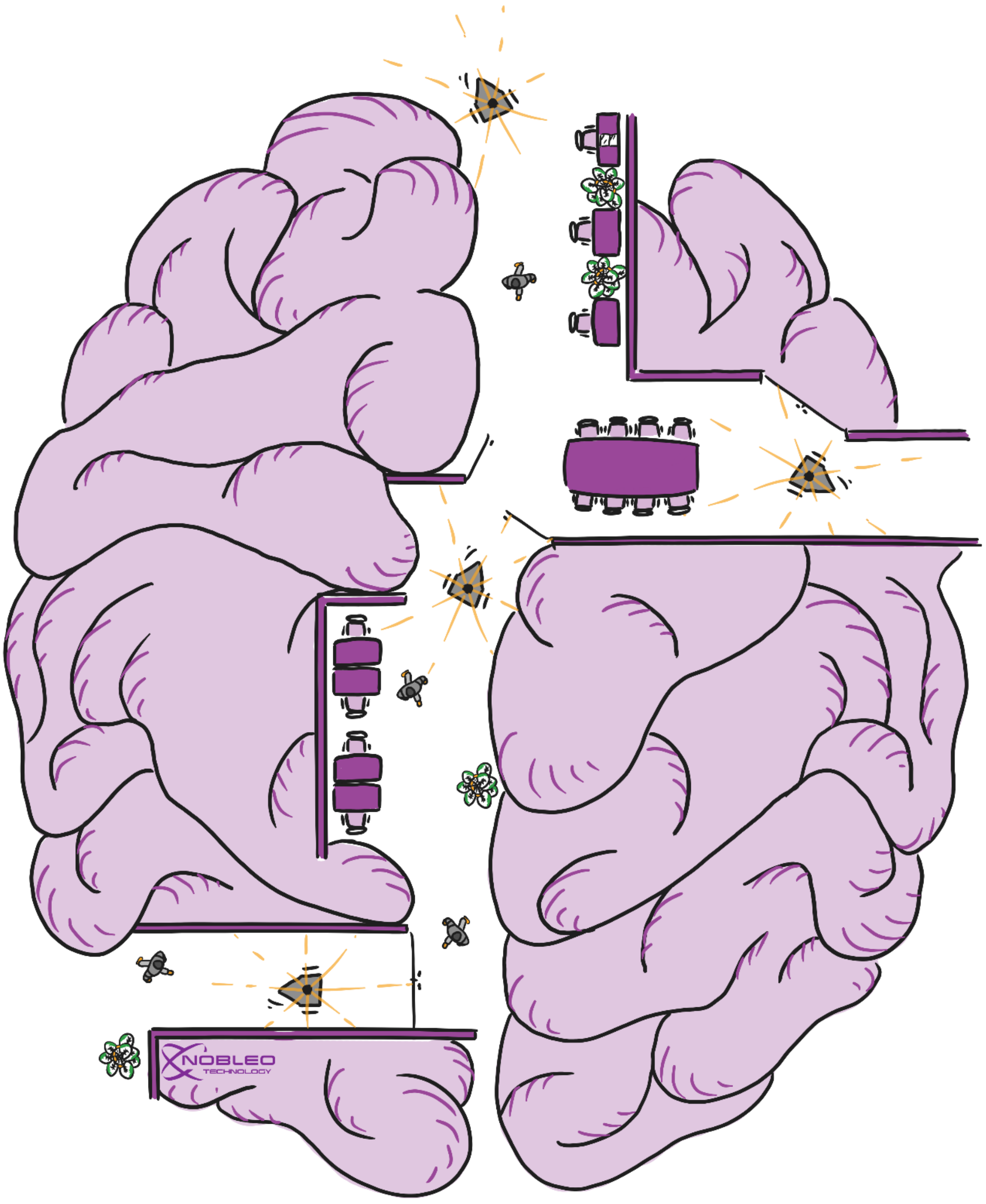
The self-learning robot

The future of intelligent navigation at scale?



Bram Odrosslij, Birgit Plantinga & Mukunda Bharatheesha

December 27, 2024

Introduction

In today's world, robots are becoming increasingly common in various industries, from airports to hospitals. Getting these robots to navigate efficiently through complex dynamic environments remains a challenge with scalability being limited due to high hardware cost.

Common approaches

Deterministic optimization approaches, commonly used in robot navigation, require engineers to fine-tune algorithms to handle a wide range of scenarios a robot may encounter. This process is both time-consuming and challenging due to the complexity of the problem. Robots employing such approaches often need expensive hardware to be able to run complex software. An investment that keeps adding when a robot fleet is expanded.

Learning-based approaches

In recent years, learning-based approaches to navigation have been actively researched and developed. Particularly, an AI discipline called *reinforcement learning* is used to enable a robot to perform a certain type of task by learning from experience. We developed a novel approach to navigation by applying the reinforcement learning principle, which we call the *self-learning robot*. We believe that it has the potential to outperform classical methods as it encapsulates manual tuning complexity with comprehendible reward functions.

While initial training may require significant computational resources, the resulting software can run on affordable hardware, specialized for AI applications like Google Coral's edge TPU devices [1] or Nvidia's embedded GPU platforms [2]. This makes it particularly valuable for businesses deploying multiple robots, as the learned behaviour can be easily replicated across an entire fleet.

Although some AI disciplines like computer vision are already widespread, reinforcement learning for robot control is only now making its first steps from research to industry, with pioneering applications in legged robotics by companies like Boston Dynamics [3].

In this article

We start off with the navigation fundamentals, followed by a comparison between commonly used deterministic optimization approaches and a learning-based approach to navigation. We then explain our novel approach with more details on the implementation in the subsequent section. After that, we demonstrate how our robot can swiftly navigate around walls and through cluttered environments. The article concludes with a discussion on future potential and applications of self-learning robots.

Navigation fundamentals

Global planning

Imagine standing in an office space, wanting to navigate to the coffee machine. Your brain automatically creates a general plan: "turn left at the conference room, walk past the desks, the coffee machine is at the end of the corridor." This mental mapping is what is known as a *global path.*

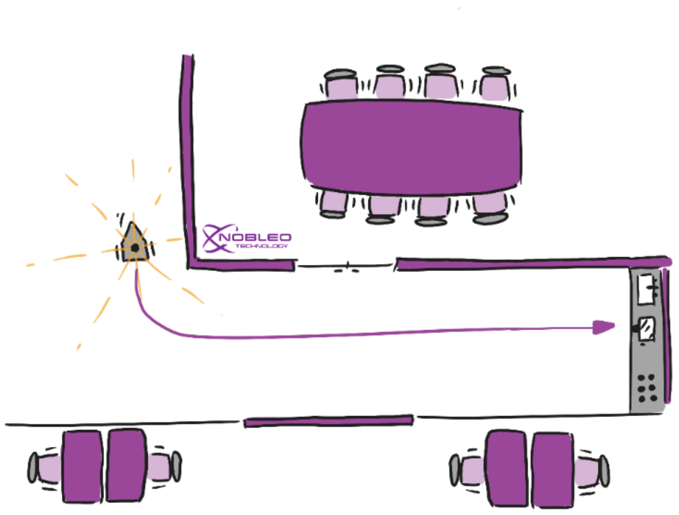


Figure 1: Mobile robot with lidar (orange) driving along global path (purple) toward the coffee machine in an office environment.

Local planning

While walking, you instinctively adjust to the dynamic office environment - stepping aside for colleagues or pausing as someone exits a meeting room. Generating these quick, reactive movements is called *local planning*.

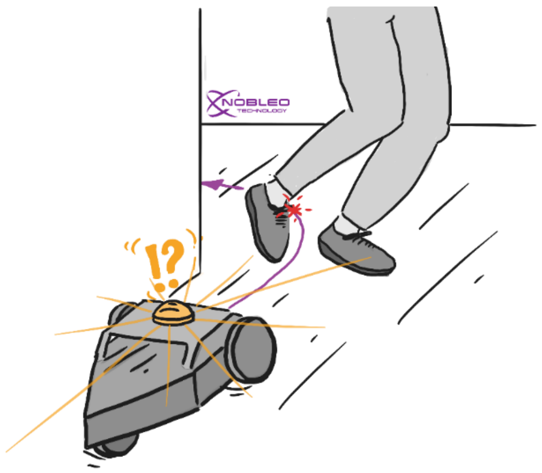


Figure 2: Mobile robot with lidar (orange) in a scenario necessitating local planning to prevent future collision (red) along global path (purple).

This article focusses on local planning in complex, dynamic environments.

Comparing local planning methods

Common approaches

Common approaches use deterministic optimization techniques to search for the best action at each point in time, based on a hand-tuned objective function.

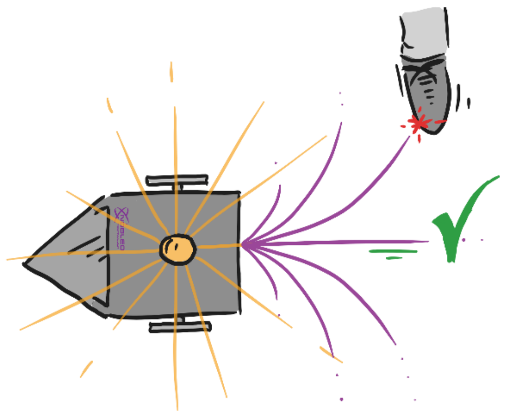


Figure 3: Selecting best action (green) based on forward simulated actions (purple).

Learning-based approach

Each timestep, the robot executes the action it has learned to be optimal for the current observation without evaluating numerous possible solutions.

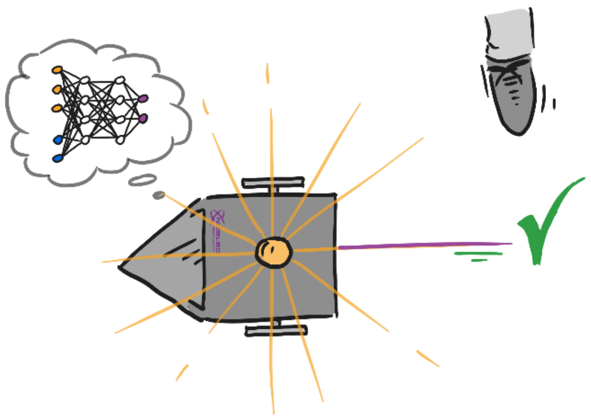


Figure 4: Direct action output based on learned behaviour.

Due to the complexity of the environment, the impact of the tuning decisions of the common methods becomes difficult to comprehend, resulting in a tedious trial-and-error process. These methods rarely include sufficient testing, making it challenging to understand the full implications of parameter adjustments. This ultimately leads to limited performance in real-world scenarios.

In contrast, the learning-based approach centres around testing, allowing desired behaviours to emerge naturally from real-world scenarios through comprehensive validation.

Moreover, common approaches often use real-time optimization to find the best action at each timestep, which can make them computationally expensive. The self-learning approach, when paired with modern AI hardware, can execute very efficiently, as it directly outputs learned optimal actions without the need for runtime optimizations.

Self-learning robot: General concept

In this section the general concept of the self-learning robot is explained. In the next section we explain our actual implementation of these concepts.

For a robot to learn how to drive to a goal on its own, the following needs to be defined:

- A robot
- The environments
- A reinforcement learning setup

Robot

A robot is defined by many aspects, but for the navigation tasks we focus on its physical dimensions, its constrained movement in space and its sensing abilities.

Environment

The environment usually consists of a map with a floor, walls and obstacles. It can be made as complex as you like. For instance, you could add moving obstacles, different textures and so on.

Reinforcement learning

The robot learns from experience using the reinforcement learning process. In short, it performs the following steps which are also visualized in Figure 5:

1. The robot receives data from its sensors which are used to generate *observations*.
2. These observations are fed into a neural network, which is defined by an architecture and its parameters. For each set of observations that the network receives, it returns an *action*.
3. The robot performs the action and receives a *reward*. The rewards are positive for good behaviour (such as driving toward the goal) and negative for bad behaviour (such as driving into a wall). Defining the reward correctly is a crucial factor in learning the desired behaviour.

These steps are repeated iteratively and after a specified number of steps, the reinforcement learning algorithm uses the observations, actions and rewards to update the parameters of the network in such a way that the network improves. This is repeated until the network no longer improves.

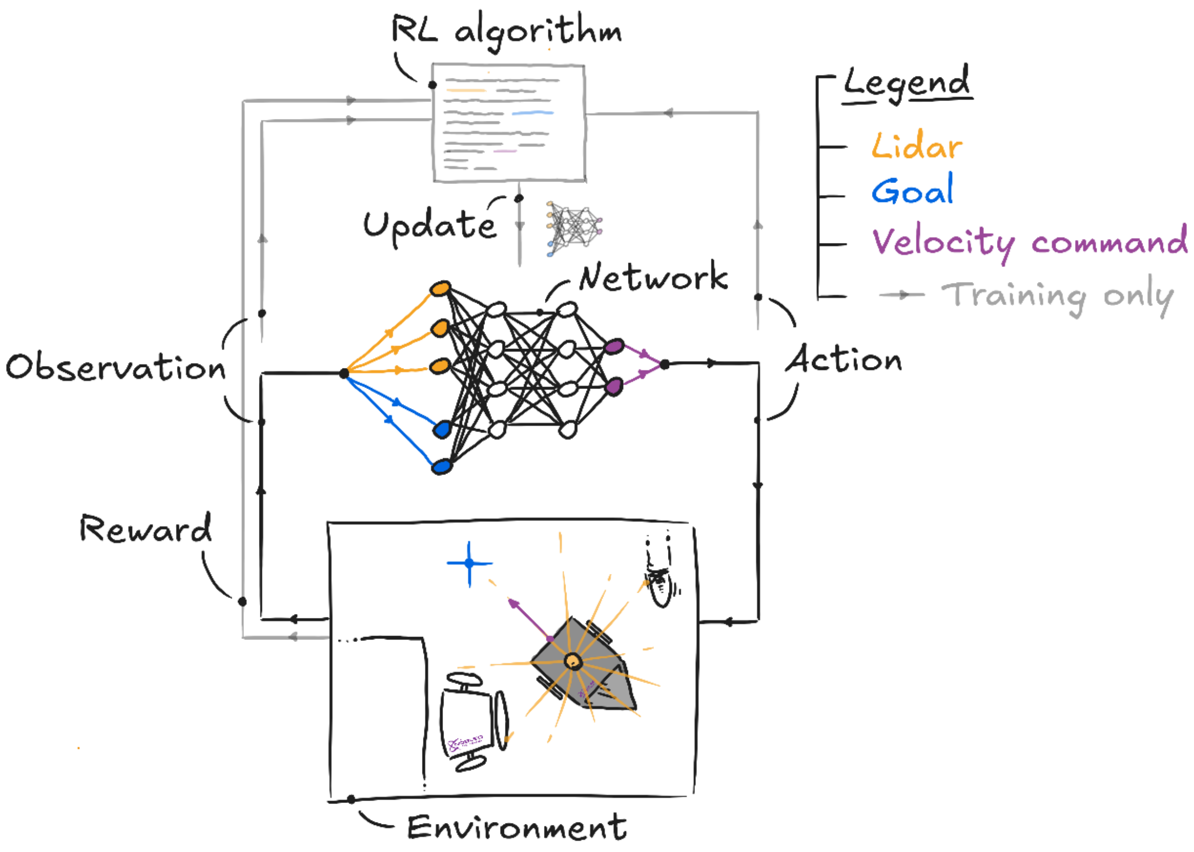


Figure 5: Diagram of the reinforcement learning training procedure (grey and black arrows) and inference process (black arrows only).

Self-learning robot: Practical implementation

In this section we explain how the general concepts from the previous section are implemented to get our robot Cindy to learn how to drive to a goal on its own.

Robot: Cindy

Cindy is a small differential drive robot with a lidar developed by Nobleo Technology mainly for educational, continuous testing and development purposes.

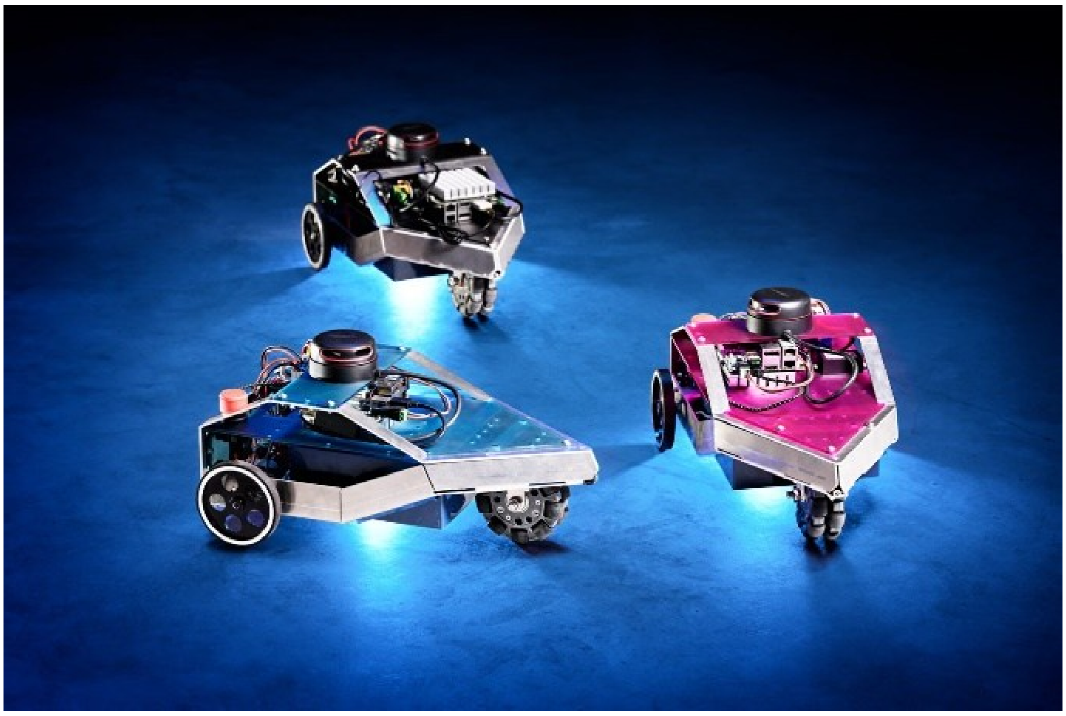


Figure 6: Nobleo's differential drive mobile robot named Cindy.

As a first step we are simulating it as an infinitely small differential drive robot with a 2D lidar sensor that returns 64 range measurements over 360 degrees.

Environment

In our demonstration we use two different sets of maps consisting of solid obstacles depicted in Figure 7:

- Wall maps: these consist of solid walls of varying lengths that the robot needs to drive around to reach its goal.
- BARN maps [4]: these consist of highly cluttered obstacle configurations.

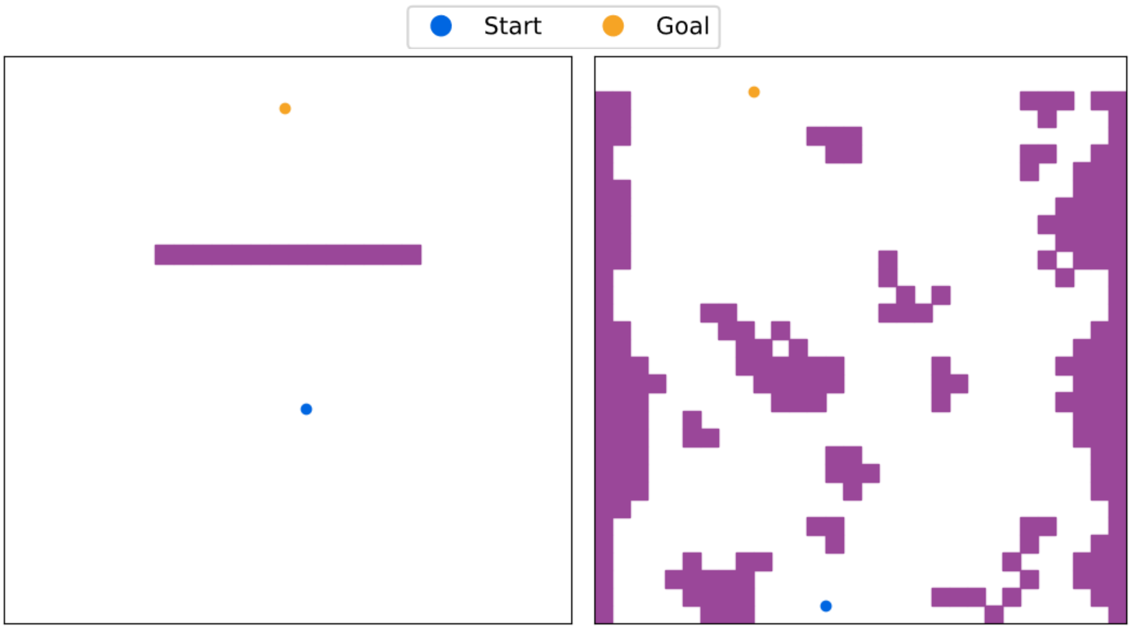


Figure 7: Examples of a wall map (left) and a BARN map (right).

Reinforcement learning

We train the robot to navigate through each of these map sets separately. This is done in simulation, which allows us to quickly train in many different scenarios.

The implementation of reinforcement learning setup discussed in the previous section is depicted in Figure 5 and can be summarized as follows:

- The *observations* consist of the lidar measurements, and the goal position with respect to the robot.
- The *action* is a command velocity that consists of a forward velocity and an angular velocity component that provide simple instructions on how fast to go forward and how fast to turn.
- The *reinforcement learning algorithm* used is Stable-Baselines3's implementation of the Proximal Policy Optimization (PPO) [5].
- The *reward* is designed such that it encourages the robot to drive to the goal with an efficient route (good behaviour), without colliding with obstacles (bad behaviour):
    - Small reward/penalty proportional to its progress towards or away from the goal.
    - Large reward for reaching the goal.
    - Even larger penalty for colliding with walls or obstacles.

For more technical details on the implementation please see Appendix A.

Self-learning robot: Performance showcase

To showcase the performance of our self-learning robot, we split the datasets into train and validation sets as seen in Table 1. The validation maps are excluded from the training process.

Table 1: Training and validation set.

	Nr. of training maps	Nr. of validation maps
Wall maps	179	37
BARN maps	240	60

It takes about 3 to 4 hours for a training session before the network stops improving when training is performed on a laptop with a 13th generation Intel i7 processor, an RTX A1000 GPU and 32gb of RAM.

In Figure 8 and Figure 9 below, you see examples of paths driven by the robot in the maps belonging to the respective datasets.

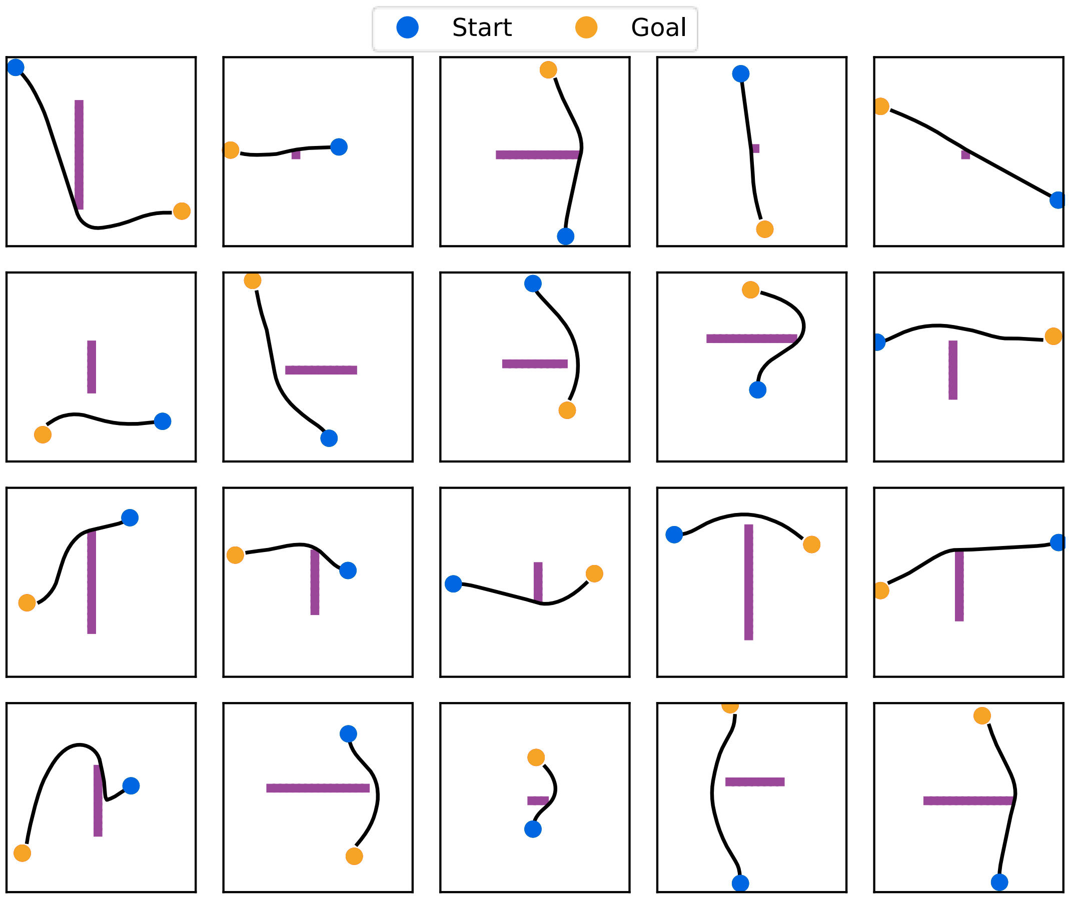


Figure 8: Validation results in maps belonging to the wall dataset.

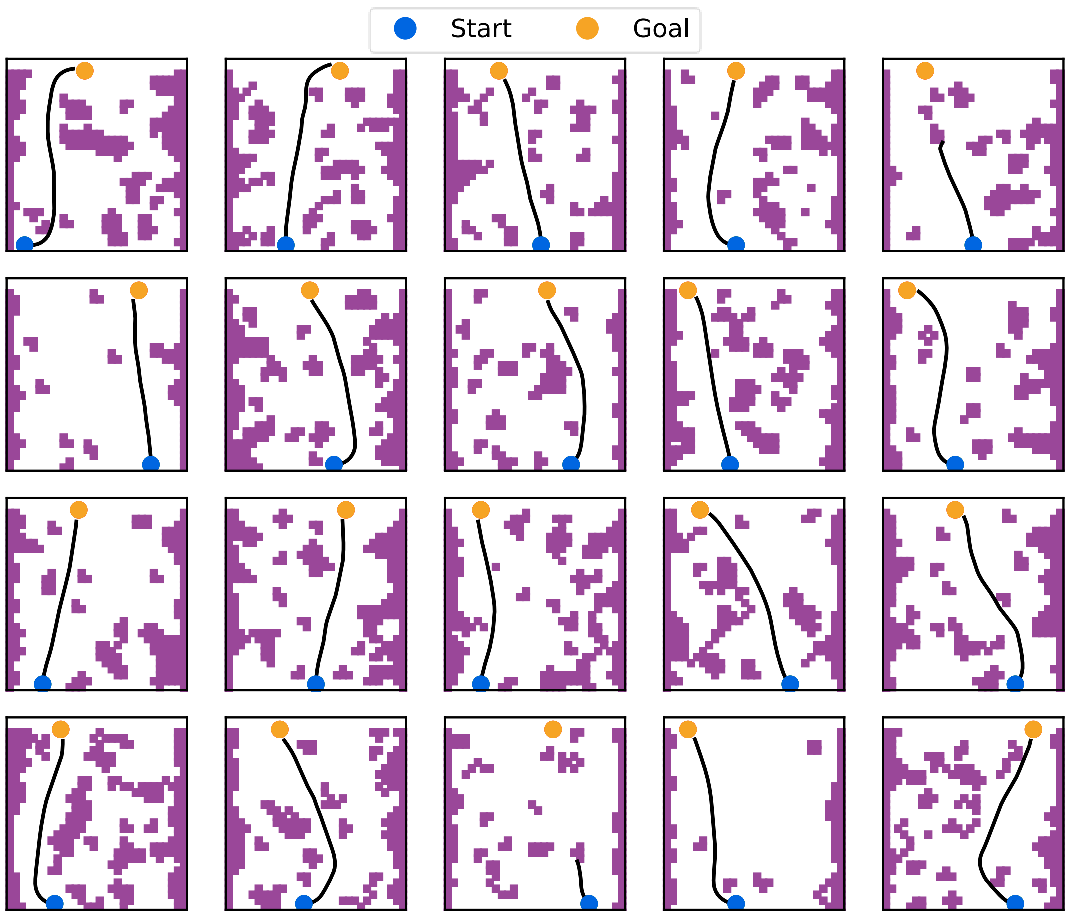


Figure 9: Validation results in maps belonging to the BARN dataset.

The results in Figure 8 and Figure 9 show that in both cases the robot takes a fast and logical route to its goal and carefully drives around obstacles.

Although it may seem as if the robot is cutting corners and driving through obstacles, this is not the case. A closer inspection of the maps reveals that the robot is not actually navigating through obstacles. Due to its small size, the robot can pass by objects at a very close distance.

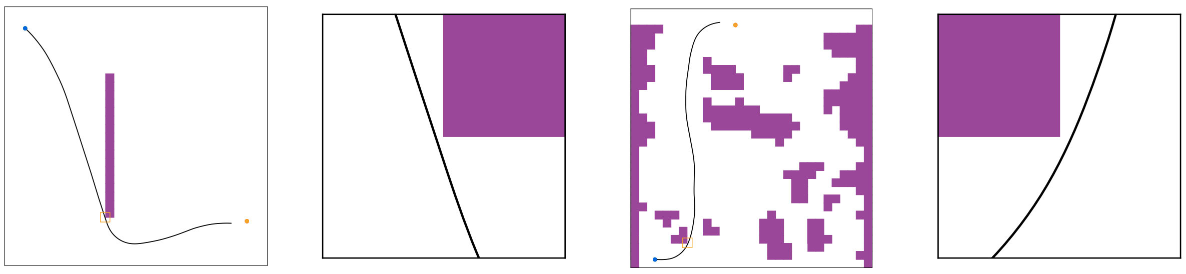


Figure 10: Zooming in on the upper left maps of the examples in Figure 8 and Figure 9 shows that the robots passes the obstacles at a close distance without colliding.

In Table 2 the success rates and average inference times are shown. The robot successfully reaches its goal in all the wall maps and successfully does so in 91.7 % (55 maps) of the BARN maps.

Table 2: Performance of our self-learning robot. The success rate is the percentage of maps in which the robot reaches the goal. The inference time is the time that it takes for the model to predict the next action based on its observation, on a regular laptop CPU.

	Nr validation maps	Success rate	Average inference time
Wall maps	37	100 %	0.2 ms
BARN maps	60	91.7 %	0.2 ms

In the BARN map examples shown in Figure 9, the robot collides once. This can have several causes, such as the reward putting too much emphasis on driving to the goal, the training data not being representative enough, the robot being entrapped and so on.

Closing words

Reinforcement learning for robotics is steadily gaining traction in the industry, thanks to its ability to simplify complex environmental modelling into understandable reward functions. One of the key advantages of reinforcement learning-based methods is the ability to explicitly define desired behaviour, unlike classical methods where the expected behaviour is typically implicit.

In our example, the robot navigates efficiently even though we never specified how it should interpret its lidar data or determine its velocity in different situations. It learned all of this on its own.

As mentioned earlier, reinforcement learning-based methods can run on relatively affordable AI hardware, such as Nvidia Jetson boards [2] and external TPU's [1]. This reduces the cost per robot and makes large-scale deployment more feasible.

Future work and opportunities

Looking ahead, our next steps involve scaling up the simulated robot size and accurately modelling lidar noise and time delays for a smoother transition to the physical world. After that, we will deploy the network on a physical robot and fine-tune the training parameters.

Given the progress AI has made in other areas, such as generative AI (e.g., ChatGPT) and image recognition, we believe that self-learning robots will also become more prominent in the future. They will be especially beneficial for two specific use cases: environments with large robot fleets and robots operating in complex, dynamic environments.

Curious?

We think that reinforcement learning for robot navigation is an innovative and promising technique that should be closely monitored in the coming years. If you're interested in its potential applications or would like to learn more about our implementation, feel free to contact us at:

bram.odrosslij@nobleo.nl

References

[1] "Coral products," Coral, [Online]. Available: https://coral.ai/products/. [Accessed 19 12 2024].

[2] "Jetson Modules," [Online]. Available: https://developer.nvidia.com/embedded/jetson-modules. [Accessed 18 12 2024].

[3] "Boston Dynamics," [Online]. Available: https://bostondynamics.com/blog/starting-on-the-right-foot-with-reinforcement-learning/. [Accessed 18 12 2024].

[4] D. Perille, A. Truong, X. Xiao and P. Stone, "Benchmarking Metric Ground Navigation," in *2020 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, IEEE, 2020.

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford and O. Klimov, "Proximal Policy Optimization Algorithms," arXiv, 2017.

[6] "Github - Stable-Baselines3," German Aerospace Center (DLR) - Institute of Robotics and Mechatronics (RM), [Online]. Available: https://github.com/DLR-RM/stable-baselines3.

[7] "Github - Hydra," Facebook Research, [Online]. Available: https://github.com/facebookresearch/hydra.

[8] Webots, "Open-source Mobile Robot Simulation Software," Cyberbotics Ltd., [Online]. Available: http://www.cyberbotics.com.

[9] "Github - Shapely," [Online]. Available: https://github.com/shapely/shapely.

APPENDIX A. TECHNICAL DETAILS

This appendix describes the technical specifications of the implementation of the self-learning robot.

Reinforcement learning algorithm	
Policy gradient method	PPO
PPO parameters	
learning_rate	0.0003
n_steps	2048
batch_size	64
n_epochs	10
gamma	0.99
gae_lambda	0.95
clip_range_vf	null
normalize_advantage	true
ent_coef	0.0
vf_coef	0.5
max_grad_norm	0.5
use_sde	false
sde_sample_freq	-1
nr of environments	6
policy	MlpPolicy
activation_fn	ReLU
net_arch	
pi	[256, 256, 256, 256, 256]
vf	[256, 256, 256, 256, 256]

Learning parameters	
total_time_steps	4.000.000
n_eval_episodes	25
eval_freq	10.000 / 6
deterministic	False
reset_num_timesteps	False
nr of environments	6

Robot simulation	
sample time	0.05 s
kinematics	
max forward velocity	0.2
min forward velocity	-0.2
max forward acceleration	0.3
min forward acceleration	-0.3
max angular velocity	0.6
min angular velocity	-0.6
max angular acceleration	0.8
min angular acceleration	-0.8
dimensions	
footprint	[[-0.001, 0.0] [0.0, $\sqrt{3}/100$] [0.001, 0.0]]
lidar	
position	[0.0, 0.0]
number of measurements	64
angular range	360 degrees
linear range	0.05-3.0 m
frequency range	20-40 Hz

Software packages	
Reinforcement learning	Stable baseline3 [6]
Configuration management	Hydra [7]
Simulation	Webots [8]
Geometry library	Shapely [9]

Rewards	
at goal	250
collision	-1500
linear*	10.0 * relative progress towards goal

*The relative progress towards the goal is computed as the projection of the progress onto a vector pointing from the robot to the goal. It is therefore positive when its distance to the goal decreases and negative when its distance to the goal increases.