

Mapless Navigation:

A reinforcement learning approach

Damian Schori

Master Thesis

Supervisor: Prof. Dr. Dejan Šeatović

ILT, Institute for Lab Automation and Mechatronics

OST, Eastern Switzerland University of Applied Sciences

damian.schori@ost.ch

Abstract

Autonomous navigation from A to B in indoor environments is a widely researched field. Many known approaches are based on a mapping of the entire environment in order to calculate a path through the space in advance. This Master thesis proposes a new approach based on reinforcement learning. It shows that a system can navigate by only using sensor data and system pose relative to a target. The method bases on the Robot Operating System (ROS) and Gazebo for simulation and training. A test vehicle with Ackermann steering was built to test the proposed method and compared with a state of the art navigation method in a real world scenario.

Evaluation experiments have shown that the proposed method is outperformed by SLAM based solutions, in terms of pose accuracy, location precision and the reliability of achieving the desired target. Though, the developed procedure offers potential in cases where prior mapping is not possible and exploration capability of a vehicle is required.

1 INTRODUCTION

The problem of autonomously navigating a robot or vehicle in an unknown environment while avoiding obstacles is an important and challenging task. In many cases (such as delivery robots or service robots), it is not possible to fully comprehend the environment in advance in order to create an optimal path planning. Path planning and static obstacle avoidance in such problems are often described as *mapless navigation* (Giovannangeli et al., 2006; Tai et al., 2017).

Many state of the art algorithms for autonomous navigation contain two parts. First, a map of the environment is created with the use of simultaneous localisation and mapping (SLAM) (Hess et al., 2016) algorithms. Second, a trajectory is calculated based on the previous created map with the goal of a collision-free path and minimal driving time.

When thinking about this task on a human level, it is possible for us to navigate through an unknown environment to a target location based on learned schematics without knowing the whole environment we are in. We have learned from situations on how to get from A to B.

Reinforcement Learning (RL) achieved much success in tasks including video games (Mnih et al., n.d., 2015) or control tasks (Lillicrap et al., 2019). In such tasks, Agents are trained in simulation where they try to maximize a given reward function by optimizing a policy using actions in the corresponding environment. The main advantage here compared to methods based on machine learning is that it is not required to have any hand-labelled data. Moreover, such tasks are mostly trained and evaluated in simulation on the same environment, which makes it difficult to determine if they would success under real world conditions. Therefore, it is still the subject of current research to

show whether it is possible to adopt with RL learned behaviour in certain situations to new environments. Due to the mentioned data inefficiency, it is not practical to train an Agent navigating to a target position in a real-world, as it would require lots of human interventions by e.g. resetting the vehicle after a crash and moving it back to the starting position.

Therefore, in this thesis an approach is presented to train a navigation strategy with the help of RL in a simulated environment and then to deploy and test it on a real vehicle without adjustments or fine-tuning.

1.1 Goal

Based on the challenges the following objectives were set:

- A state of the art path-planning algorithm, which is aware of the full map of the environment, has to be implemented on the car and tested. With this algorithm, the vehicle has to be able to autonomously navigate from A to B and avoiding obstacles. This serves as a baseline to compare the RL method.
- A new approach based on a RL method has to be developed which is only aware of its local surroundings and the target location.
- The RL method has to be trained in a simulated environment and evaluated in both simulation and real world.
- Both approaches, the state of the art path-planning algorithm and the RL approach, are compared with each other.
- To test the proposed methods a carlike vehicle based on Ackermann steering has to be developed containing the necessary sensors and actuators to observe and navigate through indoor environments.

2 RELATED WORK

2.1 Reinforcement Learning

In contrast to traditional machine learning, RL tries to maximize a reward claimed by an environment rather than finding a hidden structure in labelled data. In RL, an Agent gathers experiences by taking actions in the environment, which yields certain rewards, and uses these experiences to form a policy for the most rewarding behaviour in its environment.

In recent years, several different approaches have been developed, such as deep Q-learning (DQN) which learns an estimator $Q_{\theta}(s, a)$ for an optimal action-value function. In (Mnih et al., 2015) researchers from the Deep-Mind team were able to build a reinforcement learner based on DQN which is able to beat human level performance in various Atari video games. (Schulman et al., 2017) introduced a new on-policy algorithm called Policy Proximity Optimization (PPO), which tends to be more robust regarding hyperparameter settings.

RL has been widely applied in games and robotic tasks. In (Lillicrap et al., 2019) DQN was extended to the continuous action domain to successfully solve various simulated tasks such as the CartPole swing-up control task or car-driving. In (Silver et al., 2016) an approach based on value and policy network was developed to solve the challenging game of Go.

In (Zhang et al., 2017) a solution was provided to navigate a robot through an real environment trained with a DQN based method on depth images as state observations. In these examples, it has been demonstrated that it is possible to train Agents without the need of any hand-labelled samples and to solve challenging tasks by choosing well-defined reward functions and having the necessary amount of computing power.

However, RL also tends to be very sample inefficient. For example, in (Hessel et al., 2017), it has been shown on an Atari game running at 60 frames per second that it is necessary to run 18 million frames in order to reach human-level performance. This corresponds to about 83 hours of game-play experience. For humans it is possible to learn the game within a few minutes of gameplay.

In the case of the Atari game, the agent receives an RGB image with size (210, 160, 3). This means that the Agent 'sees' a vector of size 100'800. Combined with the possible actions the Agent can take, this results in a rather high feature space.

In order to reduce the size of this observation/ action space, neural networks are used as a feature extractor e.g. with convolutional layers in this case of an image as observations to reduce the state space.

Another challenge remains in modelling an appropriate reward function. It is a key component in the learning process and significantly determines the success of an Agent or not. In a deterministic reward function, it is mostly not possible to give the most "appropriate" reward to a given situation. An interesting future direction of designing reward function is not to code them "physically" but to learn the reward function in sense of a network. In (Christiano et al., 2017) such reward modelling is proposed which requires minimal human interaction by deciding after some steps between two examples which performs better. This combines the "judging power" of humans into the processing power of Agents running on modern computers.

2.2 Mapless navigation

Motion planning aims to navigate a vehicle from the current position to a desired target position without colliding with obstacles on its path. In current methods, first, an obstacle-map of the current environment is constructed with simultaneous

localization and mapping (SLAM) (Durrant-Whyte & Bailey, 2006) with the help of laser-scan data. Then, with the combination of well-established path planning algorithms such as described in (Roesmann et al., 2012; Rösmann et al., 2013), it is possible to navigate through the previously mapped environment without colliding into static or dynamic moving objects. However, the weakness remains that it involves a significant amount of time to drive around and map the environment in advance.

Mapless navigation such as described by Zhelo et al., (2018) attempts to autonomously navigate a robot through an unknown space to a predefined target. In (Jin et al., 2020) a solution is presented in which a robot moves through obstacles and moving objects towards a target position only with the help of the laser scan and its position relative to the target. Similarly, (Tai et al., 2017) explains a method of navigating autonomously in an unknown environment using sparse laser scans and also the relative position of the robot to the target position. They want to show that the sparse laser scan measurements could also be replaced with cheaper sensors such as ultrasonic distance sensors.

2.3 From simulation to real world

Agents in RL learn from situations which action leads to the greatest reward for a given situation. This includes that the Agent only learns to deal with the situations it has seen at training time. Thus, to successfully deploy a trained Agent to a real world scenario, it must be ensured either that the training setup corresponds as closely as possible to the test setup, or that the training "variance" is kept large enough so that all possible scenarios occur in it. Overall, there are only a few examples, which showed the successful deployment of RL methods trained in simulation into real world scenarios. In (Osiński et al., 2020) an application which trained a

RL method to control a full-size vehicle in simulation is described and tested on a real-world scenario. In (Capasso et al., 2020) the problem of lacking domain adaption between simulated and real-world data is investigated. To overcome this problem, an approach that is based on training the Agent in multiple environments simultaneously which shows increasing generalization capabilities, is presented.

3 METHODOLOGY

3.1 Notation

- "Agent", describes the software agent used in the RL approach which is responsible for training and predict the corresponding actions given states.
- "TEB", describes the state of the art navigation approach based on timed elastic bands.

3.2 Test vehicle

In order to test the proposed approaches, a test-vehicle has been constructed. The main chassis is built up from an RC-crawler kit consisting all-wheel drive and Ackermann steering. The steering is controlled by a servo, which is mounted in between the front wheels. To mount the sensors and the Nvidia Jetson a construction based on 3D Printing was manufactured.

3.2.1 Hardware specifications

The vehicle is also operated with the following components:

- **Nvidia Jetson Nano:** Serves as the central computing element on the vehicle. All sensor data are recorded and evaluated here, and commands for the actuators are generated. The Jetson Nano is equipped with a USB WIFI dongle for communication with external devices.
- **Velodyne VLP-16:** A 360 ° laser scanner is used for room and obstacle sensing. With an angular resolution of 0.1 ° and a scanning frequency of up to 20 Hz, a detection with a resolution of approx. +- 3 cm is possible. Data exchange takes place via UDP packets with the Jetson Nano.

- **Realsense T265:** The T265 tracking camera includes two fisheye camera lenses, an IMU with 6 degrees of freedom and a visual processing unit for onboard data processing. The camera calculates 3D odometry data on-board with the cameras and the IMU. In addition, the individual sensors can also be read out separately. The T265 contributes to localization in the environment, and the IMU is used to support the mapping algorithm. The data exchange takes place via a USB.
- **PCA9685:** The PCA9685 is a servo driver with 16 channels. This allows the motor controller and the servo for steering to be controlled from the Jetson Nano via the I2C protocol.
- **Power supply:** A 3-cell LIPO battery serves as a power source. The battery supplies a voltage between approx. 9.5 ... 12.5 V. This allows the laser scanner and the motor controller to be operated directly. For the Jetson Nano and the PCA9685, a DC-DC converter is connected ahead which supplies 5 V output voltage.

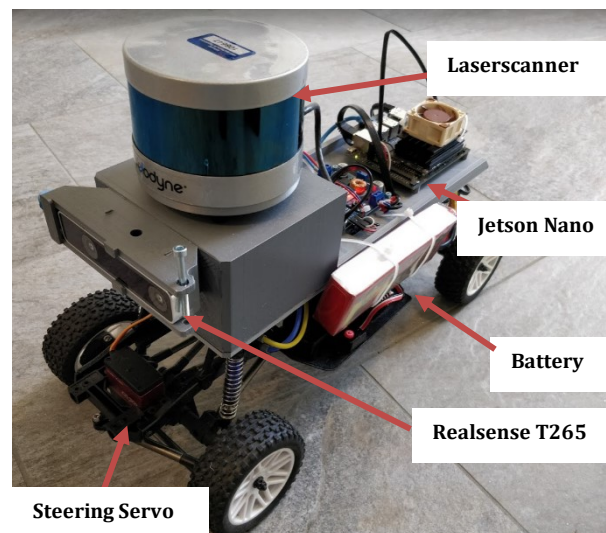


Fig. 1: Test vehicle with Ackermann steering based chassis, the necessary sensors and Nvidia Jetson device for computing

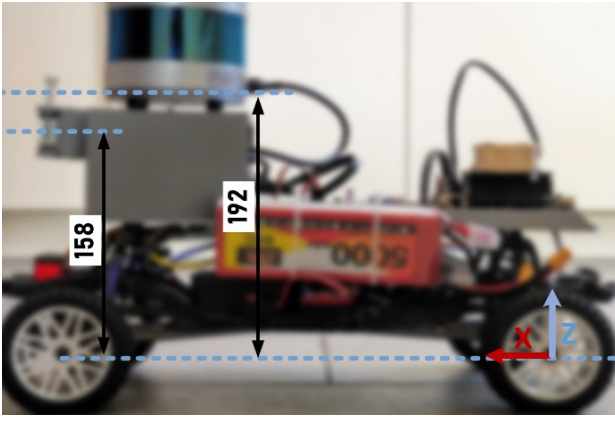


Fig. 2: Side view of test vehicle (values in mm)

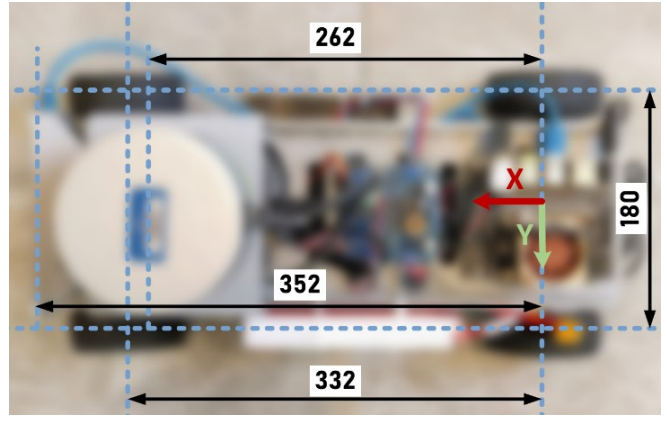


Fig. 3: Top view of test vehicle (values in mm)

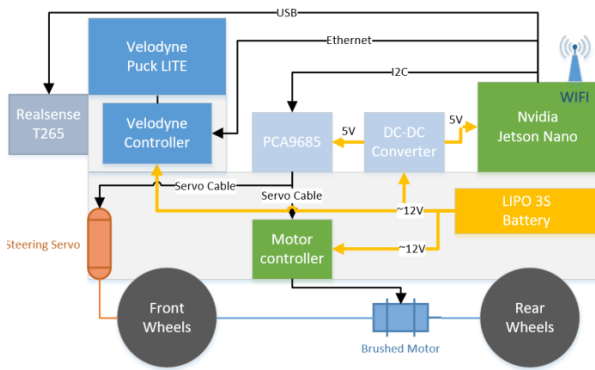


Fig. 4: Wiring schematics of the test vehicle

3.2.2 ROS TF

The test vehicle contains various sensors and actuators. In order to be able to map the geometric structure and constraints, there is the Transform (TF) Package in ROS. TF makes it possible to keep track of multiple coordinate over time. TF maintains the relationship between coordinate frames in a tree structure and provides functions to transform those coordinate frames. For the test-vehicle, a TF-tree was constructed as in Fig. 5.

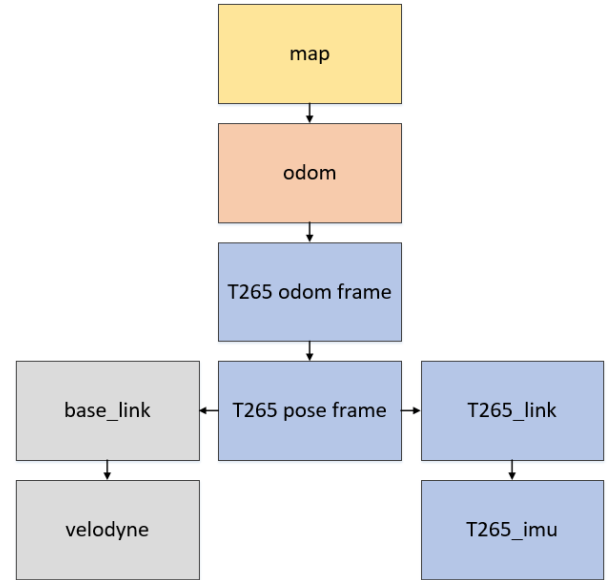


Fig. 5: TF-tree of the test-vehicle

3.2.3 Motor control

The brushed DC motor used tends to respond poorly at low speed and fall into halt. To overcome these problems, a closed loop speed control was implemented with the Realsense T265 serving as the speed sensor.

The first control parameters were determined using the second method of Ziegler-Nichols. The procedure was as follows:

1. The system was set up with a closed loop with only the proportional controller active.

2. Starting with a low gain value K_{cr} this was increased step by step until a steady-state oscillation was reached ($K_{cr} = 3$)
3. Then period P_{cr} was measured from the recorded graph of the speed measurements:

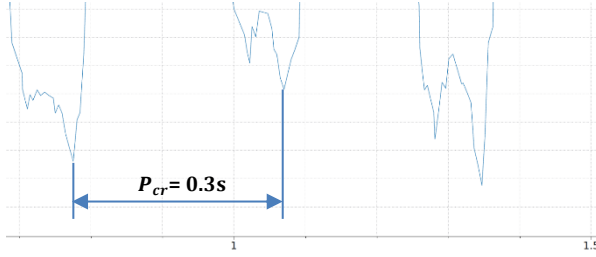


Fig. 6: Speed oscillation after increasing K_{cr} to 3

4. According to the gain estimator chart K_p and T_i were calculated as follows:

$$K_p = 0.5 * K_{cr} = 1.5$$

$$T_i = \frac{1}{1.2} * P_{cr} = 0.25$$

These values provided a working behaviour. However, errors to the target speed were adjusted only slowly; increasing T_i to 2.0 solved this problem.

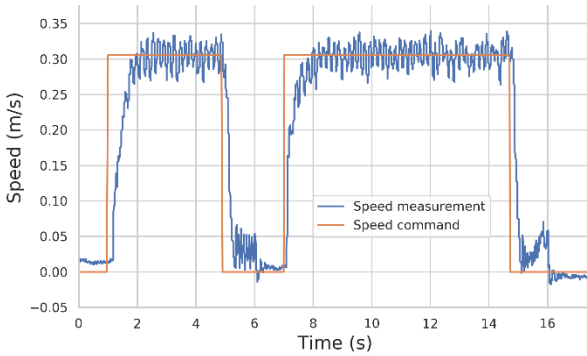


Fig. 7: Motor control: speed response for a given command with applied speed control

3.3 Mapping and localization

Although the work suggests that there is no use of a full map to navigate with the proposed RL approach, a previously created map of the test environment is used to localize the vehicle. This method of localization however could be replaced by other localization methods such as wheel odometry.

For the task of creating a map for the test-environment, simultaneous localization and

mapping (SLAM) is applied with the Google Cartographer (Hess et al., 2016) system. The system provides ROS integration, which makes it easy to use with the other components. It consists of a local and global SLAM algorithm. Local SLAM inserts a new scan into its current sub-map construction by scan matching while the global SLAM arranges sub-maps between each other so that they form a coherent global map (see Video link in chapter 4).

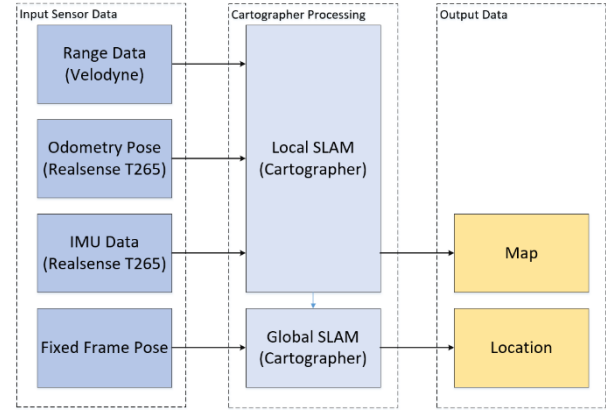


Fig. 8: Scheme of the SLAM algorithm using Google Cartographer and Laser range data from the Velodyne VLP-16 Sensor combined with odometry and imu data from the Realsense T265.

As the laser range data is slowly updated (10 Hz) compared to the other sensors available, additional sensor measurement such as odometry- and imu data (200 Hz) can be feed into Cartographer. This helps to overcome the problem of losing the current position or orientation when taking fast turns or movements.

3.4 Teb local planner navigation

As a "traditional" navigation approach the "TEB Local Planner" (TEB) is used. A plugin for the ROS framework is implemented in (*Teb_local_planner - ROS Wiki*, n.d.). The package implements an online optimal local trajectory planner for the navigation and control of mobile robots based on timed-elastic-bands. The initial trajectory generated by a global planner is optimized at runtime with respect to

minimizing the trajectory execution time (time-optimal goal), the distance to obstacles and the compliance with kinodynamic constraints such as meeting maximum velocities and accelerations.

In this work, the TEB algorithm is used as a baseline navigation approach primarily to compare the RL approach in the real-world experiments.

3.4.1 Hardware setup

For the optimal path planning with the TEB algorithm two steps are necessary:

- First, a map is created using the laser scan and the Cartographer (SLAM) algorithm by driving through the environment.
- Second, using the created map of the environment, the vehicle is localized with the AMCL algorithm (*Amcl - ROS Wiki*, n.d.). This method localizes the vehicle using a particle filter based on the current laser scan in the given map. Once the position and the environment are known, a path through the environment can be searched and calculated with the TEB Planner for a given target.

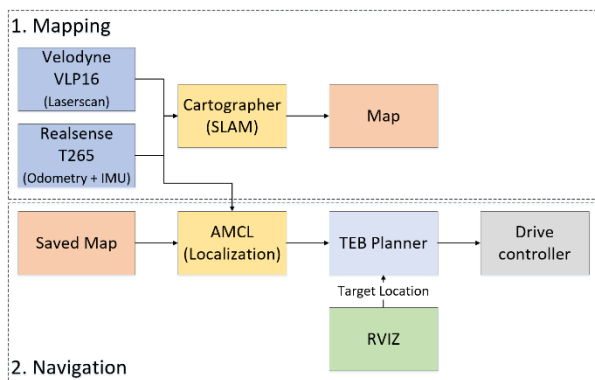


Fig. 9: Hardware and ROS setup for the TEB navigation approach

3.5 RL navigation

For the part of the RL based navigation method, a simulated environment with ROS and Gazebo was built. An Agent based on the proximal policy

optimization method was used which tries to maximize the output of the defined reward function.

3.5.1 Simulated vehicle

For the simulated vehicle, the vehicle from the MIT Racecar project was used. The dimensions and sensor positions were adjusted to match the test vehicle.

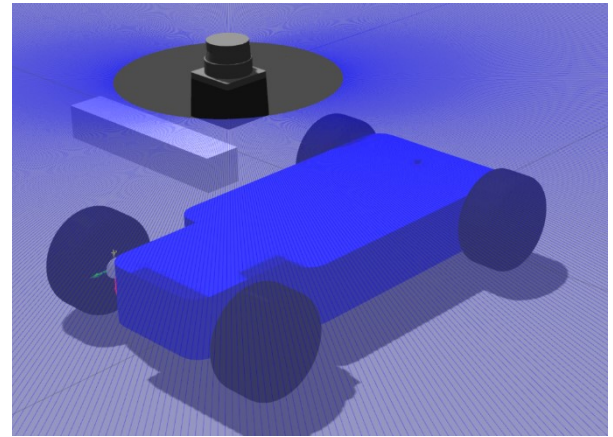


Fig. 10: Simulated vehicle in Gazebo

3.5.2 Proximal Policy Optimization

In supervised learning, it is relatively straightforward to set up a cost function for a given problem and minimize it using a gradient descent method. If the problem is solvable, there is a good chance of success with relatively little hyperparameter tuning.

However, the route to success in reinforcement learning is not as obvious, e.g., the algorithms have many moving parts that are difficult to debug, and they require a substantial effort in hyperparameter tuning in order to get pleasant results. Here, Proximal Policy Optimization (PPO) (Schulman et al., 2017) comes into place. PPO seeks a balance between ease of implementation, sample complexity and simplicity of tuning by attempting to calculate an update at each step that minimizes the cost function while ensuring that the deviation from the previous policy is relatively small.

In this work an implemented version of PPO as in (Liang et al., 2018) was used. Here, an Agent tries to maximize the output of a given reward function by optimizing a policy with the use of an actor and critic network.

3.5.3 Problem formulation

The problem is formulated as a Markov Decision Process (MDP) defined by the tuple (S, A, P_a, R_a) where S is the state space, A is the action space, P_a is the transition function and R_a is the reward function. This corresponds to the following:

- The **state space** S consists of a stacked constructed image based on the laser scan S_l and the robot position relative to the target S_p (see 3.5.4 State (Observation) for details) which gives information about the surrounding obstacles, distances to target and orientation to target.
- The **action space** are discrete values of 0 (turn to the right), 1 (drive straight) and 2 (turn to the left). The speed of the car is fixed and cannot be changed by the Agent.
- The **transition function** is deterministic and is given by physical laws in the simulation or in real world.
- The **reward function** is determined by several conditions based on the given observations (see 3.5.5 for details)

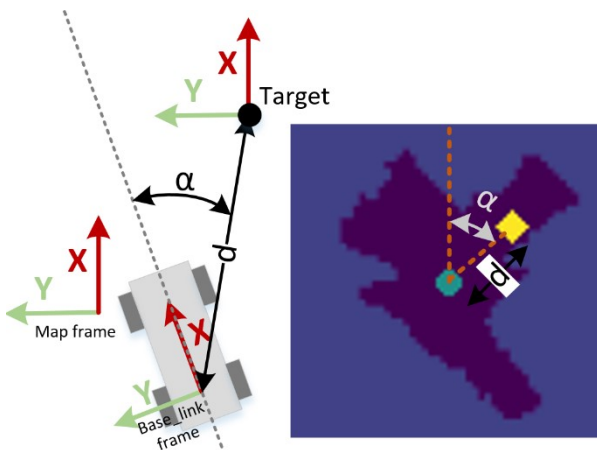


Fig. 11: Coordinate frame representation to compute the target position in the state

3.5.4 State (Observation)

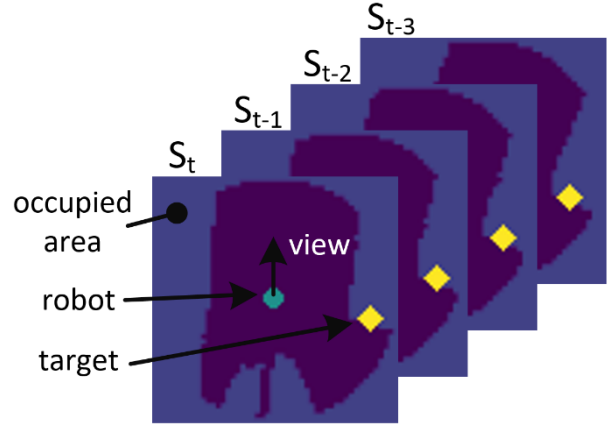


Fig. 12: State represented as an image with four time channels. The cyan circle in the middle corresponds to the vehicle's position where as the yellow diamond corresponds to the target position

The state $S_t = [S_t, S_{t-1}, S_{t-2}, S_{t-3}]$ is represented as an array of four grey-scaled images containing the state of the current time step t and of the last three time steps. Each state is encoded as an image (Fig. 12) with size $84 \times 84 \times 4$ (height, width, time steps). The image represents a spatial area of 10×10 m, which corresponds to a pixel size of approx. 12×12 cm in real world. The image is based on the laser scan sensor data S_l and the robot's position relative to the target S_p . This allows to visually represent the following:

- Information about the surrounding area in a radius of 5 metres. Whereas light blue areas denote obstacles or walls and dark blue areas denote free spaces.
- The position of the vehicle (cyan circle) relative to the surrounding area.
- The position of the target position relative to the current vehicle position. This indicates the distance as well as the orientation to the target. If the vehicle is further than 5 meters away from the target, then the target is drawn on the edge of the picture in the corresponding direction.

- Time dependent information can be extracted from the stacked states, such as:
 - Current direction of driving
 - Current direction of moving obstacles

The encoding of the sensor data has the advantage of being easily interpreted by a human. This helps in debugging situations and it is possible to encode all relevant information into the same format.

To encode the target position (yellow diamond) into the state image, the angle and the distance between the target coordinate system (same as the map frame) and the base_link frame of the vehicle is computed (see Fig. 11).

3.5.5 Reward function

An important part of RL is the reward function. The Agent mainly tries to maximize this function. Based on this function, the critic network is trained directly. It is therefore helpful to give the Agent as many "hints" as possible about which action is successful or not in the current state.

As an example: If the Agent only gets a reward when the vehicle arrives at the target position and gets a penalty in case of a collision, the Agent will take a very long time to register its first successes. Therefore, it is more helpful to give intermediate rewards for example if the vehicle gets closer to the target and in opposite to give negative rewards when it moves away from the target again.

Accordingly, the reward function in this case is modelled for each step as follows:

- A reward (positive) in case of reaching the target through distance threshold < 0.5 m (radius to the target).
- A reward (negative) in case of a collision with another object through a distance threshold.
- A reward (negative) if the vehicle gets close to an obstacle.

- A reward (positive or negative) as the difference in distance from the target at the current time step compared with the last time step. Which results in:

$$R(s_t) = \begin{cases} 100 & \text{if reached the target} \\ -100 & \text{if collided} \\ -0.5 * (1 - \min(lr_t)) & \text{if } \min(lr_t) < 1 \\ 2 * (d_{t-1} - d_t) & \end{cases}$$

Where lr_t denotes the laser scan data at time step t as an array and d_t denotes the Euclidian distance to the target at time step t .

3.5.6 Policy- and value network

The PPO-Agent uses both a policy network for computing an action given the current state and a value network, which calculates how "valuable" the current state was.

The policy network learns to find a mapping for every given state s to the most appropriate action a . From this, one can make state-action pairs $SA = \{(s_0, a_0), (s_1, a_1), \dots, (s_n, a_n)\}$, representing which actions lead to which states.

In contrast, the value network assigns values to the state of the task by calculating an expected cumulative score for the current state S . Here, the key objective is to maximize the reward, which means that actions that result in a desirable state receive more reward than others do.

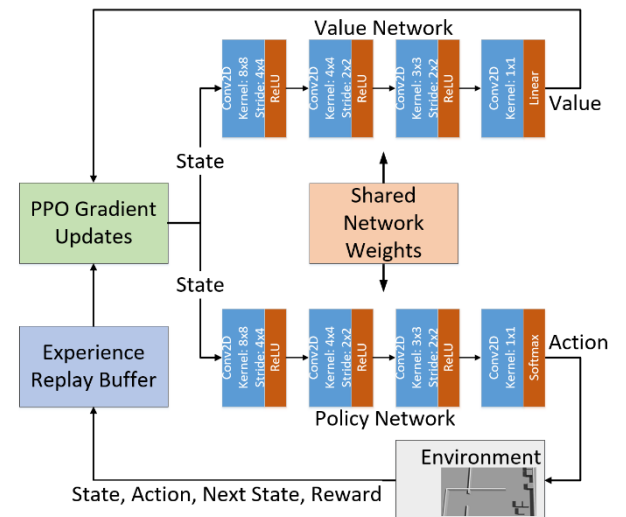


Fig. 13: Scheme of the training workflow

In order to reach the target a sequence of states/ actions is required. The state-action pair that achieves the most reward is considered as the optimal policy.

In the case of this work, both policy and value network share the same layers and have the same weights. E.g., the state contains information about surrounding obstacles and the target location. This information is relevant for both the actor and value network. Eventually both networks have to compress the state somehow and the shared layers allow more efficiency for training.

As the states are encoded as images, it makes sense to extract the relevant features with convolutional layers using a convolutional neural network (CNN).

CNN's have proven to be state of the art when it comes to the task of image analysis (Krizhevsky et al., 2017). They convolve two-dimensional input images with kernel matrices, which are optimized during the training process. This allows the network to learn local patterns, which start mostly with low-level features like edges and end up in this case with the detection of important locations in the image, such as walls and the target location.

With this, CNN's can very efficiently learn complex and abstract visual concepts and therefore find patterns where conventional methods often fail.

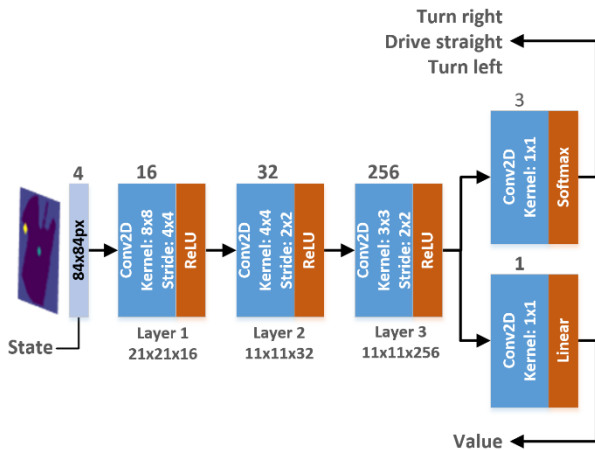


Fig. 14: Network structure for both policy and value network

In this case, the network consists of only convolutional layers. The input is the state with a resolution of 84x84px and four channels, which corresponds to the current and the last three time steps. This resolution is then broken down step by step with a 4x4 followed by a 2x2-stride operation. It has been shown that pooling layers can be successfully replaced by striding operations (Springenberg et al., 2015). On Layer 1, an 8x8-kernel matrix followed by a 4x4-kernel in Layer 2 and an 11x11-kernel in Layer 3 to flatten the output into a vector with length 256 is applied. For the policy network, a head with three outputs and a softmax activation corresponding to the possible actions is applied. For the value network, a head with one linear output is applied computing the value of the given state. Such architectures have been proven to be state of the art in several popular networks (He et al., 2015; Simonyan & Zisserman, 2015).

3.5.7 Optimization

As mentioned in (Schulman et al., 2017) vanilla policy gradients methods compute an estimate \hat{E}_t of the policy gradient π_θ to run stochastic gradient ascent:

$$L^{PG}(\theta) = \hat{E}_t[\log(\pi_\theta(a_t|s_t))\hat{A}_t]$$

In the loss formula above, π_θ is a stochastic policy and \hat{A}_t is an estimator of the advantage function at time step t .

\hat{A}_t computes the difference between the sum of all actual rewards which were received in the current episode at time step t and an estimate of what normally would happen computed by the value network. This results in a positive advantage function output if the actions taken by the Agent received more rewards than expected by the value function.

When performing multiple steps on this loss function, it can lead to large policy updates which are destructive to the network parameters.

To overcome this problem, in PPO, the logarithmic output of $\pi_\theta(a_t|s_t)$ is replaced by a ratio $r_t(\theta)$ of the current policy divided by the old policy which is clipped between $1 - \epsilon$ and $1 + \epsilon$, where ϵ is a hyperparameter and usually chosen as 0.2. $r_t(\theta)$ computes the ratio between the policy outputs of the current step and the last update step.

$$L^{CLIP}(\theta) = \hat{E}_t[\min(r_t(\theta) * \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) * \hat{A}_t)]$$

This will flatten the output of L^{CLIP} if the ratio is bigger than $1 + \epsilon$ or if it is smaller than $1 - \epsilon$. This prevents the value from being too high and overshooting above the optimum. On the other hand, it allows the last update to be reversed if the selected action was bad.

3.5.8 Game creation

In most RL applications, it is best practice to design the task like a game one can play. In terms of this work, the following game scheme is proposed:

The goal of the game is to move the yellow diamond as close to the cyan circle without moving the cyan circle into any blue areas but staying on violet areas. In order to reach this goal, one of the following actions can be taken at a time:

- Action 0: Drive to the left e.g. rotates the yellow diamond clockwise and the blue areas in direction of the black arrow (see Fig. 12) are getting closer.
- Action 1: Drive straight, e.g., the blue areas in direction of the black arrow are getting closer.
- Action 2: Drive to the right e.g. rotates the yellow diamond counterclockwise and the blue areas in direction of the black arrow are getting closer.

If the cyan circle touches any blue area, the game fails and has to be restarted.

If the cyan circle overlaps with the yellow diamond, the game is successfully finished.

Before training, a good measure to test if the task is actually solvable, is trying to play the game on its own based on the given states. If this is the case, then there is also the chance that the Agent is able to learn the task.

3.5.9 Training in simulation

The Agent has been trained in both training environments described in 3.6.1 simultaneously, which means:

- At every episode, the environment was chosen randomly by a uniform distribution.
- At every episode, the x- and y-coordinate values for the starting and target point were chosen randomly in a uniform distribution in the described area.

Furthermore, the steering angle was randomly initialized between 0.35 .. 1.0 radians before each episode. This forces the Agent to react with different kinematic settings, so that differences to the real vehicle can be tolerated better.

During training time, several parameters such as the mean reward per episode and the total loss were observed to whether the process is converging or not. Training has been manually ended after 5007 episodes, which corresponds to 656'000 steps played (after approx. 24 hours).

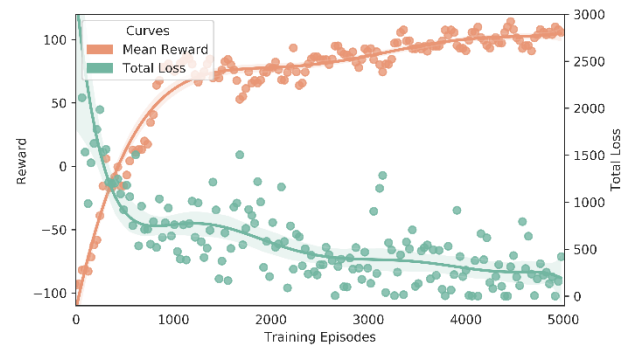


Fig. 15: Reward and loss value statistics during training of the RL-Agent

Fig. 15 shows the reward and the loss value per episode. It can be seen that the reward increases

constantly and the loss value decreases. This indicates good learning behavior.

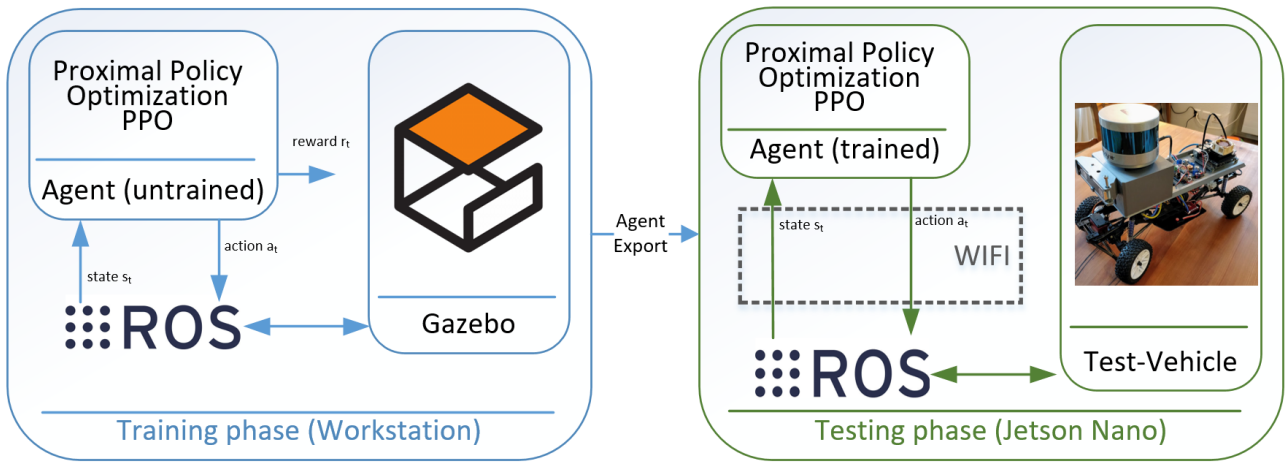


Fig. 16: Training and testing scheme. Training took place on a workstation using a gazebo simulation where testing took place on the real vehicle with the trained agent

3.5.10 What the Agent learned

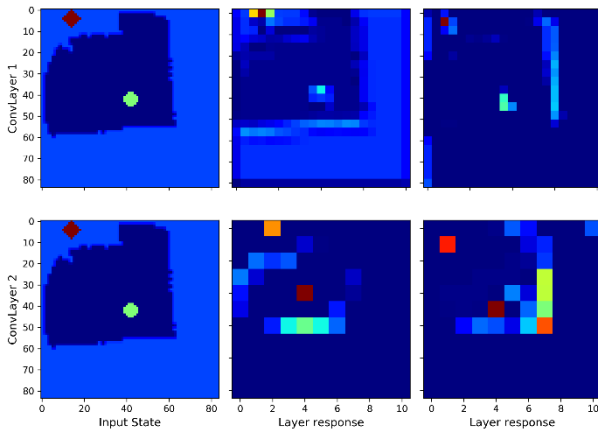


Fig. 17: Layer responses (second and third rows) of the two intermediate convolution layers of the policy network given an input state (first row) from simulation

As described in Chapter 3.5.6 the layers which are used for the value network only consist of convolutional operations. To get a better understanding of what the network is learning, individual feature maps can be made visible. One way to achieve this is the visualization of intermediate layers. In order to make the response of certain layers to an input image visible, the network can be "cut apart" at the desired layer and a




new "sub-model" can be created and initialized with the learned weights. Now the response of the layer can be made visible with a forward pass through the network of this "sub-model". In Fig. 17 the two columns correspond the two intermediate layers of the network with an output size of 21x21px for "ConvLayer1" and 11x11px for "ConvLayer2". The first row shows the first channel of a state, which is fed into the two "sub-models". The second and third row show the response by a selected kernel.

On the first intermediate layer (ConvLayer1), responses to the target location and vehicle and to the wall can be noticed. On the second intermediate layer (ConvLayer2), these reactions are even more visible. Here, only walls and/or the target position are filtered out.

3.5.11 Deployment to test-vehicle

During testing on the real vehicle, the necessary actions were computed on a remote notebook. The states are sent through a wireless network to the notebook, then the notebook on which the trained Agent is running computes the action and sends it back over the same network to the vehicle. This scheme allows to easier test and deploy changes.

3.6 Environments

-  **Start and target area**
-  **Forward direction**
-  **Backward direction**

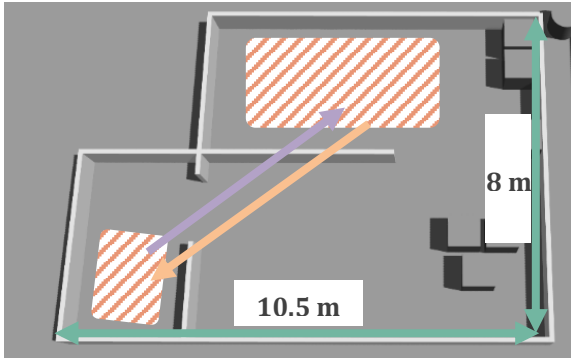


Fig. 18: TrainEnv-1, Training environment 1

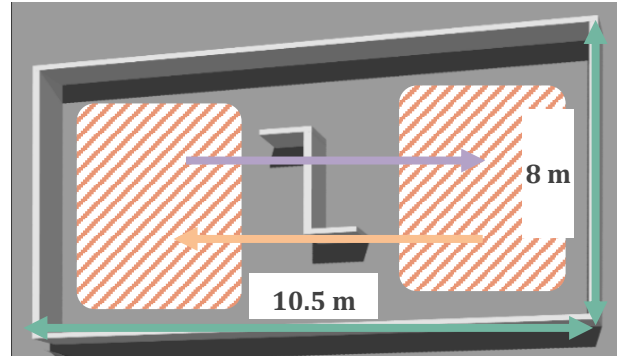


Fig. 19: TrainEnv-2, Training environment 2

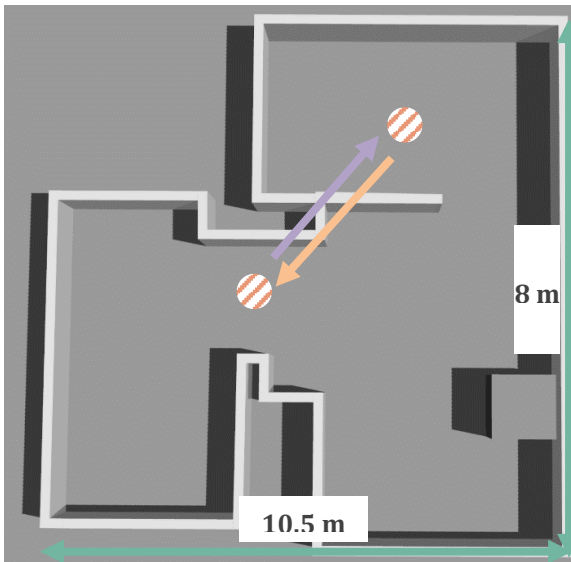


Fig. 20: ValEnv-1, Validation-environment 1

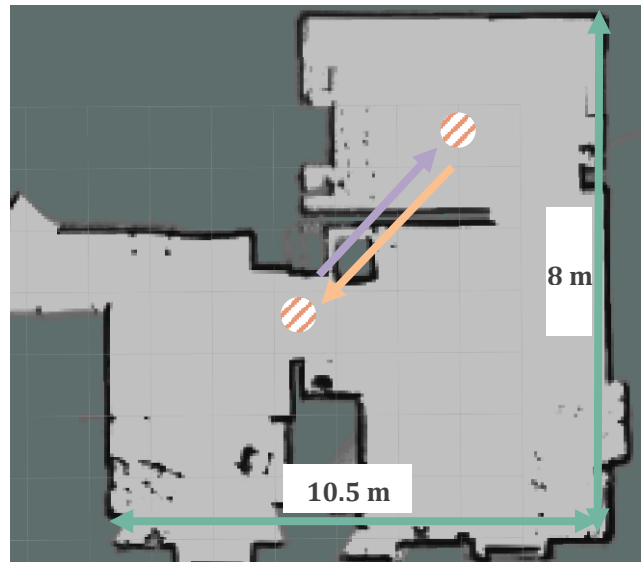


Fig. 21: TestEnv-1, Testing-environment 1

3.6.1 Simulation

A simulation was created using ROS and Gazebo for training and evaluating the Agent. The used simulated vehicle is mainly based on the MIT Racecar project (*Mit-Racecar*, n.d.). Adjustments were made so that the dimensions and sensor position correspond to the test-vehicle.

3.6.1.1 Training-environments

For training, two environments were constructed (Fig. 18, Fig. 19) to cover different situations. TrainEnv-1 (Fig. 18) is a rather simple environment with only one obstacle in the middle, which allows the car to move always towards the target. TrainEnv-2 (Fig. 19) in comparison is more complicated: To reach the target it is required to not permanently get closer to the target but to first drive around two corners.

During the training process, on every episode:

- A random environment is chosen (Env-1/ Env-2)
- The starting and target position is initialized randomly within a certain range; see violet coloured areas in figures.

3.6.1.2 Validation-environment

The validation-environment (ValEnv-1, Fig. 20) is intended to have similar (but not exact) shape and dimensions as the test-environment. This allows an interpretation of how well the Agent adapts to the real-world vehicle and conditions. Also the starting and target positions were chosen similarly as they are in the test-environment. Here the Agent has to drive around one corner in order to reach the target and therefore has no direct view towards the target.

3.6.2 Real world

For the real world tests, an indoor environment was chosen to test the capabilities of the Agent on the test vehicle.

To compare both the TEB-local planner and the reinforcement learning approach, a test setup has been created in an indoor environment.



Fig. 22: Visual representation of the testing indoor environment

3.6.2.1 Testing-environment

The real world test-environment (TestEnv-1, Fig. 21) is similar in shape and size as the validation environment (ValEnv-1, Fig. 20). A map of the environment has been created with the SLAM method mentioned in 3.3.

3.7 Software specifications

On the Jetson Nano mounted on the test-vehicle the following software is used:

- Ubuntu 18.04, Bionic Beaver
- ROS: Noetic running in a docker container
- Python 3.7

For the simulation part, a workstation operating with an 8 core CPU and an Nvidia GTX 1070 GPU, the following software is used:

- Ubuntu 20.04, Focal Fossa
- ROS: Noetic
- Gazebo: 11.1.0
- Python 3.8

4 EXPERIMENTS

In order to investigate the capabilities of the created RL navigation approach, various experiments were conducted. On one hand, the behaviour of the Agent during training and in unseen environments was investigated. On the other hand, it was also tested how well the trained Agent in the simulation performs navigation tasks on the test vehicle in real conditions.

In addition, the SLAM process and the navigation using the TEB algorithm over longer distances with dynamically moving obstacles were tested. However, these two tests were not systematically compared with other approaches. In the table below, all experiments are listed with a corresponding video-link.

Experiment	Description	Video Link
Experiment 1, RL Training Process	This experiment visually shows the trajectories chosen by the RL-Agent at different training steps regarding a given starting and target position.	TrainEnv-1: https://youtu.be/Hooh3NiFTTs
Experiment 2, 4x15 RL navigation runs in simulation	This experiment is used to determine the reliability of arriving at the target and the distribution of the distance to the target over the different runs in simulation regarding the RL-Agent. In TrainEnv-1 and ValEnv-1, 15 runs are made in the forward direction and 15 runs in backward direction.	TrainEnv-1: https://youtu.be/14DIF1tF5_M ValEnv-1: https://youtu.be/-OOgzT3pEvk
Experiment 3, 2x15 RL navigation runs in real-world	This experiment is used to determine the reliability of arriving at the target and the distribution of the distance to the target over the different runs in real world regarding the RL-Agent. In each case, 15 runs are made in the forward direction and 15 runs in backward direction.	TestEnv-1: https://youtu.be/mdwHY7vHC_I
Experiment 4, 2x15 TEB navigation runs in real-world	This experiment serves as a baseline to the reliability of arriving at the target and the distribution of the distance to the target. The path is computed by the TEB algorithm. In each case, 15 runs are made in the forward direction and 15 runs in backward direction.	TestEnv-1: https://youtu.be/u6-KPhmEHtY
Test 1, Mapping of the Testing Environment	In this video, the mapping process of the testing environment is shown using the Google Cartographer tool.	https://youtu.be/yRwpUBROW-k
Test 2, TEB Dynamic Obstacles	This experiment is meant to show the capabilities of TEB Planner in path planning over longer distances and dynamic obstacle avoidance, two sample runs were recorded.	https://youtu.be/GOrs-yyzaxg

Fig. 23: Conducted experiments

4.1 Localization in the experiments

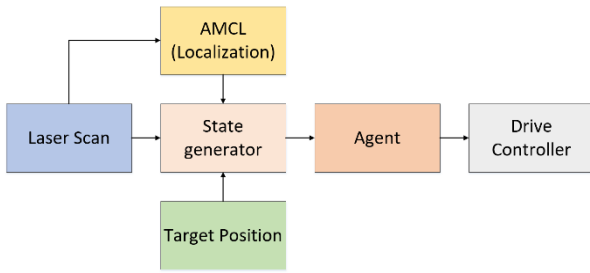


Fig. 24: Block diagram of the RL-based navigation during real world testing

As mentioned, a map of the test-environment was previously created. To localize the vehicle based on this map, the ROS AMCL package (*Amcl - ROS Wiki*, n.d.) is used. The package uses a particle filter to track the pose of the vehicle against a known map with the laserscan data. Then this pose was recorded in rosbags during the experiments. Those rosbags were then later replayed and the trajectories of the base_link frame regarding the map frame were extracted.

In simulation, the current position of the vehicle was read directly from the gazebo environment.

4.2 Experiment 1

Experiment 1 intends to show the training behaviour of the Agent at different training steps in a qualitative manner. As mentioned in 3.5.7, the start and target positions have been randomly initialized in a certain area during the training process. However, in order to better demonstrate the capabilities of the Agent at certain training steps, a fixed start and target position is chosen here. Also the steering angle of the

simulated vehicle is fixed. The Agent is initialized with the network parameters of different training levels and tested for 15 runs each in Env-1 (Training) and Env-3 (Validation), see Fig. 25 ... Fig. 28.

4.2.1 Interpretation

Looking at Fig. 25 and Fig. 26 (TrainEnv-1), a positive learning behavior is observed. This finding also correlates with the learning curve of the mean reward in Fig. 15. In the beginning, many collisions are recorded right after the start. With an increasing number of episodes, the trajectory forms a uniform line towards the target.

However, when looking at the trajectories of the validation environment at Fig. 27 and Fig. 28, the behavior of the Agent is worse than in TrainEnv-1. Which means, that fewer runs are reaching the target position over all episodes and the spread of the trajectories is larger compared to TrainEnv-1.

4.3 Experiments 2 – 4

Experiments 2-4 are designed to determine the reliability of the RL method in reaching the given target position. For this purpose, tests are carried out with the trained RL agent in the training, validation and testing environment. In all environments, navigation is carried out 15 times with a slightly randomised start position and a fixed target position in both a forward and a backward direction. During each test, the corresponding sensor and position data were recorded.

4.4 Trajectories of Experiment 1

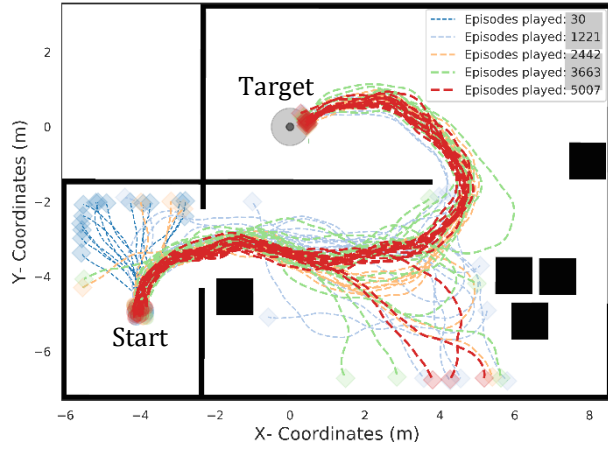


Fig. 25: Trajectories of the trained RL Agent replayed at different training stages in TrainEnv-1 in forward direction

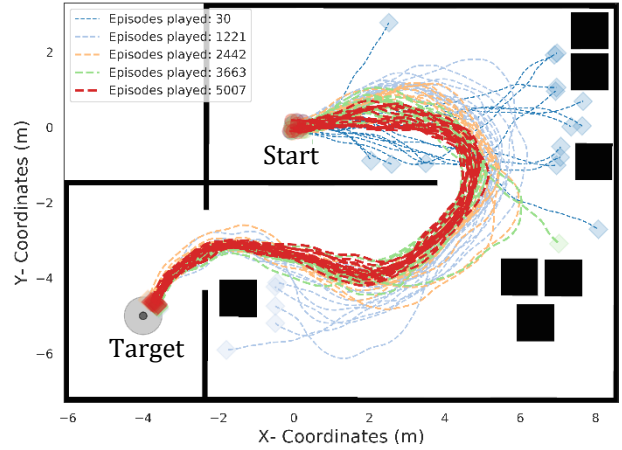


Fig. 26: Trajectories of the trained RL Agent replayed at different training stages in TrainEnv-1 in backward direction

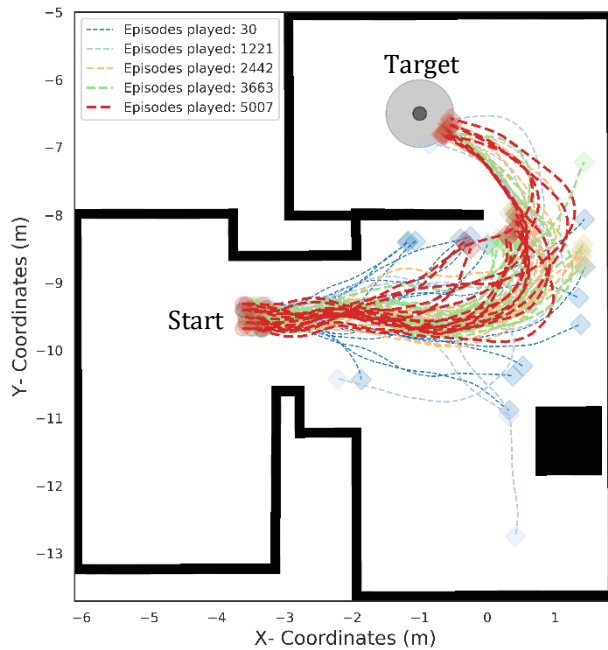


Fig. 27: Trajectories of the trained RL Agent replayed at different training stages in ValEnv-1 in forward direction

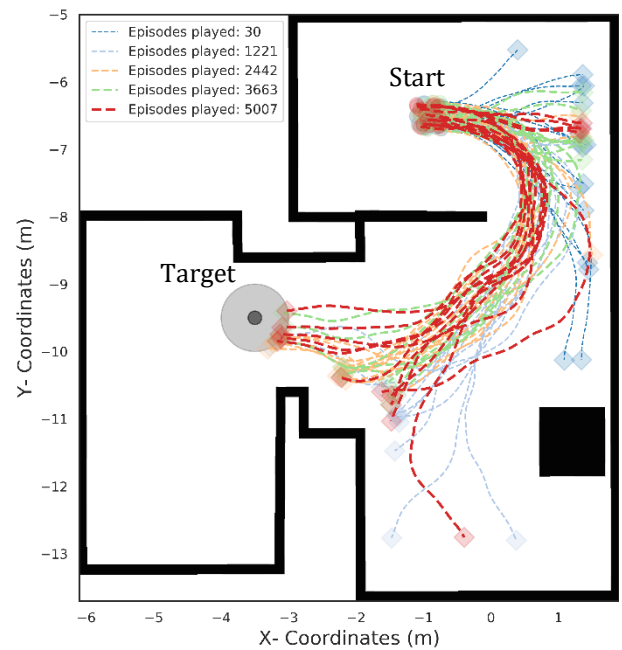


Fig. 28: Trajectories of the trained RL Agent replayed at different training stages in ValEnv-1 in backward direction

4.5 Trajectories of Experiment 2

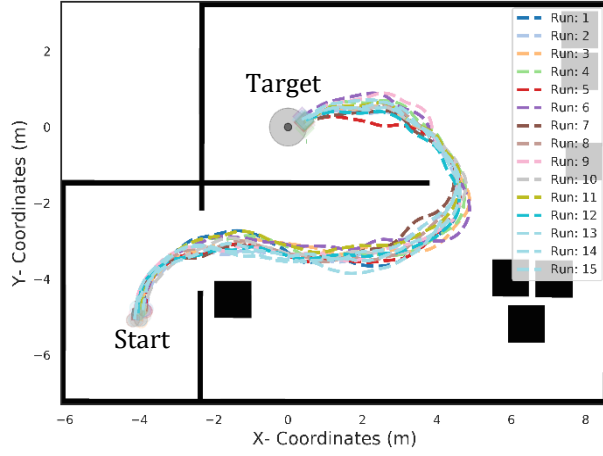


Fig. 29: Trajectories of the trained Agent tested in TrainEnv-1 in forward direction

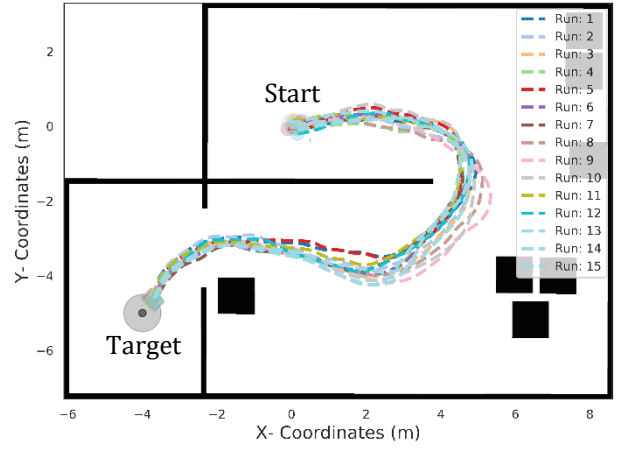


Fig. 30: Trajectories of the trained Agent tested in TrainEnv-1 in backward direction

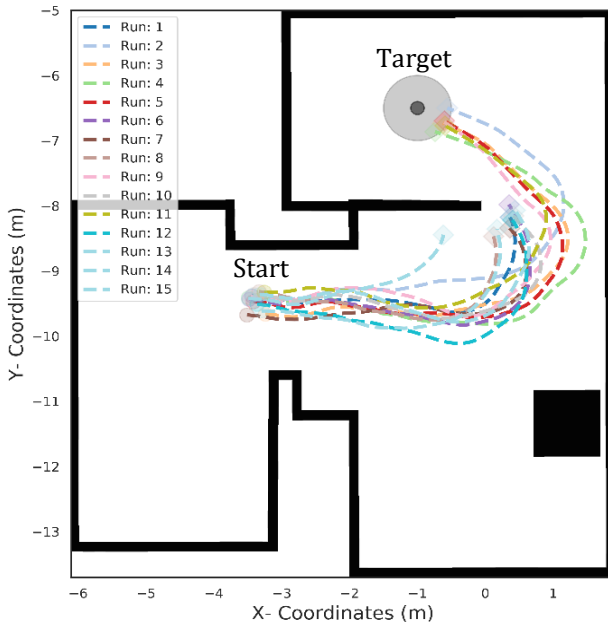


Fig. 31: Trajectories of the trained Agent tested in ValEnv-1 in forward direction

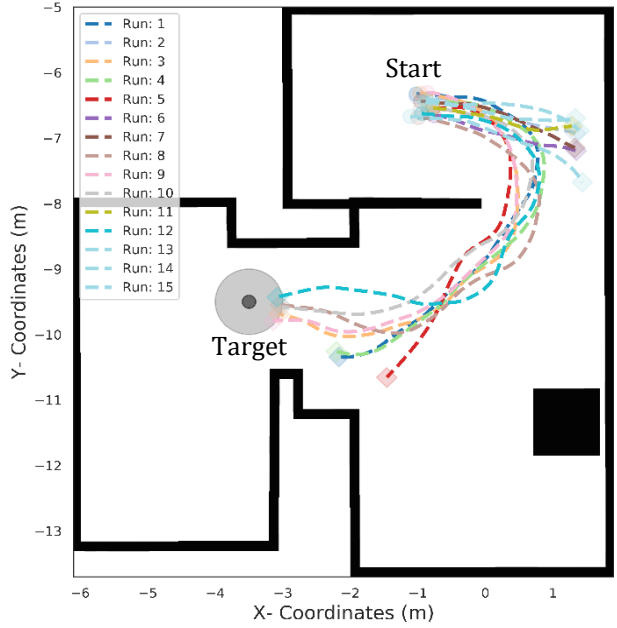


Fig. 32: Trajectories of the trained Agent tested in ValEnv-1 in backward direction

Algorithm	Environment	Direction	Target reached	Success Rate	Mean Dist. to target
RL	TrainEnv-1	Forward	15/15	100 %	0.46 m
RL	TrainEnv-1	Backward	15/15	100 %	0.46 m
RL	ValEnv-1	Forward	6/15	40 %	1.47 m
RL	ValEnv-1	Backward	5/15	33 %	3.05 m

Fig. 33: Numerical results of experiment 2

4.6 Trajectories of Experiment 3 and 4

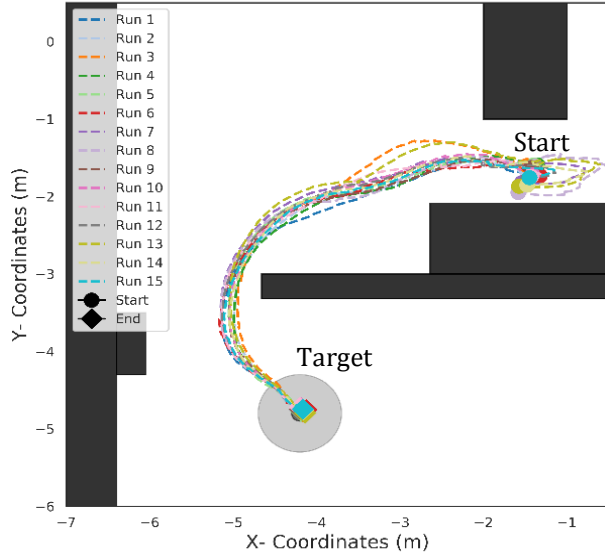


Fig. 34: Trajectories of the vehicle in TestEnv-1 in forward direction using the TEB local planner given the full map

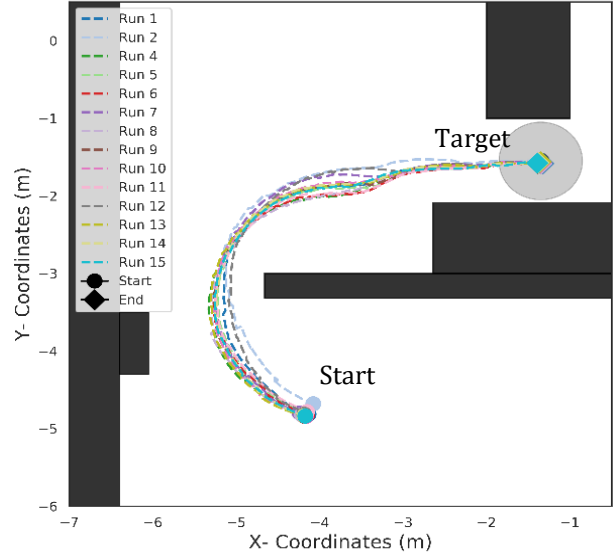


Fig. 35: Trajectories of the vehicle in TestEnv-1 in backward direction using the TEB local planner given the full map (Run 3 is invalid due to an error in the rosbag file and is not counted in the statistics).

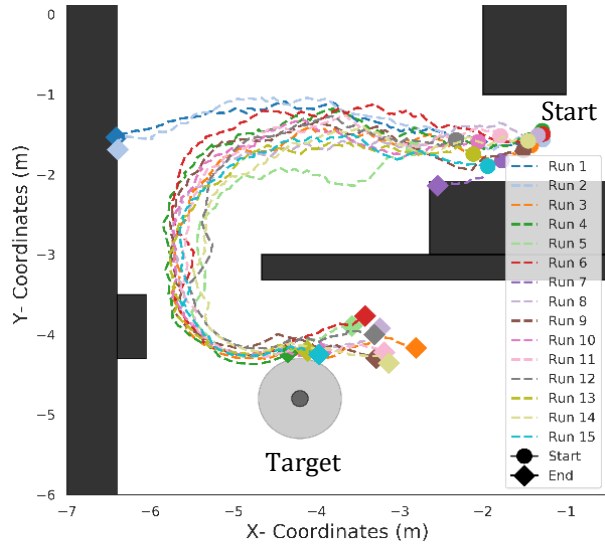


Fig. 36: Trajectories of the vehicle in TestEnv-1 in forward direction using the reinforcement learning approach given only local map information

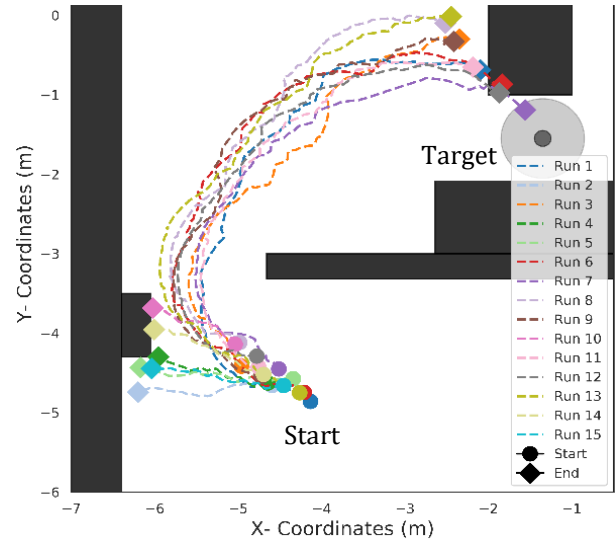


Fig. 37: Trajectories of the vehicle in TestEnv-1 in backward direction using the reinforcement learning approach given only local map information

Algorithm	Environment	Direction	Target reached	Success Rate	Mean Dist. to target
TEB	TestEnv-1	Forward	15/15	100 %	0.06 m
TEB	TestEnv-1	Backward	14/14	100 %	0.03 m
RL	TestEnv-1	Forward	3/15	20 %	1.54 m
RL	TestEnv-1	Backward	1/15	7 %	2.94 m

Fig. 38: Numerical results of experiment 3 and 4

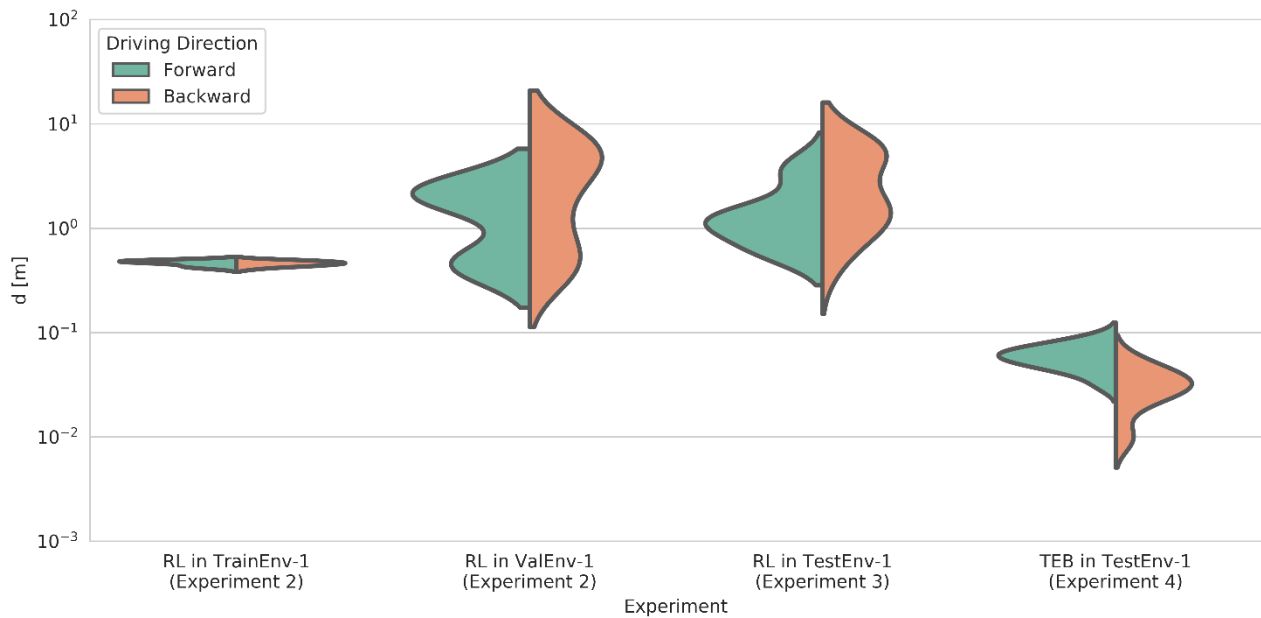


Fig. 39: Distribution of the distance to target at the last position of each run (m) of the experiments. Note that the y-axis is in logarithmic scale

4.6.1 Interpretation

4.6.1.1 Experiment 2

The runs in TrainEnv-1 each achieve a 100% success rate. This is to be expected when comparing with the training statistics. However, the success rate in ValEnv-1 decreases rapidly. The agent usually drives in the right "direction" but sometimes fails due to collisions with obstacles. This could also be due to the fact that the agent cannot generalize enough, i.e. it has partly learned the training environment "by heart" and can therefore adapt rather poorly to new environments. To counteract the lack of generalization mentioned above, more variation could be built into the training environment. E.g. with more different rooms. However, this would increase the training complexity and duration.

4.6.1.2 Experiments 3 and 4

When looking at the two methods, it can be seen that the TEB-variant reaches the target in 100 % of the cases with the defined tolerance and has only a minimal deviation from the ideal target point. In

contrast, the RL method has a poor success rate of 0 % for direction 1 and 7 % for direction 2.

However, the RL agent manages to get at least close to the target in the majority of cases. In both the forward and reverse directions, an error regarding the chosen orientation in the last few metres before the target can be observed. This can be caused by the fact that the target on the state is very close to the vehicle point. This means that the agent can only detect poorly an angular deviation in the last few metres from the target.

Comparing Experiments 3 and 4 with Experiment 2, it can be noticed that the behaviour of the Agent in ValEnv-1 and TestEnv-1 is very similar to each other. This at least indicates that the Agent on the real vehicle shows similar behaviour as in the simulation and thus copes well with slightly varying kinematics and settings.

4.7 Experiment 5

To demonstrate the capabilities of the TEB algorithm with a known map, three runs with dynamic obstacles were recorded (see Video). The TEB

planner was able to successfully navigate towards longer-range targets and avoid dynamically moving obstacles. However, this experiment was only intended to show the capabilities of the TEB planner and is not compared to another experiment.

5 DISCUSSION

The aim of this work was to investigate the problem of mapless navigation using a reinforcement learning approach for indoor environments. A simulated environment was built using ROS and Gazebo as a simulation engine in which it is possible to control a vehicle based on Ackermann steering. An RL approach based on Policy Proximity Optimization has then been used to train an Agent in the simulated environment given the target position relative to its current position and laser range data to detect obstacles. This means that the Agent was not aware of the full map of its environment but only of the current surroundings in 5 m radius.

The trained Agent has then been validated both in a simulated validation environment and in a real-world scenario. For this, a test vehicle has been constructed in advance based on RC-car components, the required sensors and a Nvidia Jetson board to carry out the control and computing tasks. Furthermore, the proposed approach has been compared to a state of the art TEB navigation method with the full map known.

In the experiments of the RL based approach in both simulated validation environment and real world environment, it could be shown that the vehicle misses the target most times, but only by little distance. Since the Agent was only trained in two different training environments, just a small variance of possible situations is covered. This makes it difficult for the Agent to generalize and there is also the threat that parts of the environments will be learned "by heart". This lack of generalization could be addressed by dynamically changeable environments or a much larger number of different situations.

Based on the recorded trajectories in most fail-cases, the direction of travel of the vehicle in the last area of the trajectory shortly before the destination is not

directed towards the target, meaning the vehicle drives past the target. One reason for this misbehaviour could be that the target position in the state available to the RL-Agent is very close to the vehicle in this situation and thus an angle deviation is difficult to detect on the state observation. A possible solution to this problem could be to adjust the state dynamically; e.g. reduce the horizon of the observation when approaching the target and thus increase the resolution of the observation. Another possibility would be to limit the horizon of the observation in general and to divide the final goal into several sub-goals, e.g. to search for the best possible goal in the current state regarding the final goal.

In general, the resolution in the state is rather coarse, which only allows a rough approximation of the actual environment. In principle, this resolution could be chosen finer, but this would increase the feature space in the state and thus mean more trainable parameters which would have impact on training time and complexity. Here, an optimal trade-off between accuracy and training time probably has to be found.

As stated in various resources, in modern RL, the central challenge remains on how to reuse a learned expertise to solve a problem in one environment and apply it to another problem in another environment. The comparison of the trajectories in ValEnv-1 of experiment 2 and those in TestEnv-1 of experiment 3 demonstrate very similar behaviour and therefore show that the Agent copes well with the test conditions even though they differ from the simulated conditions. From this, it can be concluded that in principle a deployment from simulation to real world is possible as long as the varying parameters in the simulation are taken into account. This was solved here by changing the steering angles and the start and target positions dynamically in each training episode, which has forced the Agent to

become more "tolerant" towards changing parameters.

6 CONCLUSION

In this thesis, an existing motion planner algorithm was implemented on a self-made Ackermann steering based vehicle. In addition, a discrete control RL approach was trained end-to-end in simulation and then compared to the "traditional" motion planner in a test environment on the test vehicle.

By encoding the laser scan findings and the target position relative to the vehicle coordinate frame as an image, the RL approach can be applied directly to environments without knowing the full map. It could be successfully shown that it is possible to train an Agent in simulation and to deploy it to a real-world

and low cost test vehicle without retraining or adapting any parameters.

However, the success rate of actually reaching the target with the defined tolerance (0.5 m radius) is low (20% in forward resp. 7% success rate in backward direction) compared to the approach with the state of the art TEB local planner (100% success rate in both directions).

The proposed method has potential in the sense that a complete map of the environment is no longer necessary, and sensors can be replaced by less expensive variants. Given the fact that the deployment of trained reinforcement learning based application in simulation to real world applications is still sparsely studied, this work can serve as a basis for future projects in this direction.

7 APPENDIX

The software code is hosted on GitHub at: <https://github.com/dschori/Ackerbot>

8 REFERENCES

- amcl*—ROS Wiki. (n.d.). Retrieved 6 March 2021, from <http://wiki.ros.org/amcl>
- Capasso, A. P., Bacchiani, G., & Broggi, A. (2020). From Simulation to Real World Maneuver Execution using Deep Reinforcement Learning. *ArXiv:2005.07023 [Cs]*. <https://doi.org/10.1109/IV47402.2020.9304593>
- Christiano, P., Leike, J., Brown, T. B., Martic, M., Legg, S., & Amodei, D. (2017). Deep reinforcement learning from human preferences. *ArXiv:1706.03741 [Cs, Stat]*. <http://arxiv.org/abs/1706.03741>
- Durrant-Whyte, H., & Bailey, T. (2006). Simultaneous localization and mapping: Part I. *IEEE Robotics Automation Magazine*, 13(2), 99–110. <https://doi.org/10.1109/MRA.2006.1638022>
- Giovannangeli, C., Gaussier, P., & Desilles, G. (2006). Robust Mapless Outdoor Vision-Based Navigation. 2006 *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 3293–3300. <https://doi.org/10.1109/IROS.2006.282501>
- He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. *ArXiv:1512.03385 [Cs]*. <http://arxiv.org/abs/1512.03385>
- Hess, W., Kohler, D., Rapp, H., & Andor, D. (2016). Real-Time Loop Closure in 2D LIDAR SLAM. 2016 *IEEE International Conference on Robotics and Automation (ICRA)*, 1271–1278.
- Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., & Silver, D. (2017). Rainbow: Combining Improvements in Deep Reinforcement Learning. *ArXiv:1710.02298 [Cs]*. <http://arxiv.org/abs/1710.02298>
- Jin, J., Nguyen, N. M., Sakib, N., Graves, D., Yao, H., & Jagersand, M. (2020). Mapless Navigation among Dynamics with Social-safety-awareness: A reinforcement learning approach from 2D laser scans. 2020 *IEEE International Conference on Robotics and Automation (ICRA)*, 6979–6985. <https://doi.org/10.1109/ICRA40945.2020.9197148>
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6), 84–90. <https://doi.org/10.1145/3065386>
- Liang, E., Liaw, R., Moritz, P., Nishihara, R., Fox, R., Goldberg, K., Gonzalez, J. E., Jordan, M. I., & Stoica, I. (2018). RLlib: Abstractions for Distributed Reinforcement Learning. *ArXiv:1712.09381 [Cs]*. <http://arxiv.org/abs/1712.09381>

- Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2019). Continuous control with deep reinforcement learning. *ArXiv:1509.02971 [Cs, Stat]*. <http://arxiv.org/abs/1509.02971>
- Mit-racecar*. (n.d.). GitHub. Retrieved 13 February 2021, from <https://github.com/mit-racecar>
- Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (n.d.). *Playing Atari with Deep Reinforcement Learning*. 9.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529–533. <https://doi.org/10.1038/nature14236>
- Osiński, B., Jakubowski, A., Miłoś, P., Zięcina, P., Galias, C., Homoceanu, S., & Michalewski, H. (2020). Simulation-based reinforcement learning for real-world autonomous driving. *ArXiv:1911.12905 [Cs]*. <http://arxiv.org/abs/1911.12905>
- Roesmann, C., Feiten, W., Woesch, T., Hoffmann, F., & Bertram, T. (2012). Trajectory modification considering dynamic constraints of autonomous robots. *ROBOTIK 2012; 7th German Conference on Robotics*, 1–6.
- Rösmann, C., Feiten, W., Wösch, T., Hoffmann, F., & Bertram, T. (2013). Efficient trajectory optimization using a sparse model. *2013 European Conference on Mobile Robots*, 138–143. <https://doi.org/10.1109/ECMR.2013.6698833>
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. *ArXiv:1707.06347 [Cs]*. <http://arxiv.org/abs/1707.06347>
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587), 484–489. <https://doi.org/10.1038/nature16961>
- Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv:1409.1556 [Cs]*. <http://arxiv.org/abs/1409.1556>
- Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2015). Striving for Simplicity: The All Convolutional Net. *ArXiv:1412.6806 [Cs]*. <http://arxiv.org/abs/1412.6806>
- Tai, L., Paolo, G., & Liu, M. (2017). Virtual-to-real Deep Reinforcement Learning: Continuous Control of Mobile Robots for Mapless Navigation. *ArXiv:1703.00420 [Cs]*. <http://arxiv.org/abs/1703.00420>
- teb_local_planner—ROS Wiki*. (n.d.). Retrieved 20 February 2021, from http://wiki.ros.org/teb_local_planner

- Zhang, J., Springenberg, J. T., Boedecker, J., & Burgard, W. (2017). Deep Reinforcement Learning with Successor Features for Navigation across Similar Environments. *ArXiv:1612.05533 [Cs]*. <http://arxiv.org/abs/1612.05533>
- Zhelo, O., Zhang, J., Tai, L., Liu, M., & Burgard, W. (2018). Curiosity-driven Exploration for Mapless Navigation with Deep Reinforcement Learning. *ArXiv:1804.00456 [Cs]*. <http://arxiv.org/abs/1804.00456>

9 DECLARATION OF AUTHORSHIP

I hereby declare that I made the available work independently and without use of others than the indicated aids. The out strange sources directly or indirectly taken over thoughts are marked as such. The work was not submitted or published so far in same or similar form.

Rapperswil, 17. March 2021

A handwritten signature in black ink, consisting of a series of loops and a long horizontal stroke extending to the right.

Damian Schori