$

On the other hand, SAC is an off-policy algorithm that add an entropy measure of the policy into the reward to encourage exploration. The idea in this case is to randomly learn a policy that is able to succeed in the assigned task. During the method design phase we have considered both PPO and SAC alternatives in order to choose the most appropriate one for the problem at hand. This analysis is reported in Sec. 3.2.

In spite of the applied RL learning algorithm, the utilization of a reward function is required. In this work it has been defined with positive signals when the robot completes a docking.

2.2 Dealing with the Problem of Sparse Rewards

The task at hand, the autonomous docking of mobile robots, tends to be a sparse reward task. This is due to the fact that the robot only receives a positive reward just when a successful docking is achieved. However, this is an uncommon situation if the robot starts the training phase without any prior information. Sparse reward tasks have been an important subject of research [18, 19]. A solution to this problem is *reward shaping*, which consists of adding extra rewards obtained from the environment. Recently, new solutions have explored alternatives to adding new rewards obtained by sensing the environment. One of these is *behavioral cloning* [15], which aims to learn from demonstrations of a real person controlling the robot towards the target. Another is *curriculum learning* [18], which begins the training phase considering a very relaxed version of the problem, getting more complex over time until the robot can solve the initially given task.

As a novelty, our proposal considers a reward shaping solution consisting of the addition of new extrinsic rewards taking advantage of the visual information the robot receives, as well as pattern detection methods. This concept is put into practice by modeling a Convolutional Neural Network that detects the position and extension of the charging station pattern in the image.

Thus, an $extrinsic\_reward(\cdot)$ function has been defined, which evaluates the orientation of the detected pattern w.r.t. the camera and its proximity. The former is related to the difference between the center of the pattern $C = (C_x, C_y)$ and the center of the image, while the latter is rated by the area A of the bounding box containing the pattern in the image: a large area means a close pattern and vice versa. The defined function is as follows:

$$extrinsic\_reward(C, I_s, A) = \frac{er_x(C, I) + er_y(C, I) + er_z(A)}{Max_s * 9} \tag{3}$$

$$er_x(C, I) = \frac{-|C_x - I_w/2|}{I_w/4} + 3, \qquad er_y(C, I) = \frac{-|C_y - I_h/2|}{I_h/4} + 3$$

$$er_z(A) = \frac{A * 3}{Max_a}$$

being $I_s = (I_w, I_h)$ the image size, Max_s the number of maximum steps per episode, and Max_a the maximum possible area of the pattern projected on the image. Notice that these parameters play a normalization role. The functions $er_x(\cdot)$ and $er_y(\cdot)$ are similar, mapping the distance between the center of the pattern and the center of the image, to the range $[1, 3]$ in both the x and y axes. In its turn, $er_z(\cdot)$ maps the distance between the robot and the docking station to the range $[0, 3]$ according to the area of the pattern appearing in the image. This way, this extrinsic reward function provides signals from $2/(Max_s * 9)$ (pattern detected in the corner of the image and far away) to $1/Max_s$ (accomplished docking). Additionally, a negative reward of $-1/Max_s$ is provided when the pattern is not detected, encouraging the agent not to lose sight of the pattern too long. The normalization carried out considering Max_s pursuits the moderation of this extrinsic reward so it does not totally govern the learning process.

2.3 The Pattern Detection Network

In order to detect in images the pattern of the charging station, we propose the utilization of a CNN. These networks exhibit a great performance while detecting objects in images, although they require a heavy training phase that needs, among others, a vast repository of training data. Moreover, they require a fine-tuning to detect specific objects, as is the case of charging station patterns.

To face these issues we propose exploiting transfer learning, along with the generation of synthetic data for fine-tuning the model. For doing so, we relied on the Unity's *Perception*[3] package, which implements a toolkit for generating large-scale datasets. Using this toolkit, synthetic images and their ground truth can be generated, whether for 2D object detection, class segmentation or pose estimation (among others). Additionally, it is possible to model the randomization of different parameters in the environment where the synthetic images generation takes place, as is the case of lighting conditions, colors and textures of objects, etc. (see Fig. 4). Sec. 3.1 describes the generated dataset in this work.

Regarding transfer learning, this entails the selection of an initial network architecture pre-trained on an extensive dataset to enable the creation of a generalized model [13]. Then, modifications to the pre-existing model are done to fine-tune it for the problem at hand, which includes the utilization of the synthetically generated data for further training the network. Notice that this step

[3] `https://github.com/Unity-Technologies/com.unity.perception`

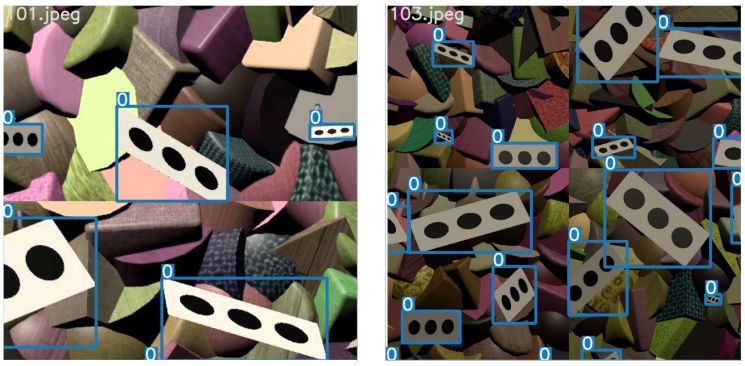


Fig. 4. Batch of synthetic data for the pattern considered in this work, used for the fine-tuning of the CNN. It is composed of 2 images and its ground truth. Lighting, colours, textures, rotation and scale are randomized parameters.

needs to be done out of Unity. Although we have employed PyTorch, there are other valid alternatives as Caffe, Tensorflow, etc. Once the network has been trained, Unity permits to import models in the common *.onnx* format by means of Barracuda[4], a lightweight cross-platform Neural Networks inference library.

3 Evaluation

3.1 Implementation Details

The main two decisions to be made in the method implementation are both the RL network and the pattern detection CNN to be used. The network implemented in the Reinforcement Learning algorithm is the CNN architecture proposed by Mnih et al. [20], consisting of three convolutional layers followed by two fully-connected layers with a single output for each valid action.

Regarding the pattern detection CNN, as introduced in Sec. 2.3, in this work we resorted to transfer learning and the Faster R-CNN model with a MobileNetV3-Large FPN backbone as initial network architecture. Said network was pre-trained on Imagenet, a widely used repository of data large enough (1.2M images) to create a generalized model. The fine-tuning of this method was carried out using a synthetic dataset consisting of 2000 images with approximately 5 patterns appearing in them, which are accompanied by their respective ground truth (patterns' bounding boxes), as shown in Fig. 4.

It is also worth mentioning the pattern chosen to identify the charging station, composed of three dark circles in a row over a white rectangle. This pattern has proven to be discriminant enough to be distinguished from other objects in most domestic scenarios, unequivocally identifying the charging station [4]. However, other geometric patterns or even QR codes could be used.

[4] `https://github.com/Unity-Technologies/barracuda-release`

3.2 Analysis of RL alternatives

To discern which reinforcement learning algorithm use, SAC or PPO, we analyzed the resources required by each one for training a RL network. SAC demands much more RAM due to the experience buffer it requires, which collects experiences during the training phase. In the context of this work, this becomes unfeasible due to the high size of visual observations. As a consequence, although SAC takes much fewer episodes to teach the robot to dock at its charging base, each iteration is significantly slower than in PPO. These are some of the reasons why it has been decided that PPO is more optimal for this particular problem.

Moreover, we have analyzed the evolution of two different learning metrics when considering three different RL methods: PP0, PP0 + Behavior cloning, and PPO + Extrinsic rewards (our proposal). For that, these methods have been in a 350k steps training phase each one during approximately 3 hours and a half, using a GPU Nvidia GeForce GTX 1660 Super. During the execution of these training phases they were computed the average scores of the episodes completed each 10k steps (the maximum length of an episode is 5k steps), as well as the length of those episodes measured in steps. Fig. 5 shows the obtained results. As evidenced by the evolution of the average score obtained by the PPO method, the sparse rewards problem appears. This is due to the robot not consistently getting positive rewards over time, leading in the long term to random movements or even to the robot standing still, not achieving a significant score improvement. When behavior cloning [15] (BC) is considered, a decent score is achieved quickly. However, after some time improvement is no longer happening, as the algorithm is overfitting to the behavior provided in the demo, which normally does not include all possible cases. Finally, our proposal keeps improving longer, converging to a score slightly higher than 1. This reward is split into 1 point when docking is successful (provided as intrinsic reward), and a small reward for looking at the pattern every step (provided as extrinsic reward). As it can be observed, during the very first episodes (until 25k steps), the robot gets

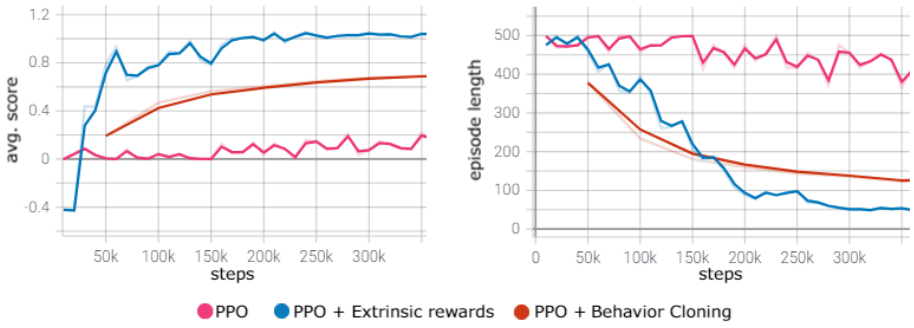


Fig. 5. Average score and episode length (expressed in batches of size 10), computed from the episodes completed in batches of 10k steps, in a 350k steps training phase for PPO, PPO + BC and our proposal.

negative rewards. We argue that this changes once the algorithm detects that keeping the pattern in sight gives positive rewards. Between 25k and 50k steps, the robot learns that if it gets closer to the pattern, rewards are higher. This happens until the robot actually docks successfully, obtaining a huge reward (1 point) and having little room for improvement. From step 175k, the robot is able to perform autonomous docking, hence converging faster than the competitors.

Regarding the evolution of the length of the episodes for the different methods (right part of Fig. 5), we can see how it remains almost horizontal for PPO, while PPO + BC and our proposal experiment a decrease in said length when the training phase progresses. This is because as both methods are trained, the robot performs the docking in less steps.

3.3 Performance Results

We have evaluated the performance when carrying out docking for two methods: PPO + Behavior cloning and PPO + Extrinsic rewards. The PPO method is omitted, since the resultant RL network was unable to accomplish docking. For this test two different scenarios have been considered: one with obstacle-free paths, and other one with objects placed at random locations between the robot and the charging station. A total of 3000 autonomous docking tasks were carried out in each scenario, giving 100s to the robot to complete each one. For each task execution they were recorded: the robot initial position and orientation w.r.t. the charging station, the execution time and the docking success.

In the obstacle-free scenarios, our proposal achieved a performance of 99.8% successful docking tasks, resulting in a highly reliable method. Regarding the execution time, it was low: $\sim$ 10s on average. In its turn, the PPO + Behavior cloning method reached a 64% of success with an execution time of $\sim$ 12s.

As for the scenarios containing obstacles, the results achieved by these methods are shown in Fig. 6. In this figure, the task executions are grouped according to the initial position and orientation of the robot w.r.t. the charging station,

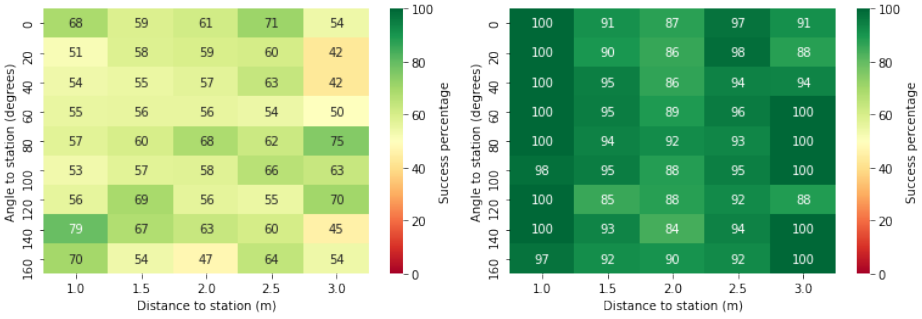


Fig. 6. Average success (expressed as a percentage) when docking at the charging station according to the initial robot distance and orientation w.r.t. said station using: PPO + Behavior cloning (left) and PPO + Extrinsic rewards (right).

aiming to measure how they affect docking. As can be seen, in both cases the robot has a similar success rate regardless of the initial orientation, but the distance has a remarkable effect in our method: the performance is quite high for low distances, it slightly decreases for medium distances, and increases again for larger ones. This can be due to the fact that with a larger distance, it is more probable for the robot to avoid the obstacle. Regardless of this, the average performance of our method is quite superior to the one using BC, a $\sim 93\%$ of successful docking tasks versus a $\sim 59\%$.

Regarding execution times, they are not affected by the initial orientation, but they tend to increase with growing distances with both methods. The reported averaged times were similar: 17s for BC and 14s for our proposal.

4 Conclusions and future work

This work has presented a novel method for performing the autonomous docking of mobile robots using a Reinforcement Learning (RL) network. An innovative way to face the sparse reward problem is presented, which considers reward shaping. This consists of providing extrinsic rewards to the RL network, which are built on the output of a Convolutional Neural Network (CNN) in charge of detecting the pattern identifying the charging station. We dealt with the problem of fitting a CNN for detecting a specific pattern by means of transfer learning and synthetic training data generation. This way, the RL network is fed with: images that visually sense the environment, distance measurements for perceiving and avoiding obstacles, and the extrinsic rewards, and produces actions to be carried out by the robot (translations and rotations) in order to accomplish docking. The method has been designed and validated using different tools from Unity. An extensive evaluation has been made, where our proposal achieved a success of $\sim 100\%$ and $\sim 93\%$ in obstacle-free and cluttered paths, respectively, also showing short execution times (10s and 14s, respectively).

In future work, we plan to connect Unity with the popular Robot Operating System (ROS) using the *ROS-TCP-Connector* package.

Acknowledgements. This work was supported by the research projects WISER (DPI2017-84827-R) and ARPEGGIO (PID2020-117057).

References

[1] Matteo Luperto et al. “Towards long-term deployment of a mobile robot for at-home ambient assisted living of the elderly”. In: *2019 European Conference on Mobile Robots (ECMR)*. IEEE. 2019, pp. 1–6.

[2] Iis P Tussyadiah and Sangwon Park. “Consumer evaluation of hotel service robots”. In: *Information and communication technologies in tourism 2018*. Springer, 2018, pp. 308–320.

[3] José Raúl Ruiz-Sarmiento, Cipriano Galindo, and Javier González-Jiménez. "Robot@ home, a robotic dataset for semantic mapping of home environments". In: *The International Journal of Robotics Research* 36.2 (2017), pp. 131–141.

[4] J. González-Jiménez, C. Galindo, and J.R. Ruiz-Sarmiento. "Technical improvements of the Giraff telepresence robot based on users' evaluation". In: *2012 IEEE RO-MAN*. 2012, pp. 827–832.

[5] Yuanzhe Wang et al. "Autonomous Target Docking of Nonholonomic Mobile Robots Using Relative Pose Measurements". In: *IEEE Transactions on Industrial Electronics* (2020), pp. 1–1.

[6] A.M. Burgueño Romero et al. "A collection of Jupyter Notebooks covering the fundamentals of computer vision". In: *ICERI2020 Proceedings*. Online Conference, 2020, pp. 5495–5505.

[7] M. F. Yahya and M. R. Arshad. "Detection of markers using deep learning for docking of autonomous underwater vehicle". In: *2017 IEEE I2CACIS*. 2017, pp. 179–184.

[8] Andreas Kriegler and Wilfried Wöber. *Vision-based Docking of a Mobile Robot*. Tech. rep. EasyChair, 2021.

[9] Frederik Ebert et al. *Visual Foresight: Model-Based Deep Reinforcement Learning for Vision-Based Robotic Control*. 2018. arXiv: `1812.00568 [cs.RO]`.

[10] Lei Tai, Giuseppe Paolo, and Ming Liu. "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation". In: *2017 IEEE/RSJ IROS*. 2017, pp. 31–36.

[11] *H3 Dynamics homepage*. https://www.h3dynamics.com/. Accessed: 4/20/21.

[12] Arthur Juliani et al. "Unity: A general platform for intelligent agents". In: *arXiv preprint arXiv:1809.02627* (2018).

[13] Kaiming He, Ross Girshick, and Piotr Dollar. "Rethinking ImageNet Pre-Training". In: *Proceedings of the IEEE/CVF ICCV*. Oct. 2019.

[14] Javier Gonzalez-Jimenez, Cipriano Galindo, and Carlos Gutierrez-Castaneda. "Evaluation of a Telepresence Robot for the Elderly: A Spanish Experience". In: *Natural and Artificial Models in Computation and Biology*. 2013, pp. 141–150.

[15] Faraz Torabi, Garrett Warnell, and Peter Stone. *Behavioral Cloning from Observation*. 2018. arXiv: `1805.01954 [cs.AI]`.

[16] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017). arXiv: `1707.06347`.

[17] Tuomas Haarnoja et al. *Soft Actor-Critic Algorithms and Applications*. 2019. arXiv: `1812.05905 [cs.LG]`.

[18] Joshua Hare. *Dealing with Sparse Rewards in Reinforcement Learning*. 2019. arXiv: `1910.09281 [cs.LG]`.

[19] Mel Vecerik et al. *Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards*. 2018. arXiv: `1707.08817 [cs.AI]`.

[20] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533.