Local Planning for Unmanned Ground Vehicles using Imitation Learning

Johan Henningsson

Department of Automatic Control

MSc Thesis
TFRT-6210
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Abstract

Mobile robotics is an expanding field worldwide leading to the need for advanced path-planning algorithms that can traverse various environments. Current state-of-the-art path-planning algorithms used at the Swedish Defence Research Agency, FOI, tend to be inflexible and parameter dependent. The parameters might need to be tuned for each new environment, which is a very labor-intensive process.

This thesis investigates the possibility to replace computationally heavy path-planning algorithms with neural networks using Imitation Learning. Neural networks with and without Long Short-Term Memory (LSTM) layers were trained and evaluated. The network without LSTM failed to capture the temporal dependency of the input data, which lead to poor performance. Using LSTM layers performed close to the imitated algorithm in the training environment and in certain situations, the trained neural network outperformed the algorithm by a big margin. In conclusion, neural networks are, after training, able to replace path-planning algorithms and in certain scenarios, the network outperforms the algorithm. Further work is needed to get a robust local planner with a neural network as a base.

Acknowledgements

Firstly, I would like to thank my supervisors Fredrik Bissmarck and Jonas Nordlöf at the Swedish Defence Research Agency, FOI, and Anders Rantzer from the Department of Automatic Control at Lund University. Fredrik and Jonas have provided great discussions and ideas during our meetings and I always had somewhere to turn if I had any questions. Anders gave great feedback on my writing, elevating my work to the next level.

Secondly, a big thank you to my examiner Bo Bernhardsson from the Department of Automatic Control at Lund University for always answering all my questions and guiding me in the right direction.

Lastly, I would like to mention two fellow Master Thesis students who have been great support: Nicke Löfwenberg and Maja Boström. Thank you for being a sounding board during this whole project. You provided good insights and feedback that helped me improve my work as well as keep me going when I got stuck.

Contents

1 Introduction

The use of mobile robotics is booming worldwide, with applications in many different fields such as logistics, health care, agriculture, and military. This development gives rise to the need for navigational systems that can traverse various environments with potentially unknown and dynamic obstacles [Fragapane et al., 2020]. The current state-of-the-art solutions for the navigation task are generally split into two parts, a global and a local planner. The global planner constructs the long-range plan from the starting point to the goal point and it assumes a static environment. The global plan is passed to the local planner and the local planner will perceive the environment through different sensors, e.g. LiDAR sensors and cameras, and based on this local perception it will make a short-term plan that optimally should follow the global plan as closely as possible at the same time as avoiding obstacles. The current approaches that employ this hierarchical structure run into problems when handling fast-moving obstacles and the optimization algorithms provide a large computational burden. Often the algorithms need to be hand-engineered for the specific environment to meet safety and robustness constraints, which makes the implementation a very involved process [Alatise and Hancke, 2020].

Development in recent years has started implementing machine learning components as a part of the navigation solution to mitigate the problems present in the optimization algorithms. Deep Reinforcement Learning (DRL) has provided a promising solution for handling unknown and dynamic obstacles [Everett et al., 2018] [Guldenring et al., 2020] [Chen et al., 2018]. However end-to-end DRL algorithms have been shown to have trouble learning good behavior in certain environments with sparse rewards [Faust et al., 2017]. There is also the problem that always accompanies DRL, i.e. how the network best should be trained and how to generate the needed training data. Different ways of solving these issues have been introduced in recent years with slightly different approaches. In [Kästner et al., 2021] a DRL-based control switch was proposed that depending on the local environment switches between using a traditional local planning algorithm or a DRL-based planner. A local planner purely based on Reinforcement Learning (RL) combined with a global planning algorithm was proposed in [Faust et al., 2017], whereas in [Chiang et al., 2018] an end-to-end RL planner was proposed. The use of Behaviour Cloning (BC), which is a Supervised Learning (SL) approach, in navigational tasks has been researched extensively, and using this approach has advantages and disadvantages. Supervised learning assumes that the data is i.i.d. (independent and identically distributed), which the sequential data from a navigational task is not. An advantage to using SL is that the training is quick and easy to set up. Approaches to solving or at least mitigating the problems with SL in navigational tasks have been researched, Data Aggregation (DAgger) being one of them [Ross et al., 2010].

Current path planning algorithms used at the Swedish Defence Research Agency, FOI, tend to be very parameter dependent. The algorithms used for the path planning are optimization algorithms that have a lot of options: choosing weights for how much a certain behavior should be penalized, how far in the future shall the algorithm look, what update rate should the local plan have, how close is the Unmanned Ground Vehicle (UGV) allowed to come to an obstacle, etc. It is often hard to determine which parameter affects the performance of the UGV, leading to a trial-and-error-based approach with no clear plan of action. Some settings might even lead to the optimization algorithm being slowed down leading to the local planner missing deadlines, which is detrimental to performance. What parameter settings work in one environment might struggle in another and the tuning is a time-consuming endeavor. This provides an issue if the UGV is to be introduced for military applications. The UGV needs to work in numerous environments without the demand for tuning parameters. It would be unacceptable if the UGV got stuck in a narrow corridor just because some setting was a bit off, and thereby not being able to solve its task, which in the worst case could lead to a failed mission and possibly military personnel getting injured or killed.

When the path planning algorithm is well-tuned the performance is good and the path is close to optimal, i.e. it follows the global plan. Having this in mind, a naive starting point is to investigate what happens if a neural network tries to imitate the optimization algorithm used for navigation. Could the network learn to achieve the same path following performance from only trying to imitate without the need for tuning? The idea going into this thesis is that the network could learn good path-following behavior from imitating the planning algorithm in one environment. When introduced to a new environment where the planning algorithm requires a lot of work tuning parameters, the neural network could work directly without any tuning. The neural network should only be trained in the first environment and still perform in the new environment without additional training. Using a neural network would also mitigate the risk of missing deadlines since the prediction step of the neural network would always be below a set time and consistent for all iterations.

1.1 Research questions

In the list below the research questions that will be investigated are listed in the order of priority.

1. Can a neural network replace current state-of-the-art path following algorithms and local planners while keeping the same level of performance?
2. Can a neural network generalize to new environments better than current state-of-the-art path following algorithms?
3. How can a neural network be trained to imitate current state-of-the-art path following algorithms? What machine learning approaches can be used?
4. How can the network architecture be constructed to fully utilize the temporal information in the input data?

The main goal of this thesis is to investigate the possibility of using a trained neural network as the local planner instead of a computationally heavy optimization algorithm. A number of metrics are evaluated including the success rate of reaching the goal, time to reach the goal, distance traveled, and generalization to novel settings and environments. If the neural network can achieve close to the same metrics as the optimization algorithm but it generalizes to novel environments better, it would be a promising result for further development.

1.2 Limitations

The construction and implementation of the global path planner and the handling of dynamic obstacles are out of the scope of this thesis work. While the testing of the path-planning algorithm on a physical UGV unit would potentially give more insights, this thesis exclusively deals with simulated UGV control and thus treats such tests as out of scope. Similarly, training data obtained from the physical UGV was also deemed out of scope.

2

Background

This chapter presents the path-planning problem and some common algorithms that are used to solve this problem are presented. Thereafter, a very brief review of Machine Learning is presented.

2.1 Path planning

The goal of a path-planning problem is to compute a continuous path from a starting point to a goal point while avoiding collision with static and dynamic obstacles, at the same time, respecting the kinematic and dynamic motion constraints of the mobile robot. As mentioned in the introduction, path planning is generally split up into a global planner and a local planner. For the global planner, there exist several algorithms that can find an obstacle-free trajectory between start and goal, e.g. Dijkstra's algorithm, A*, and Rapidly Exploring Random Tree (RRT), which all have their pros and cons. For some algorithms, there are optimality guarantees for the generated trajectory, where optimal can have a different meaning in different scenarios.

In Fig. 2.1 a path-planning problem is displayed, with three example trajectories. All trajectories are successful in providing a continuous path from start to goal, however, if finding the shortest path is of importance then trajectories 2 and 3 are better than 1, but if keeping a certain distance to the obstacles is of importance then trajectory 1 might be more optimal than 2 and 3. The focus of this thesis is the local planner so it is assumed that there exists a global plan connecting the start and the goal that is close to optimal. The local planner is the adaptive part of the planner, which handles static and dynamic obstacles not known beforehand when constructing the global plan. Constructing a functioning local planner has its difficulties. How should obstacles be handled? What is a safe distance from the obstacle? How far in the future should the local planner plan? How should the local planner behave if an obstacle is placed on the global plan, making it impossible to follow the global plan? These questions have led to the development of several local planners, e.g. Dynamic

Window Approach (DWA) [Fox et al., 1997], Timed Elastic Band (TEB) [Rösmann et al., 2012] and Model Predictive Control (MPC) [Rösmann et al., 2021]. The three mentioned approaches are based on optimization algorithms, which have the goal of minimizing a cost function or maximizing an objective function. There is a fine line when constructing the cost or objective function, making sure that the local planner is robust and can handle obstacles in a safe way as well as generating a trajectory that is optimal both in distance and time. Optimization algorithms tend to have issues when handling fast-moving obstacles. For this problem machine learning approaches have proven useful [Everett et al., 2018] [Guldenring et al., 2020] [Chen et al., 2018].

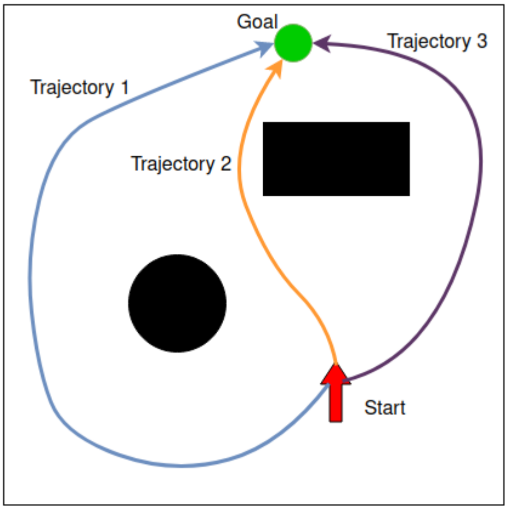


Figure 2.1 A path planning problem with three viable trajectories. Trajectory 1 is successful but it is not optimal when evaluating the length of the trajectory. However, if keeping a certain distance from the obstacles is important then it might be optimal.

2.2 Machine learning

Machine learning approaches are often split into three or four categories: supervised learning, unsupervised learning, reinforcement learning and some like to add a fourth category, semi-supervised learning. In this thesis, the machine learning approach used is SL, however, a brief overview of RL will also be presented for the comparison.

2.2.1 Supervised learning

In Supervised Learning (SL) the available data has labels, meaning that each sample has the correct or true value connected to it. In an image classification setting the data consists of the image of e.g. a dog together with the label "Dog". In that way, the learning algorithm can train on datasets where the correct class or value is known. Generating labeled data might be very expensive, time-consuming, and require an expert to manually label the data, therefore multiple labeled datasets have been generated for certain tasks. Some examples are the MNIST dataset with 60000 training images of handwritten numbers and ImageNet with over 14 million labeled images for object recognition tasks. For novel settings, the labeled data has to be generated before the learning can take place. In navigation settings, SL is often called Behavioural Cloning (BC) or Imitation Learning (IL), depending on the scenario. For navigational tasks the data might consist of the readings of the surrounding from the different sensors and the label is the velocity generated from the algorithm from which the neural network is trying to imitate. The learning is done through the use of a loss function that compares the generated output from the neural network to the label or correct value. Different loss functions are used for different purposes, e.g. for classification tasks cross-entropy loss is often used and for regression tasks mean square error (MSE) is a common choice. For this thesis the latter type of loss functions, i.e. those for regression tasks, are used.

2.2.2 Reinforcement learning

In contrast to SL, RL does not need any training data to be generated in order for it to learn. Instead, it learns from directly interacting with the environment. The classical terminology is that the agent acts in the environment and is provided feedback on its actions, often as a numerical reward which it tries to maximize during learning. For this to work, the agent needs to be able to observe the environment, i.e. the state, and have a set of actions that can affect the environment. The problem is formulated as a Markov Decision Process (MDP) and includes three parts: state, action, and goal. The three parts are highly dependent on the setting of the problem, as well as what the wanted learning outcome is. The agent learns how to map states to actions, also known as the policy, in order to maximize the reward and the goal is to find the optimal policy, i.e. find the action in each state that maximizes the reward. The reward is the feedback signal from the environment telling the agent if the performed action was good or bad and constructing this feedback signal is a crucial part of reinforcement learning [Sutton and Barto, 2018]. The feedback signal can be constructed in many different ways to generate a certain behavior. The agent can be penalized for every timestep it has not reached the goal to teach the agent that reaching the goal fast is important. Alternatively, the agent can be penalized if it is deviating from the global plan, teaching the agent that following the global plan is important. Choosing which combinations of penalties and rewards can be a challenging task and the chosen combination might not always generate the wanted behavior.

3

Methods

This chapter begins by presenting the mobile robot and the software used in the thesis and the path-planning algorithms used during the data generation. The chapter then presents the construction of the simulated environments, the data generation, the neural network architecture, and how the performance evaluation is done.

3.1 System

The mobile robotics platform used in this thesis is an Unmanned Ground Vehicle (UGV) and specifically the Husky platform from Clearpath Robotics, see Fig. 3.1. UGVs are used in applications where it may be inconvenient, dangerous, or impossible to have a human operator present. The Husky is a four-wheeled differential drive robot that can be equipped with multiple sensors, e.g. LiDAR, cameras, GPS units, and IMUs, but for this thesis's application only the data from a LiDAR sensor is used as input. The Husky can be used in many different environments with varying and challenging terrain.

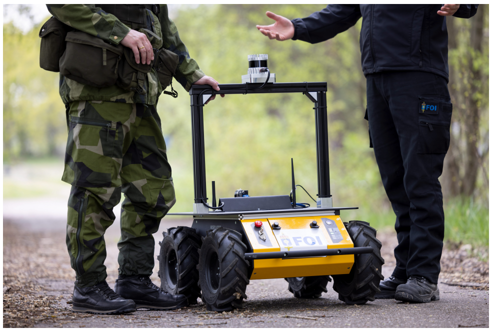


Figure 3.1 The UGV considered in the thesis, the Husky platform equipped with a LiDAR sensor. A simulated version of the Husky is used during data generation and testing.

3.1.1 Software

Testing and generation of training data are done in the simulation tool Gazebo [Koenig and Howard, 2004]. It is an open-source 3D robotics simulator with many built-in tools and functionalities that allows for the simulation of complex robotics tasks in various environments, see Fig. 3.2. In that way, different algorithms, approaches, and parameter settings can be tested before being implemented on the physical robot. For many commercial robots, there exist libraries with the robot prebuilt with various localization and mapping algorithms implemented. Therefore the focus can instead be on further development and improvement of new algorithms. Inside Gazebo, the interface for the robot is Robot Operating System (ROS), which is a set of software libraries and tools that help build robot applications. For this work, an advantage is that there are libraries with global and local planners that easily can be interchanged, which provides an easy way to test the performance of different algorithms. Inside ROS there is an application called RViz, which is used to visualize the robot and sensor data from Gazebo. In Fig. 3.3 a screenshot from RViz is displayed where the red circles are the visualization of the LiDAR point cloud.

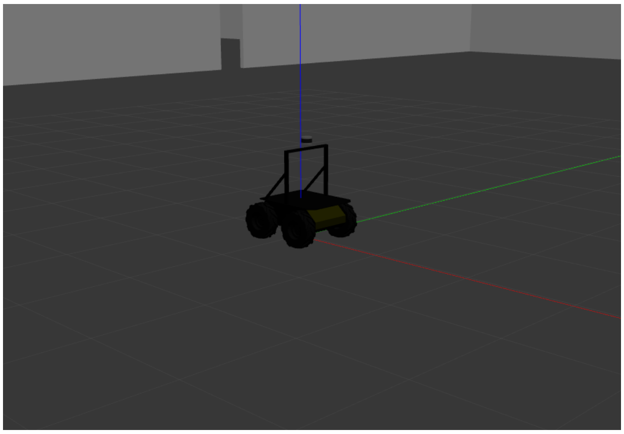


Figure 3.2 The Husky platform equipped with a LiDAR sensor inside the simulation tool Gazebo, where all the data generation and testing is done.

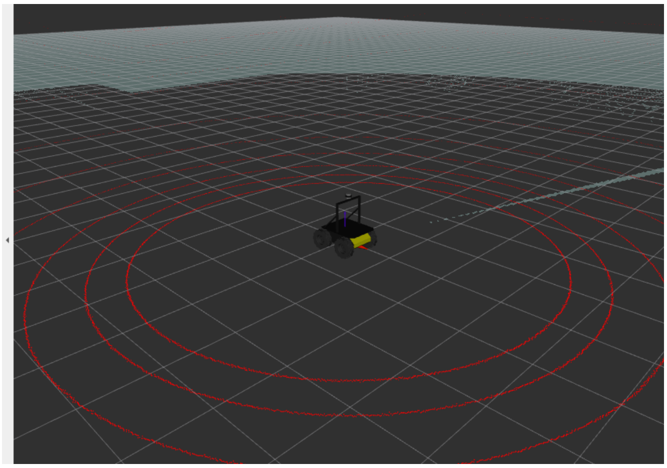


Figure 3.3 The environment in Fig. 3.2 visualized in RViz, where the LiDAR point cloud is visualized as red dots. In RViz the different sensor readings from Gazebo can be visualized.

3.2 Path planning algorithms

Based on the results presented in [Filotheou et al., 2020] the best combination of planner, inside the ROS library, is `navfn` as global planner and `teb_local_planner` as local planner. However `navfn` assumes that the robot is circular, which the Husky is not. Luckily the global planner `global_planner` has almost the same performance as `navfn` but it does not assume a circular robot [Filotheou et al., 2020]. Therefore the choice of global and local planner for this thesis is `global_planner` and `teb_local_planner`.

3.2.1 Global planner

The planning algorithm used for `global_planner` is based on A* and Dijkstra's algorithm, which are common choices for finding the shortest path between points. The focus of this thesis will be on the local planner so no effort to optimize and tune the global planner will be made.

3.2.2 Local planner

Timed Elastic Band (TEB) is a state-of-the-art local planner that is often used for navigation tasks. TEB discretizes the trajectory along the prediction horizon in terms of time and applies a continuous numerical optimization scheme. The global path is split into a number of intermediate robot poses and aims to find the time-optimal trajectory from the current state to the next state or pose [Rösmann et al., 2012] [Rösmann et al., 2017]. The planned trajectories are closer to the actual optimal solution, but constraints are implemented as penalties only, therefore precautions have to be taken in order for constraints to be met. It is suited for all robot types and can handle dynamic obstacles. However, it requires a large computational burden for its updates. TEB has many tunable parameters and they have a significant effect on how the local planner performs and the settings can be highly dependent on the environment. So having optimized TEB in one environment the performance when transferred to a new environment might drop significantly.

3.3 Initial work

The construction of the simulated environment was done in Gazebo. The simulation software has a built-in building editor with the possibility to manually create buildings with the wanted properties. In Fig. 3.4 the map of the created training environment is shown where the different starting points for the UGV during the data generation are displayed as red arrows indicating the orientation of the UGV. The building can be seen in Fig. 3.5 but without the roof added. The Husky is located at the starting point approximately at (5, 10) in Fig. 3.4 pointing to the right. The visualization of the sensors in Gazebo is shown in Fig. 3.6, where the red points are the LiDAR point cloud and the blue regions are a costmap that is used by the planner to construct the global and local plans and avoid walls and obstacles. In addition to the training environment displayed in Fig. 3.4 another environment was built, the validation environment, and its map can be seen in Fig. 3.7. The validation environment is used to mitigate the risk of overfitting to the attempts from the training environment. Evaluation of the local planners will be done in both the training environment as well as the validation environment.

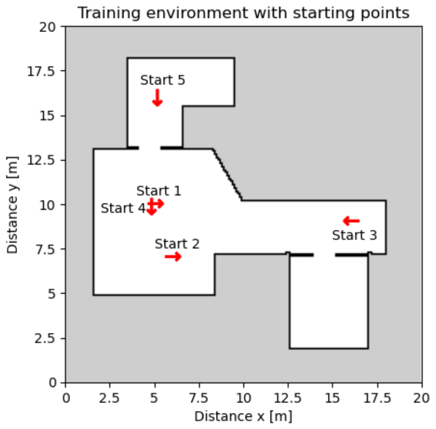


Figure 3.4 Map of the training environment with the five starting points (red arrows) used during data generation. The black lines are walls, the white areas are inside the building and the grey areas are outside. Only goal points in the white areas are viable.

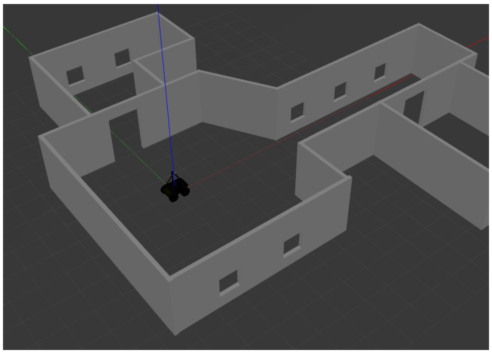


Figure 3.5 Simulation environment Gazebo where the Husky is located inside the training environment, whose map is displayed in Fig. 3.4. The roof has been removed to be able to see the UGV.

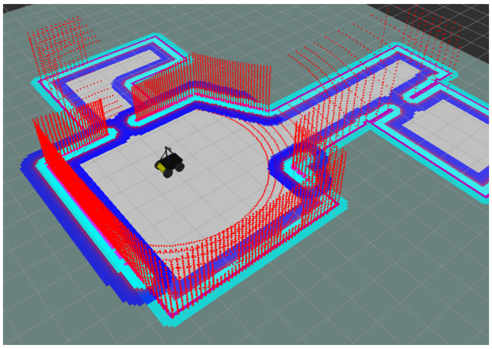


Figure 3.6 The environment in Fig. 3.5 visualized in RViz. The red points are the LiDAR point cloud and the blue regions are the costmap, which is used by the optimization algorithm to avoid walls and obstacles.

Having the two environments constructed, the necessary software for simulating the UGV in the environment was created. As mentioned in Section 3.1 the UGV used in this thesis is the Husky platform from Clearpath Robotics. There exist multiple libraries created by Clearpath Robotics that handle everything from bringing up the simulated Husky in Gazebo and its physics to the navigation stack. Also, different sensors and configurations are available, e.g. LiDAR sensors, cameras, and IMUs. These libraries are both used in the simulated setting but could also be used on the real Husky. In this way, new sensors can be tested and evaluated in the simulated environment before being introduced to the physical UGV.

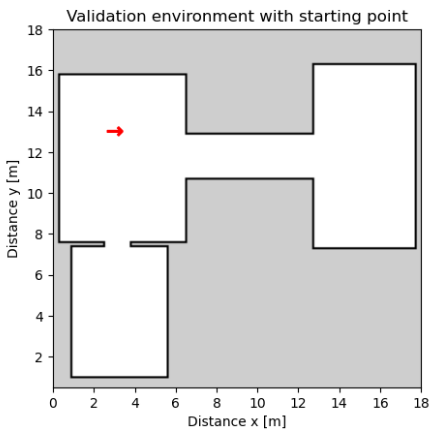


Figure 3.7 Map of the validation environment with the starting point (red arrow) used during data generation. The validation environment is used to minimize the risk of overfitting.

3.4 Data generation and preprocessing

The training data is, as mentioned in Section 3.1.1, generated in Gazebo where the UGV is provided with a large number of randomized points in the environment, and the observations and actions chosen by TEB are recorded. The recorded data consists of LiDAR data, position and angle of the UGV, velocities published by TEB, and lastly, the global plan generated by `global_planner` mentioned in Section 3.2.1. In Fig. 3.8 the recorded data for one attempt is shown, except the LiDAR data. All of the data is recorded in a so called "rosbag", where all the messages are saved. The global plan is published once before the UGV starts moving, and position and angle feedback are being published with a frequency of 5 Hz. The frequency of the LiDAR data can manually be set and the choice was made to set it to 5 Hz as well to match the frequency of the feedback from the system. Another option would be to set the frequency higher for the LiDAR data and use multiple frames for each point of feedback from the system, but for simplicity's sake, the frequency was kept at 5 Hz. In Fig. 3.9 the randomized goal points are shown for the five starting points. 50 goal points are recorded for each starting point and since the goal points are randomly generated some end up on the same coordinates, i.e. there are not 50 individual points for each starting point in Fig. 3.9. The validation data consists of 50 attempts in the validation environment, see Fig 3.10.

Before the network could use the recorded data as input it had to be formatted in a way suitable for the network. The recorded rosbags are processed and the relevant information is extracted from the recorded messages. The LiDAR data is an important input, however it can often look similar for different goal points, especially if the UGV starts from the same position in each simulation. So how should the

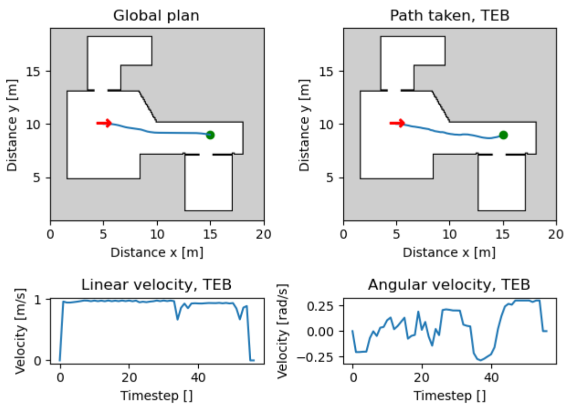
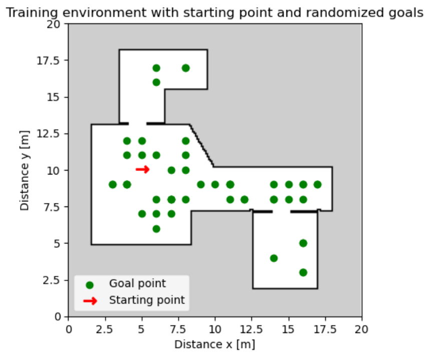


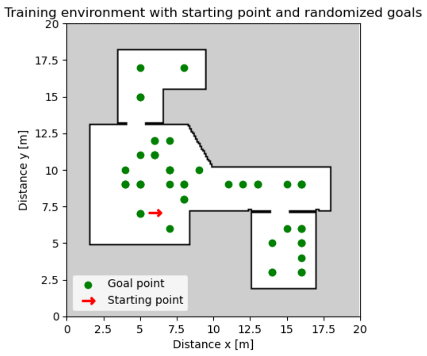
Figure 3.8 One example trajectory in the training data, with starting point (red arrow), goal point (green), and corresponding linear and angular velocities generated by TEB. The top left plot displays the global plan generated by global_planner and the second plot shows the path the UGV followed using TEB. The data presented is recorded for each start-goal pair and it is used when training the network.

UGV know where to go? Another input should be added to give the network an idea of where the goal point is and how the UGV should be steered to reach this goal. Therefore in each timestep, the input to the network should consist of the LiDAR data and some other input that indicates where the goal is placed. This other input has throughout the thesis changed to give a better performance of the network.

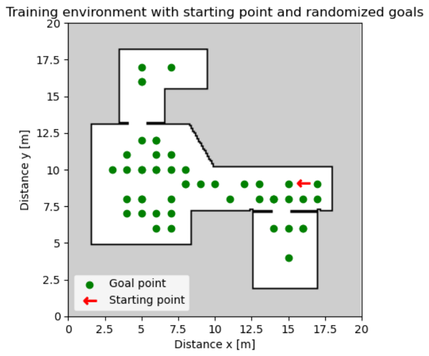
The first idea was at each timestep to give the distance and angle from the current position of the UGV to the goal point. Using this input gave the network some indication of where to go but it had a hard time handling goal points behind walls where a detour is needed to reach the goal. The next idea was to make better use of the information in the global plan instead of only the goal point. The global plan was split up into several waypoints depending on the distance from the start to the goal. Distance and angle to the waypoints gave better results compared to only using the goal point, however when examining the performance, the distance to the waypoint provided no positive effect on the network. The final inputs used in each timestep are LiDAR data as well as the angle from the current position of the UGV to a waypoint. In Fig. 3.11 the waypoints are displayed for one example trajectory and the corresponding angle to waypoint used as input to train the network is also shown. In Fig. 3.12 an illustration of the angle to the waypoint used as input to the network is shown. The angle is calculated in reference to the local coordinate system (grey) of the Husky.


Training environment with starting point and randomized goals
Distance y [m]
Distance x [m]
Goal point
Starting point

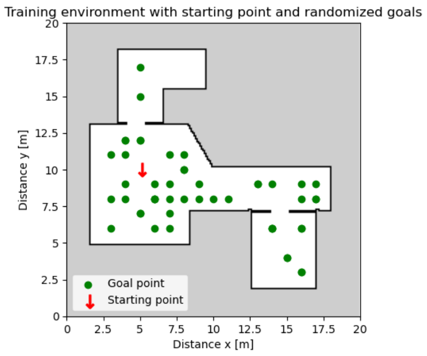
(a) Starting point 1 with randomized goals


Training environment with starting point and randomized goals
Distance y [m]
Distance x [m]
Goal point
Starting point

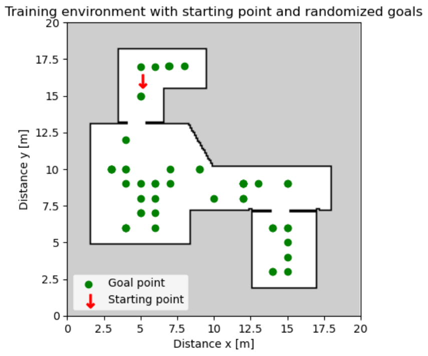
(b) Starting point 2 with randomized goals


Training environment with starting point and randomized goals
Distance y [m]
Distance x [m]
Goal point
Starting point

(c) Starting point 3 with randomized goals


Training environment with starting point and randomized goals
Distance y [m]
Distance x [m]
Goal point
Starting point

(d) Starting point 4 with randomized goals


Training environment with starting point and randomized goals
Distance y [m]
Distance x [m]
Goal point
Starting point

(e) Starting point 5 with randomized goals

Figure 3.9 Training environment with randomized goals for the five starting points, described in Fig. 3.4, used for generating the training data. For each start-goal pair an attempt is recorded, where the local planner used is TEB and the recordings are to be used when training the neural network.

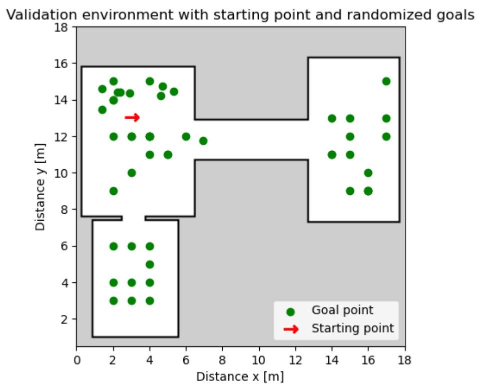

Validation environment with starting point and randomized goals
Distance y [m]
Distance x [m]
Goal point
Starting point

Figure 3.10 Validation environment with randomized goals, used for data generation. For each start-goal pair, an attempt is recorded, where the local planner used is TEB. The recordings are used as the validation dataset to mitigate the risk of overfitting to the training dataset.

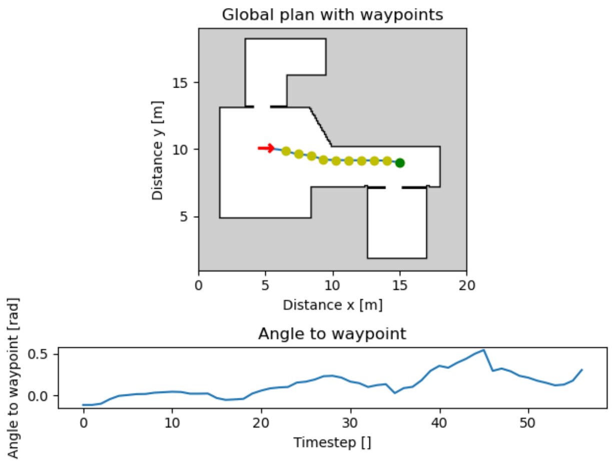


Figure 3.11 Global plan with waypoints for the example trajectory in Fig. 3.8 and the corresponding angle to the waypoint which is used as input for the network. Starting point (red arrow), goal point (green). The angle to waypoint is used to teach the network to follow the global plan.

The LiDAR data generated in Gazebo is a point cloud with 32 vertical lines, each with 256 samples and this point cloud is translated to a depth image with the dimensions 32x256, where each index is the distance from the sensor to the point that corresponds to that index. In Fig. 3.13 five depth images are shown for five timesteps of the example trajectory in Fig. 3.8. Index 0 and 255 lie next to each other behind the UGV. The number of samples can be increased to 2048 samples

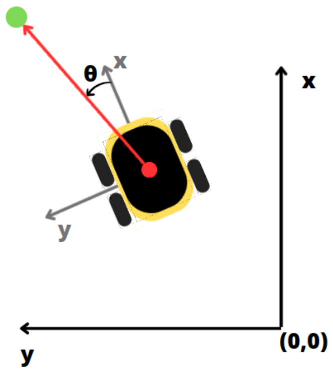
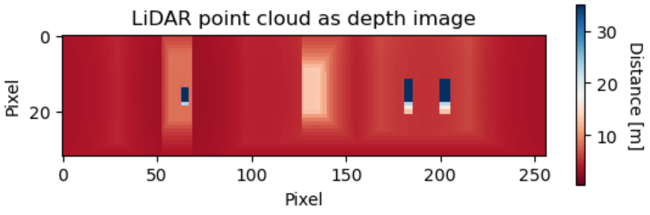
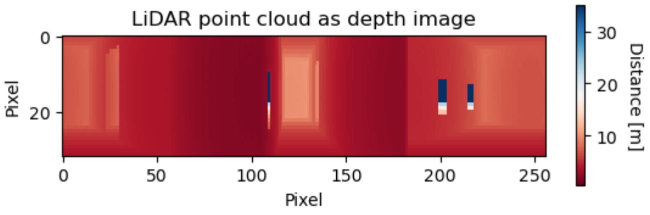


Figure 3.12 Angle to waypoint θ used as input for the network in relation to the global coordinate system, where (0, 0) is the starting point. The green point is a waypoint and the black/yellow shape is the UGV. The angle to waypoint is calculated as the angle from the UGVs local coordinate system. The angle lies in the interval $(-\pi, \pi]$, where 0 is straight ahead and π is behind the UGV.

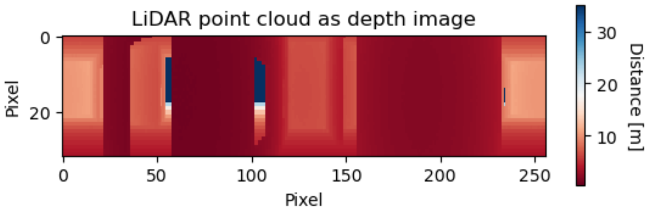
instead of 256 if more details of the environment are needed. After evaluation, 256 samples provided enough information for the application of this thesis while keeping the size of images smaller allowing for faster computations. The LiDAR data is not always complete with all 32x256 points since some points might lie outside the maximum range of the LiDAR sensor. For those missing points, the distance is set to the maximum distance, which in this case is set to 100 m. Before the LiDAR data is fed to the network, the depth image is normalized between 0 and 1, i.e. divided by the maximum distance. The angle input to the network is a number in the interval $(-\pi, \pi]$ and is calculated in each timestep using the feedback of the UGVs position and orientation which is then related to the location of the waypoint.



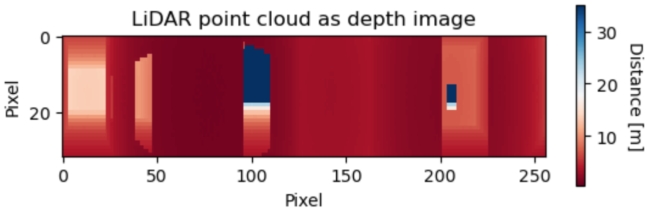
(a) LiDAR depth image for starting point



(b) LiDAR depth image during path



(c) LiDAR depth image during path



(d) LiDAR depth image for goal point

Figure 3.13 LiDAR point cloud translated to depth images for four timesteps in the example trajectory in Fig. 3.8. It shows the distance from the UGV to each point in the point cloud in the LiDAR data. The images show the inside of the building with the blue rectangles being windows and the red areas being walls with different distances from the LiDAR sensor. The image wraps around, meaning that the left and right sides of the depth image lie next to each other behind the UGV.

3.5 Network architecture

To simplify the creation and implementation of the neural network, the deep learning API Keras is used [Chollet et al., 2015]. Keras makes it easy to build networks with prebuilt layers using a very simple syntax. There even exist pre-trained networks that easily can be tested and used for other applications.

Since the LiDAR data can be structured as an image, the idea is to let such an image go through a pre-trained network that is trained on image classification tasks. This way of using knowledge gained from solving one task to another is called transfer learning. In that way, good features of image analysis can be extracted without having to train a new network on vast amounts of data. The last classification layer can be omitted from the pre-trained network and the pre-trained weights can be frozen so no updates will take place for this part of the network. The chosen pre-trained network is ResNet50V2 introduced in [He et al., 2016]. It is an improved version of ResNet50 introduced in [He et al., 2015] and it is a residual network. In this case, it is based on Convolutional Neural Networks (CNN), which are used for computer vision applications. ResNet50V2 has 109 layers with 25.6 million parameters. The pre-trained weights used have been trained on image classification tasks on ImageNet mentioned in Section 2.2.1. The choice of image classification network could easily be interchanged for a different model in Keras and the results would probably be similar for other models trained on ImageNet.

After the LiDAR data has been passed through the pre-trained image classification the output is flattened and the angle input is concatenated. Following the flattening and concatenation there are either Long Short-Term Memory (LSTM) blocks or vanilla fully connected layers, to evaluate the effect of the temporal dependency in the input data. During testing, different network structures have been evaluated e.g. concatenating the angle input on different places in the network and repeating the angle input in a vector instead of just one number to achieve more connections to the angle input to the next layers. Concatenating the angle input as a vector of length 1000 seemed to influence the network to the extent needed, i.e. the network started to steer towards the waypoint instead of always taking the same path. The number of nodes in the fully connected layers decreases in each layer until the new output layer, which is trained to deliver suitable linear and angular velocity. The optimizer used is ADAM, the loss function is Mean Squared Error (MSE) and the learning rate is 0.001 and decreases when the training loss stagnates. The network is trained with data generated from simulations recorded using TEB. The LiDAR data and the angle input are passed through the network and the output is compared to the velocities generated from TEB, calculating the loss. In that way, the network should learn basic behavior from TEB. The two network architectures are shown in Fig. 3.14, where NN is the neural network without LSTM and NN-LSTM is the neural network with LSTM.

The training of the networks took a couple of hours, depending on the structure of the network, using a NVIDIA GeForce GTX 1080 Ti. For details about how the neural network was incorporated as a local planner, please consult Appendix A. In the appendix, the Robot Operating System (ROS) is briefly explained and the structure of the local planner node is discussed.

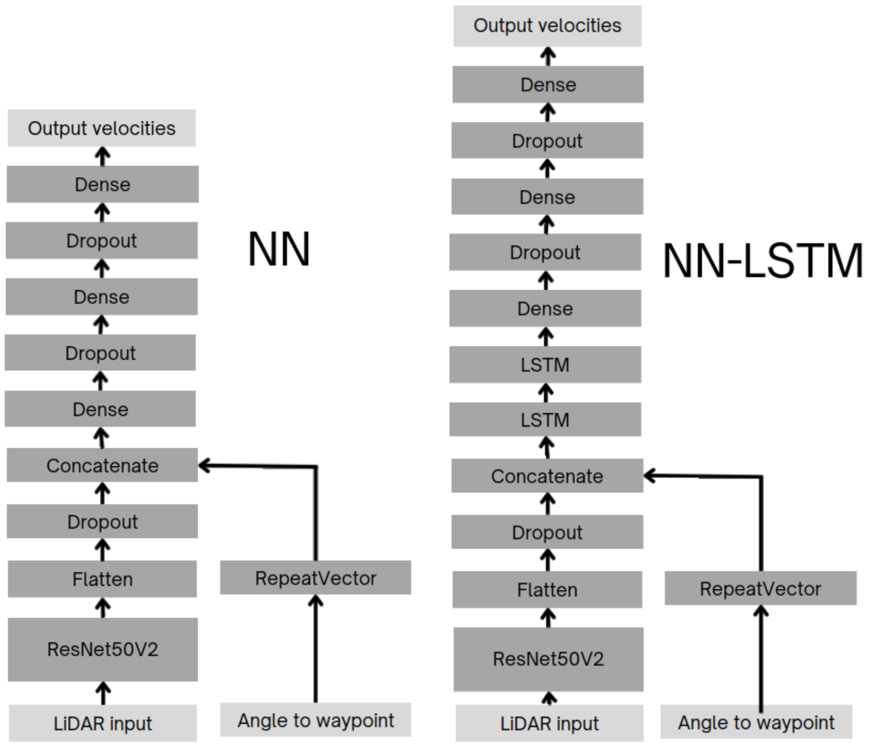


Figure 3.14 Network architectures for the two networks used in this thesis. NN-LSTM is constructed in the same way as NN but with two LSTM layers added. The input to the networks is LiDAR data and angle to waypoint and the output is the linear and angular velocity for a given timestep.

3.6 Evaluation

The three different local planners that are compared are listed below:

1. Timed Elastic Band, TEB
2. Neural network without LSTM, NN
3. Neural network with LSTM, NN-LSTM

The performance of the local planners is evaluated in a number of ways:

- Success rate of reaching the goal, where the tolerance is 20 cm from the goal
- Time to reach the goal
- Distance traveled
- Generalization to novel settings and environments

Evaluation of the local planners is done in both the training environment as well as the validation environment. The first test was done in the training environment with the same starting points as when generating training data and five goal points were selected to give a good distribution of goal points, see Fig 3.15. For each start-goal pair, five attempts were recorded and the performance was evaluated. The reason for evaluating the performance in the training environment is to see how the trained neural networks perform in the environment they were trained in. If they fail to perform even in this favorable setting they will have a hard time when introduced to new environments. In the validation environment two starting points were chosen and five goal points, see Fig. 3.16. An obstacle is also added to the validation environment to test how the local planners handle the situation, Fig. 3.17. Five attempts were recorded for each start-goal pair, with and without the obstacle, corresponding to a total of 100 attempts for the validation environment.

A run is classified as failed if the UGV collides with a wall or if 200 timesteps are passed. The number of timesteps is chosen based on the size of the environments, 200 timesteps should give the planner more than enough time to reach the goal with a margin if the local path is not optimal. 200 timesteps with the frequency 5 Hz corresponds to 40 s.

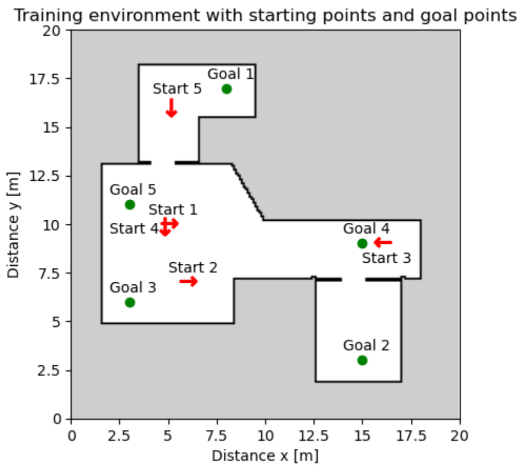


Figure 3.15 Starting points and goal points inside the training environment used for evaluation of the different local planners. The goal points are chosen to get a good distribution of locations in the environment.

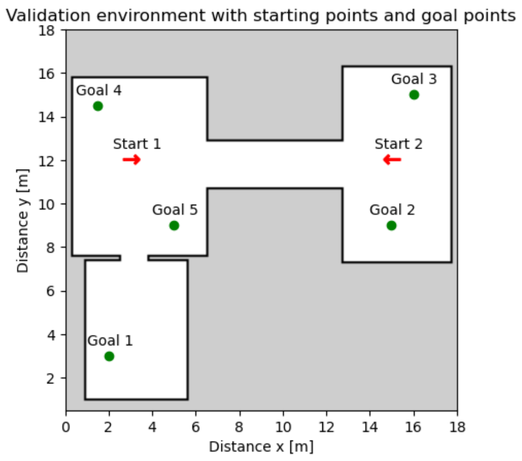


Figure 3.16 Starting points and goal points inside the validation environment used for evaluation of the different local planners. The goal points are chosen to get a good distribution of locations in the environment.

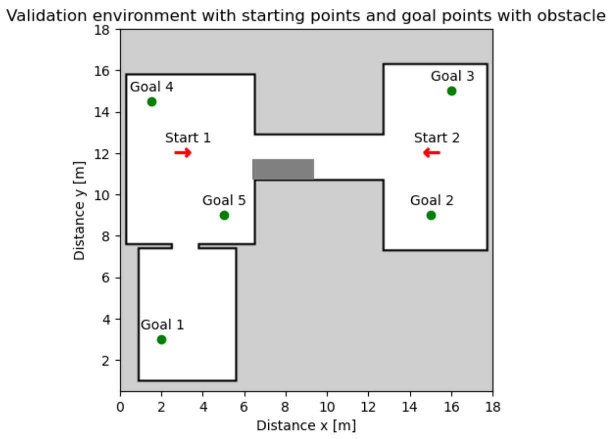


Figure 3.17 Starting point and goal points inside the validation environment, see Fig. 3.16, but with an additional obstacle (dark grey) added to the corridor.

4 Results

The results are split into two sections: results from the training environment and results from the validation environment. In each part, the performance metrics discussed in Section 3.6 are evaluated. No models have trained on any data from the validation environment, they have only seen the data during validation where no training takes place. Evaluating the two networks in the validation environment will give an indication of how the networks generalize to novel settings. The average time, average distance, and distance of the global plan are calculated for successful attempts. Including the failed attempts would skew the results since the attempts with collisions are cut short and the attempts that are timed out are also cut short. Since only the successful runs are part of the metrics this has to be taken into account when comparing the metrics between local planners. If the local planner tends to succeed on goal points far from the starting point but fails to reach goal points close to the starting point then the average time and average distance will be higher. To mitigate this problem the average distance is compared to the global plan for that particular attempt, i.e. if the global plan is 10.0 m and the local planners generate a path of 10.4 m then the path traveled is 104.0% of the global plan. For the average time, it is harder to construct a good metric. If the average global plan is much longer when compared to the other local planners then it is expected to have a longer average time as well.

4.1 Training environment

In Table 4.1 the results from the training environment are displayed. The numbers are calculated from five attempts from 25 start-goal pairs, i.e. 125 total attempts. In Appendix B.1 the results for each start-goal pair are displayed. NN fails to learn good behavior from the training data with only a 40.0% success rate. TEB and NN-LSTM have a similar success rate in the training environment, however, NN-LSTM is a bit slower in reaching the goal and the path is a bit longer. In Fig. 4.1 one of the recorded attempts is displayed, where TEB and NN-LSTM reach the goal, whereas NN collides with a wall.

	TEB	NN	NN-LSTM
Success rate	**97.6%**	40.0%	95.2%
Average time to reach the goal for successful runs [timesteps/seconds]	**72.7 timesteps / 14.5 s**	82.2 timesteps / 16.4 s	81.6 timesteps / 16.3 s
Average distance to goal for successful runs [m]	**9.15 m**	10.84 m	9.37 m
Average distance of the global plan for successful runs [m]	**9.03 m**	10.54 m	9.08 m
Distance travelled divided by distance of global plan [%]	**101.3%**	102.9%	103.2%

Table 4.1 Performance metrics for the three local planners: TEB, NN, and NN-LSTM in the training environment. TEB performs slightly better than NN-LSTM, which in turn performs significantly better than NN.

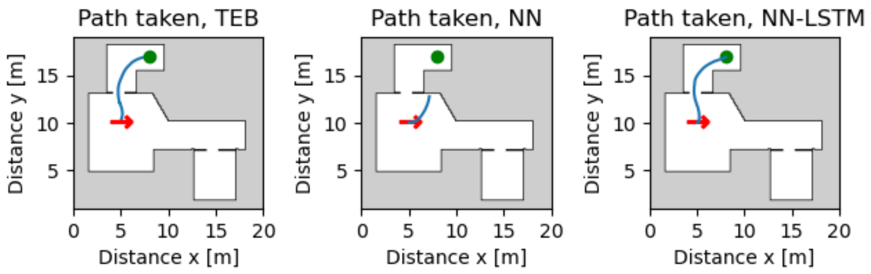


Figure 4.1 Path taken during testing for the three local planners: TEB, NN, and NN-LSTM, for one attempt between Start 1 and Goal 1 in Fig 3.15. NN failed to reach the goal and collided with the wall.

4.2 Validation environment

4.2.1 Start 1

Based on the poor performance of NN in the training environment the decision was made to only test NN on one starting point in the validation environment to evaluate if more time should be put into generating additional attempts. In Table 4.2 the performance of the three local planners on the five start-goal pairs starting from Start 1, see Fig. 4.2, are displayed. The results for each start-goal pair can be found in Appendix B.2. The success rate is significantly lower for NN (52.0%) compared to TEB (100.0%) and NN-LSTM (88.0%), therefore the decision was made to continue with only TEB and NN-LSTM.

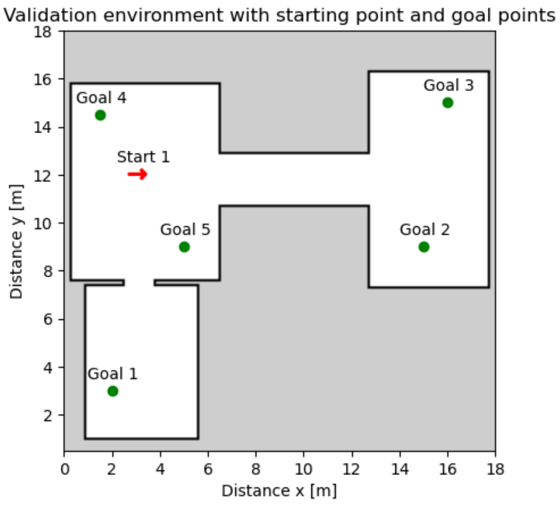


Figure 4.2 Start 1 and goal points. Same as Fig. 3.15 but without starting point 2.

	TEB	**NN**	**NN-LSTM**
Success rate	**100.0%**	52.0%	88.0%
Average time to reach the goal for successful runs [timesteps/seconds]	**60.1 timesteps / 12.0 s**	87.2 timesteps / 17.4 s	82.8 timesteps / 16.6 s
Average distance traveled to goal for successful runs [m]	**8.65 m**	9.15 m	9.65 m
Average distance of the global plan for successful runs [m]	8.68 m	**8.30 m**	9.46 m
Distance travelled divided by distance of global plan [%]	**99.7%**	110.2%	102.0%

Table 4.2 Performance metrics for the three local planners: TEB, NN, and NN-LSTM in the validation environment, only from Start 1 without obstacle corresponding to 25 total attempts per local planner. TEB performs better than NN-LSTM, which in turn performs significantly better than NN.

4.2.2 Start 1 and Start 2 with and without obstacle

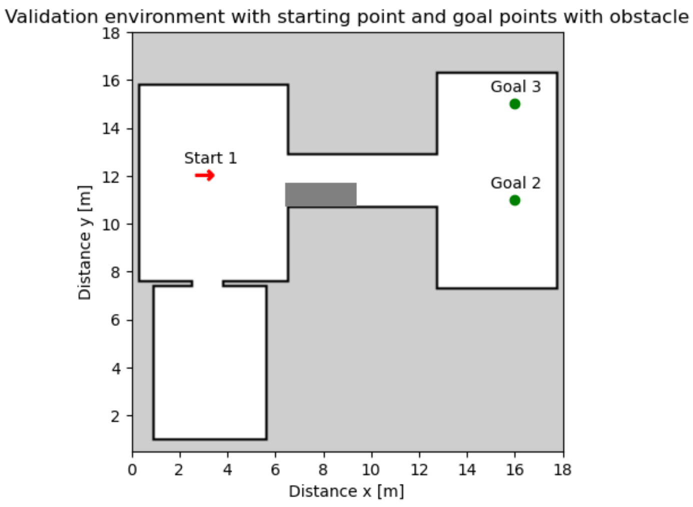
In Table 4.3 the performance metrics evaluated in the validation environment are displayed for TEB and NN-LSTM, for the two starting points with and without the obstacle corresponding to 100 total attempts per local planner. The performance between the two local planners is pretty similar, but if the attempts that force the UGV to pass the obstacle are isolated, the performance metrics tell a different story. The isolated attempts can be seen in Fig. 4.3 and for these attempts, the performance metrics are displayed in Table 4.4, see Appendix B.2 for details. TEB has a hard time passing the obstacle and has a success rate of only 44.0% and the attempts that manage to reach the goal are almost 10 s slower than NN-LSTM. In Fig. 4.4a and 4.4b two examples are shown where TEB gets stuck and NN-LSTM succeeds. In Appendix C, trajectories for TEB and NN-LSTM are displayed for Start 1 and 2 without the obstacle.

	TEB	**NN-LSTM**
Success rate	85.0%	**86.0%**
Average time to reach the goal for successful runs [timesteps/seconds]	**72.0 timesteps / 14.4 s**	93.3 timesteps / 18.7 s
Average distance traveled to goal for successful runs [m]	**8.76 m**	10.02 m
Average distance of the global plan for successful runs [m]	**8.68 m**	9.66 m
Distance traveled divided by distance of global plan [%]	**101.0%**	103.7%

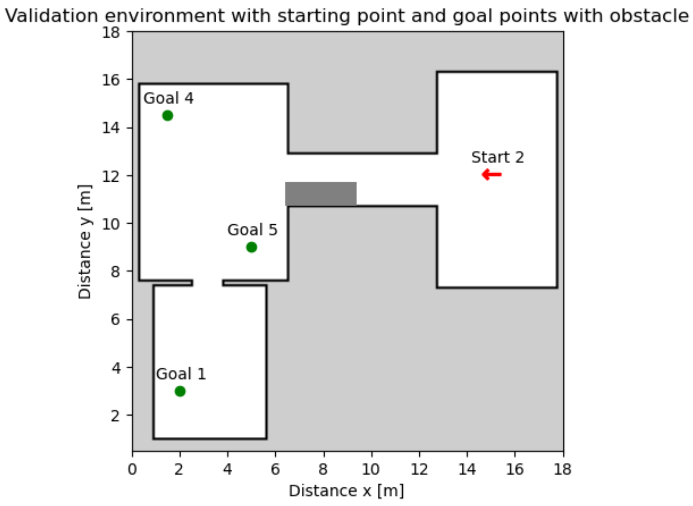
Table 4.3 Performance metrics for TEB and NN-LSTM in the validation environment for the start-goal pairs in Fig. 3.16 and 3.17. 100 total attempts per local planner. NN-LSTM has a slightly higher success rate compared to TEB, however TEB has a shorter average time and average distance.

	TEB	**NN-LSTM**
Success rate	44.0%	**80.0%**
Average time to reach the goal for successful runs [timesteps/seconds]	152.1 timesteps / 30.4 s	**103.2 timesteps / 20.6 s**
Average distance traveled to goal for successful runs [m]	15.14 m	**14.62 m**
Average distance of the global plan for successful runs [m]	**14.57** m	14.74 m
Distance traveled divided by distance of global plan [%]	103.9%	**99.2%**

Table 4.4 Performance metrics for TEB and NN-LSTM in the validation environment. The start-goal pairs that pass the obstacle are isolated, see Fig. 4.3 for a visualization of these points. 25 total attempts per local planner. NN-LSTM performs significantly better than TEB.

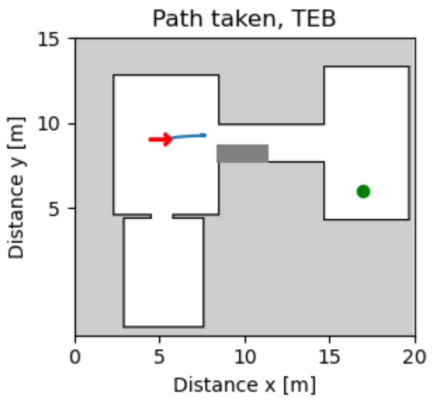

Validation environment with starting point and goal points with obstacle
Goal 3
Start 1
Goal 2
Distance y [m]
Distance x [m]

(a) Starting from Start 1 and going to Goal 2 and 3.

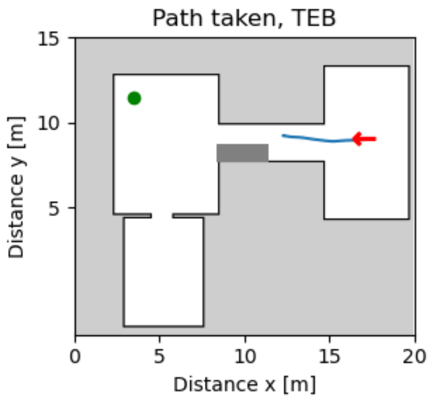

Validation environment with starting point and goal points with obstacle
Goal 4
Start 2
Goal 5
Goal 1
Distance y [m]
Distance x [m]

(b) Starting from Start 2 and going to Goal 1,4 and 5.

Figure 4.3 Validation environment with starting points and goals where the path passes the obstacle. It is an extraction of start-goal pairs from Fig. 3.17. These start-goal pairs are used when calculating the results in Table 4.4.



(a) Path taken for TEB and NN-LSTM for one attempt. TEB fails to find a viable path, whereas NN-LSTM passes through the corridor.



(b) Path taken for TEB and NN-LSTM for one attempt. TEB fails to find a viable path, whereas NN-LSTM passes through the corridor.

Figure 4.4 Two example attempts where the path passes the obstacle.

5 Discussion

This chapter begins by discussing the use of a neural network as a replacement for local planners using optimization algorithms. Then the chapter discusses the training of the neural network, the temporal dependency of the input data, and the network architecture. The chapter concludes with a discussion of future work.

5.1 Neural network as replacement

In the below discussion, the comparison between NN-LSTM and TEB will be discussed. NN will for this part be omitted since the performance is that much lower compared to the other two. In the training environment, TEB has a success rate of 97.6%, the path followed is 1.3% longer than the global plan and the average time is 14.5 s. For NN-LSTM the same metrics are 95.2%, 3.2%, and 16.3 s. Based on these numbers the two planners can be concluded to have similar performance. NN-LSTM has a slightly less optimal path and takes a bit longer average time. The network is trained to imitate TEB with data generated from the training environment, so expecting that NN-LSTM should outperform TEB in this environment is unreasonable. It is more interesting to evaluate the performance in novel environments, to decide if a neural network can work as a replacement for TEB. If training would be needed for each new environment it would defeat the purpose of introducing a neural network to avoid having to tune parameters for TEB. In the validation environment, TEB had a success rate of 85.0%, the path followed was 1.0% longer than the global plan and the average time was 14.4 s. These results are presented in Table 4.3, which are calculated for Start 1 and Start 2 with and without the obstacle. For NN-LSTM the metrics are 86.0%, 3.7%, and 18.7 s. The average time is a bit longer for NN-LSTM, however, the average distance of the global plan for the successful attempts is approximately 1.0 m longer which is one contributor to the increase in average time. NN-LSTM has a slightly higher success rate, however, it is a bit slower and has a longer path. The reason for the drop in the success rate for TEB is due to the added obstacle in the corridor. For the attempts where the path passes the obstacle the success rate is only 44.0% compared to NN-LSTM with 80.0% and for the successful attempts TEB is 10 s slower than NN-LSTM, see Table 4.4. For certain scenarios, TEB fails considerably, whereas NN-LSTM performs much better. As expected, when comparing the results for NN-LSTM between the training and validation environment there was a drop in success rate from 95.2% to 88.0%.

What is the reason that TEB struggles to pass the obstacle? The reason for this struggle is based on the parameters used for the optimization algorithm. TEB uses a costmap around each obstacle as a way to determine if the path is close to an obstacle or not and the costmap is influenced by a number of parameters in the setup. In Fig. 5.1a the costmap without the obstacle is shown and there is a clear path in the middle of the corridor where there is no costmap and therefore the UGV is safe to pass here. When adding the obstacle to the corridor, see Fig 5.1b, there is no longer a path that goes through the corridor without going inside the costmap. When the UGV encounters this situation TEB will try to find a way to pass through the narrow path and it often starts to oscillate at the beginning of the obstacle. Sometimes TEB finds a way through after some hesitation but often it gets completely stuck not being able to move past the obstacle. To mitigate this problem the parameters related to the costmap would have to be adjusted to fit the new environment. Since the trained neural network has been trained to imitate TEB without the use of the costmap it is not affected by the overlapping costmap and does not show the same hesitant behavior. However, NN-LSTM does not have a 100.0% success rate passing the obstacle but the problem is not getting timed out and being hesitant to pass the obstacle, instead, the main issue was that it collided with the obstacle on some attempts. There were attempts where the UGV just barely collided with the obstacle or a doorway when passing and still managed to reach the goal. But since an attempt is considered failed if the UGV collides with anything, the attempt was marked as failed.

5.2 Training

The learning approach used in this thesis is very simple and the learning is fast. Supervised Learning (SL) is easy to implement and the training can be done in a short amount of time compared to other machine learning approaches. The drawback with SL is that labeled data is needed for training which often is time-consuming and expensive to obtain. But when the data has been generated, training is an easy process. Using only SL gave results with a performance at the same level as TEB in many situations, and in other situations, the trained network managed to outperform TEB. SL can be a good starting point for imitating behavior to later fine-tune the behavior using other machine learning methods, such as RL. Training from scratch using RL often requires millions of timesteps to learn basic behaviors. SL could

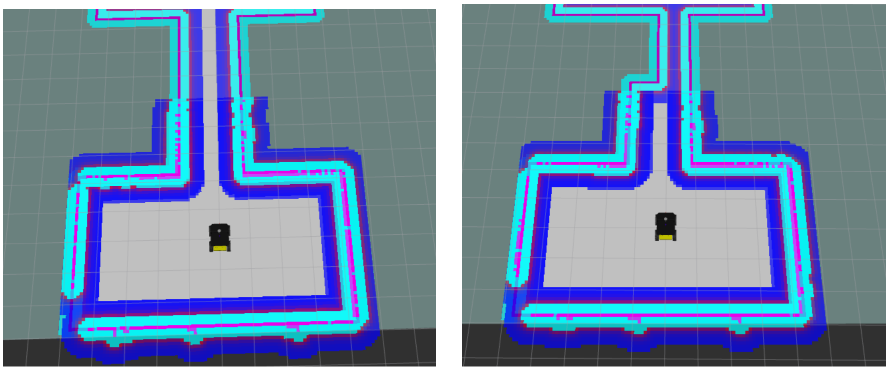

(a) Costmap without obstacle **(b)** Costmap with obstacle

Figure 5.1 Costmaps with and without the obstacle added to the environment, visualized in RViz. In Fig. 5.1a there is a clear path in the middle of the corridor, whereas in Fig. 5.1b the costmap overlaps providing difficulty for TEB.

therefore be a fast and efficient starting point to get a good baseline behavior, from which RL can be applied to make the behavior better. Another alternative to RL that could be used to improve the baseline behavior is to use DAgger [Ross et al., 2010].

The performance of the neural network is highly dependent on the training data. During the testing of different iterations of networks, it was clearly seen in some iterations that the network overfitted on the training data and when introduced to a new starting point or a new environment the UGV would get stuck or deviate greatly from the global plan. The training data would improve if the starting position of the UGV also was randomized for each iteration as it is implemented for the goal points. Unfortunately, the settings for the starting point are not as easily changed as generating goal points so there is no way to make a script to randomize starting points easily. The best option was to choose several starting points in the environment that generates simulations that explore the whole environment from different directions and angles. Since the generation of data is time-consuming, the choice was to use five starting points during data generation in the training environment, see Fig. 3.4, but the number could easily be extended to more starting points if time permitted.

5.3 Temporal dependency

NN and NN-LSTM were trained using the same method but the results were vastly different. In the training environment, the success rate was only 40.0% for NN, whereas NN-LSTM had a success rate of 95.2%. NN fails to capture the temporal dependency in the input data. The optimal action at the current timestep is highly dependent on earlier actions, therefore some sort of memory is needed to best utilize the temporal dependency. NN has a tough time reaching goal points that demand a lot of turning during the first timesteps of the attempt. In Table 4.1 the average global plan of the successful attempts is approximately 1.5 m longer than for the other local planners, indicating that NN struggles with goal points close by that demand a lot of turning. An example of this behavior can be seen in Fig. 4.1. Conclusively, adding LSTM layers gives the network the capacity to capture the temporal dependency of the input data.

Using LSTM is a well-established way of introducing temporal dependency in the network. In recent years the use of Transformers [Vaswani et al., 2017] has developed as a way to introduce positional dependency in sequential data outperforming LSTM.

5.4 Neural network architecture

The choice of the particular network architecture shown in Fig. 3.14 was made after some trial and error when testing. A lot more time could be put into tweaking the number and type of layers, dropout parameters, how to input the angle input, etc. The network architecture for NN-LSTM, shown in Fig. 3.14, gave good enough performance to start generating the results presented in the earlier chapter. To see the effect of having LSTM layers, NN was constructed the same way as NN-LSTM but without the LSTM layers, to get a better comparison.

Using ResNet for handling the LiDAR data can also be changed to another pre-trained network, e.g. MobileNet, or even create a stack of Convolutional Neural Networks (CNN) which would have to be trained from scratch. Since ResNet gave satisfying results, no effort was put into evaluating another structure. Keras has multiple pre-trained networks, so the change to another network would be only as involved as changing the name from ResNet50V2 to MobileNet.

The trained neural network has no clear limit when predicting the velocities. During training it has never trained on any linear velocities larger than 1 m/s and angular velocity of ± 0.3 rad/s, however, there is nothing stopping the network from predicting velocities larger than these values. The velocities have never been far off from the known max, but it is important to keep in mind if the system is sensitive to inputs larger than the max. It is of course easy to add some lines of code that check if the predicted velocity is larger than the max and in that case set the value to the max velocity instead.

5.5 Future work

A possible improvement to the thesis would be to add RL on top of the existing neural network to optimize the generated behavior to outperform TEB. In that way, the UGV could get penalties for colliding with obstacles and therefore learn to avoid them. This could be done using either online or offline RL.

Throughout the thesis, the environment has been static with all obstacles known beforehand. This is not an accurate depiction of what situations the UGV is aimed to be used in. A natural next step is to introduce dynamic obstacles to the environment and use e.g. RL for learning good behavior in handling those obstacles. TEB is known to have issues with handling fast-moving obstacles so this next step could really prove the benefits of using a neural network compared to the existing algorithm.

Since the testing has only been done in the simulated setting the obvious next step would be to implement the local planner on the physical robot and evaluate its performance in the real world.

5.6 Ethical considerations

Navigation systems for UGVs can be used in a variety of applications such as logistics, health care, and military. The one developed during this thesis is not a working solution at this stage, and it is only a small part of the complete navigation system needed for an UGV. However, further development might result in a working local planner that can be used in the navigation system for the applications listed earlier.

With all new development there needs to be a discussion about how this new technology can be used and how accessible it will be. The new technology needs to be regulated and controlled such that it is only used for its intended purpose. FOI works closely with The Swedish Armed Forces and the research is often aimed at being used in a military setting at some point. Therefore, it is of great importance that the research is done in an ethical way and that the development is used in the right setting. A cost-benefit analysis has to be done to make sure that the new development is a net positive to society and does not create any potential risks.

6 Conclusion

This thesis explored the possibility of using a trained neural network as a replacement for state-of-the-art path planning algorithms, with the motivation to avoid having to tune parameters for each new environment and have a less computationally heavy local planner. Neural networks with and without Long Short-Term Memory layers were trained with data recorded using Timed Elastic Band with Supervised Learning as the learning method. The results showed that in many situations a neural network with Long Short-Term Memory layers can perform close to the same level as the planning algorithm and in some situations, the neural network outperformed the algorithm by a big margin. However, there are still some issues using a neural network since the Unmanned Ground Vehicle is more likely to collide with an obstacle or a wall. Despite not being completely satisfactory the results provide a promising baseline for further development of implementing machine learning components into path planning tasks.

Bibliography

Alatise, M. B. and G. P. Hancke (2020). “A review on challenges of autonomous mobile robot and sensor fusion methods”. *IEEE Access* **8**, pp. 39830–39846. DOI: 10.1109/ACCESS.2020.2975643.

Chen, C., Y. Liu, S. Kreiss, and A. Alahi (2018). *Crowd-robot interaction: crowd-aware robot navigation with attention-based deep reinforcement learning*. DOI: 10.48550/ARXIV.1809.08835. URL: https://arxiv.org/abs/1809.08835.

Chiang, H.-T. L., A. Faust, M. Fiser, and A. Francis (2018). *Learning navigation behaviors end-to-end with autorl*. DOI: 10.48550/ARXIV.1809.10124. URL: https://arxiv.org/abs/1809.10124.

Chollet, F. et al. (2015). *Keras*. https://keras.io.

Everett, M., Y. F. Chen, and J. P. How (2018). *Motion planning among dynamic, decision-making agents with deep reinforcement learning*. DOI: 10.48550/ARXIV.1805.01956. URL: https://arxiv.org/abs/1805.01956.

Faust, A., O. Ramirez, M. Fiser, K. Oslund, A. Francis, J. Davidson, and L. Tapia (2017). “PRM-RL: long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning”. DOI: 10.48550/ARXIV.1710.03937. URL: https://arxiv.org/abs/1710.03937.

Filotheou, A., E. Tsardoulias, A. Dimitriou, A. Symeonidis, and L. Petrou (2020). “Quantitative and qualitative evaluation of ros-enabled local and global planners in 2d static environments”. *Journal of Intelligent & Robotic Systems* **98**. DOI: 10.1007/s10846-019-01086-y.

Fox, D., W. Burgard, and S. Thrun (1997). “The dynamic window approach to collision avoidance”. *IEEE Robotics & Automation Magazine* **4**:1, pp. 23–33. DOI: 10.1109/100.580977.

Fragapane, G. I., D. A. Ivanov, M. Peron, F. Sgarbossa, and J. O. Strandhagen (2020). “Increasing flexibility and productivity in industry 4.0 production networks with autonomous mobile robots and smart intralogistics”. *Annals of Operations Research* **308**, pp. 125–143.

Guldenring, R., M. Görner, N. Hendrich, N. J. Jacobsen, and J. Zhang (2020). “Learning local planners for human-aware navigation in indoor environments”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6053–6060. DOI: `10.1109/IROS45743.2020.9341783`.

He, K., X. Zhang, S. Ren, and J. Sun (2015). *Deep residual learning for image recognition*. arXiv: `1512.03385 [cs.CV]`.

He, K., X. Zhang, S. Ren, and J. Sun (2016). *Identity mappings in deep residual networks*. arXiv: `1603.05027 [cs.CV]`.

Kästner, L., J. Cox, T. Buiyan, and J. Lambrecht (2021). *All-in-one: a drl-based control switch combining state-of-the-art navigation planners*. DOI: `10.48550/ARXIV.2109.11636`. URL: `https://arxiv.org/abs/2109.11636`.

Koenig, N. and A. Howard (2004). “Design and use paradigms for gazebo, an open-source multi-robot simulator”. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. Vol. 3, 2149–2154 vol.3. DOI: `10.1109/IROS.2004.1389727`.

Open Robotics (2020). *Move_base [[attachment:overview_tf.png]]*. This work is licensed CC BY 3.0. To view a copy of this license, visit `https://creativecommons.org/licenses/by/3.0/`. URL: `http://wiki.ros.org/move_base`.

Quigley, M., K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng (2009). “Ros: an open-source robot operating system”. In: vol. 3.

Rösmann, C., W. Feiten, T. Woesch, F. Hoffmann, and T. Bertram (2012). “Trajectory modification considering dynamic constraints of autonomous robots”. In: *ROBOTIK 2012; 7th German Conference on Robotics*, pp. 1–6.

Rösmann, C., F. Hoffmann, and T. Bertram (2017). “Integrated online trajectory planning and optimization in distinctive topologies”. *Robotics and Autonomous Systems* **88**, pp. 142–153. ISSN: 0921-8890. DOI: `https://doi.org/10.1016/j.robot.2016.11.007`. URL: `https://www.sciencedirect.com/science/article/pii/S0921889016300495`.

Rösmann, C., A. Makarow, and T. Bertram (2021). “Online motion planning based on nonlinear model predictive control with non-euclidean rotation groups”. In: *2021 European Control Conference (ECC)*. IEEE. DOI: `10.23919/ecc54610.2021.9654872`. URL: `https://doi.org/10.23919%2Fecc54610.2021.9654872`.

Ross, S., G. J. Gordon, and J. A. Bagnell (2010). *A reduction of imitation learning and structured prediction to no-regret online learning*. DOI: `10.48550/ARXIV.1011.0686`. URL: `https://arxiv.org/abs/1011.0686`.

Sutton, R. S. and A. G. Barto (2018). *Reinforcement learning: An introduction*. MIT Press.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin (2017). *Attention is all you need.* DOI: `10.48550/ARXIV.1706.03762`. URL: `https://arxiv.org/abs/1706.03762`.

A

ROS and local planner node

A.1 Robot Operating System

Robot Operating System (ROS) is despite its name, not a real operating system but instead a software framework on top of the operating system that provides a structured communications layer between processes and systems [Quigley et al., 2009]. The ROS framework uses peer-to-peer communication and supports multiple languages, e.g. C++ and Python. Communication is done using nodes, messages, and topics, see Fig. A.1. Processes that perform computation are called nodes, and different nodes send and receive information between each other by passing messages. Messages are published to a specific topic that other nodes can subscribe to. A topic is simply a string that explains the type of message such as "map" or "points". There might exist multiple publishers and subscribers for the same topic and generally, the two are not aware of each other. A node can be a subscriber and a publisher at the same time, subscribing to one topic, performing calculations, and publishing to another topic. ROS provides a standardized approach for the communication between different components in a system, which eases the integration and implementation of new sensors. In ROS there exists a node that handles the movement of a mobile base called move_base. In Fig. A.2 the structure of move_base is shown and it can be seen that there are many components in the architecture. move_base subscribes to topics related to the map, goal point, and different sensors, e.g. odometry and LiDAR and it publishes the wanted linear and angular velocity to the topic `cmd_vel` for which the base_controller subscribes. The base_controller will then translate these velocities to the rotation of the wheels. The focus of this thesis is primarily to make changes to the local_planner inside move_base.

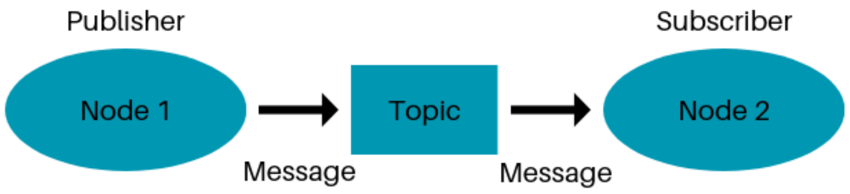


Figure A.1 ROS communication. Information is passed between nodes using messages that are published to topics. A node that sends a message to a topic is called a publisher and one that reads a message from a topic is called a subscriber.

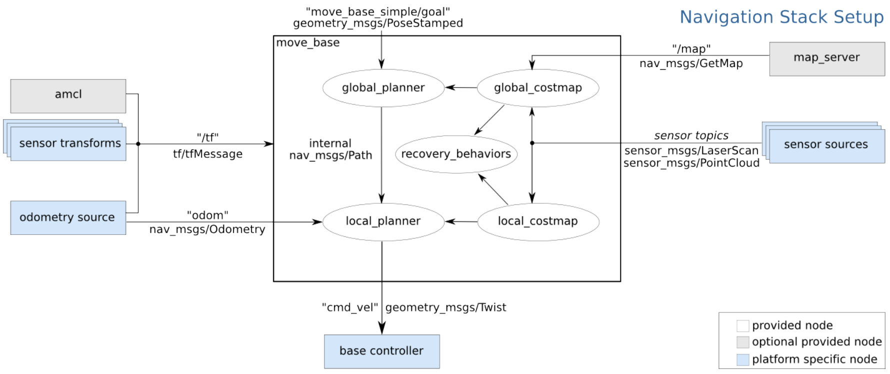


Figure A.2 move_base architecture inside ROS [Open Robotics, 2020]. move_base handles the movement of a mobile base. It gets information from the map_server, localization nodes, sensor sources, etc., and uses this information to create costmaps shown in Fig. 3.6, global plans, and local plans.

A.2 Local planner

Having trained the neural network, the next step was to build a local planner node that can substitute TEB. ROS provides the option to create your own local and global planners and it is easy to change between already existing algorithms and your own to test performance differences. When the local planner node is launched it starts by loading the trained neural network and then it starts subscribing to the topic `move_base/GlobalPlanner/plan`, where the global plan is published. Until the global plan is published there is no need to subscribe to either LiDAR or feedback from the UGV, see Fig. A.3. After the global plan is received the global plan is split up into waypoints, according to Fig. 3.11, which later are used for calculating the angle to the waypoint during the UGVs movement. As input to the neural network, LiDAR data and angle to waypoint are needed, which have to enter as input at the same time. Therefore, there is a need to synchronize the receiving of LiDAR data and feedback from the system, since the messages might arrive at slightly different times. ROS uses message_objects called `TimeSynchronizer` or `ApproximateTimeSynchronizer`, where the latter is used when the messages do not arrive at the same time. The synchronization object subscribes to topic `points`, related to LiDAR, and topic `move_base/feedback`, related to the position and angle of the UGV.

When one message from each topic is received it processes the input as mentioned in Section 3.4, i.e. converts the LiDAR point cloud to a depth image, and based on the position and angle of the UGV calculates the angle to the next waypoint. The processed data is now ready to be used as input for the neural network, and based on the input the network predicts the linear and angular velocities. The output from the network is then published to topic `cmd_vel` that the base_controller subscribes to. The synchronization object will continue to subscribe to LiDAR and feedback and generate velocities to the UGV, either until it gets within a tolerance of the goal point, or if it's manually interrupted by the operator. The internal structure of the local planner node is shown in Fig. A.4.

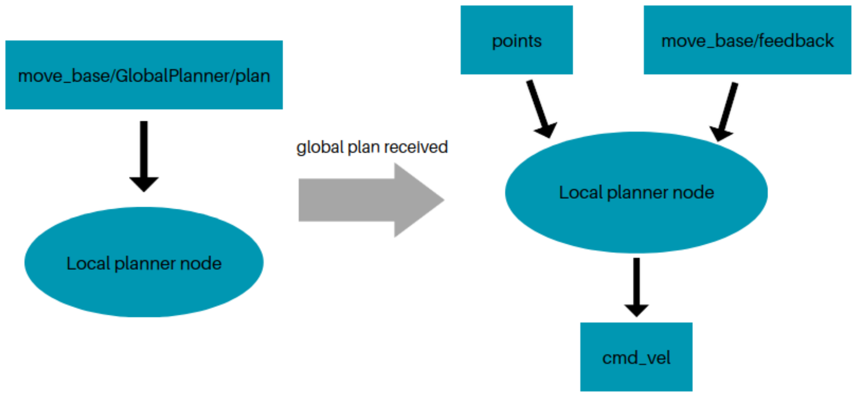


Figure A.3 Local planner node created to replace TEB with a neural network. The node starts subscribing to the topic move_base/GlobalPlanner/plan and when the global plan is received it starts subscribing to points and move_base/feedback and it starts publishing to cmd_vel.

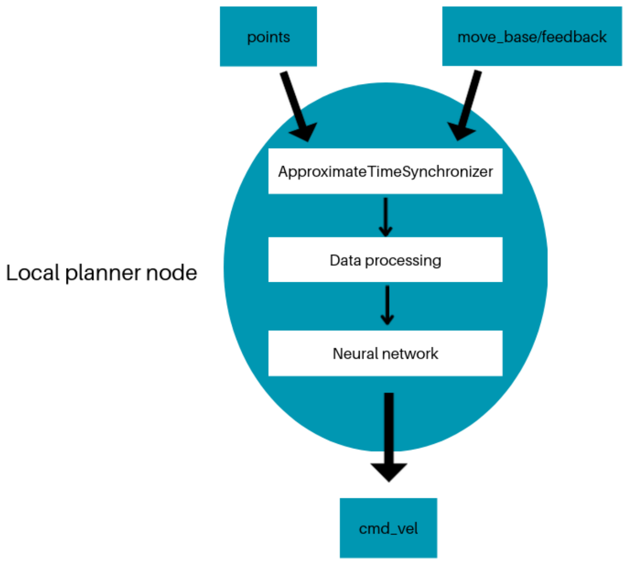


Figure A.4 Internal structure of the local planner node. The node subscribes to the topics points and move_base/feedback, synchronizes the messages, processes the data, feeds it to the neural network, and the output is published to the topic cmd_vel.

B

Results

The below tables are the results from the testing of the three local planners: TEB, NN, and NN-LSTM. A box represents a start-goal pair and each box has five attempts recorded. The number in the box is the number of failed attempts as well as the cause of the failure, either collision or timing-out. An empty box means that all five attempts for that start-goal pair were successful.

B.1 Training environment

Number failed attempts and cause, TEB					
	Goal 1	Goal 2	Goal 3	Goal 4	Goal 5
Start 1		1 T			
Start 2	1 T				
Start 3	1 T				
Start 4					
Start 5					
C=collision, T=timed out					

Table B.1 Number of failed attempts and cause for each start-goal pair, using TEB as local planner. A box represents five attempts.

Number failed attempts and cause, NN					
	Goal 1	Goal 2	Goal 3	Goal 4	Goal 5
Start 1	5 C		5 C		5 C
Start 2	5 T	5 C	5 C	2 C	5 T
Start 3		5 C			
Start 4	5 C	5 T		5 T	2 T
Start 5	5 C	4 C		2 C	5 C
C=collision, T=timed out					

Table B.2 Number of failed attempts and cause for each start-goal pair, using NN as local planner. A box represents five attempts.

Number failed attempts and cause, NN-LSTM					
	Goal 1	Goal 2	Goal 3	Goal 4	Goal 5
Start 1					
Start 2			1 T		
Start 3		1 C			
Start 4	2 C				
Start 5		2 C			
C=collision, T=timed out					

Table B.3 Number of failed attempts and cause for each start-goal pair, using NN-LSTM as local planner. A box represents five attempts.

B.2 Validation environment

Number failed attempts and cause, NN					
	Goal 1	Goal 2	Goal 3	Goal 4	Goal 5
Start 1	3 T, 2 C	3 C	1 C	3 C	
C=collision, T=timed out					

Table B.4 Number of failed attempts and cause for each start-goal pair, using NN as local planner. A box represents five attempts.

Number failed attempts and cause, TEB					
	Goal 1	Goal 2	Goal 3	Goal 4	Goal 5
Start 1					
Start 1 with obstacle		🟦 1 T	🟦 3 T	1 T	
Start 2					
Start 2 with obstacle	🟦 3 T			🟦 4 T	🟦 3 T
C=collision, T=timed out, Blue color=path passes obstacle					

Table B.5 Number of failed attempts and cause for each start-goal pair, using TEB as local planner. A box represents five attempts.

Number failed attempts and cause, NN-LSTM					
	Goal 1	Goal 2	Goal 3	Goal 4	Goal 5
Start 1				3 C	
Start 1 with obstacle		🟦	🟦		
Start 2		1 T, 1 C			2 T
Start 2 with obstacle	🟦 1 C	1 T, 1 C		🟦 3 C	🟦 1 C
C=collision, T=timed out, Blue color=path passes obstacle					

Table B.6 Number of failed attempts and cause for each start-goal pair, using NN-LSTM as local planner. A box represents five attempts.

C

Trajectories

Below figures display the successful trajectories from the testing in the validation environment using TEB and NN-LSTM, corresponding to the results in Table B.5 and B.6. The failed trajectories due to collision or timing out have been omitted from the plots.

C.1 Validation environment: Start 1 without obstacle

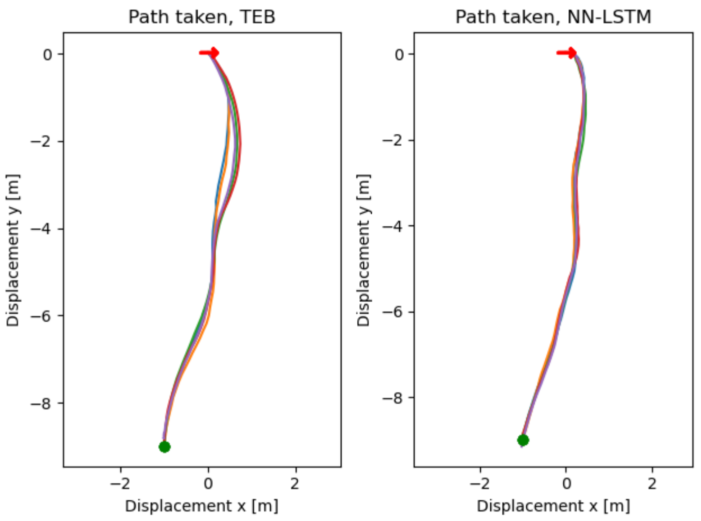


Figure C.1 The successful trajectories during testing for the local planners TEB and NN-LSTM in the validation environment. From Start 1 (red arrow) to Goal 1 (green circle), see Fig. 3.16.

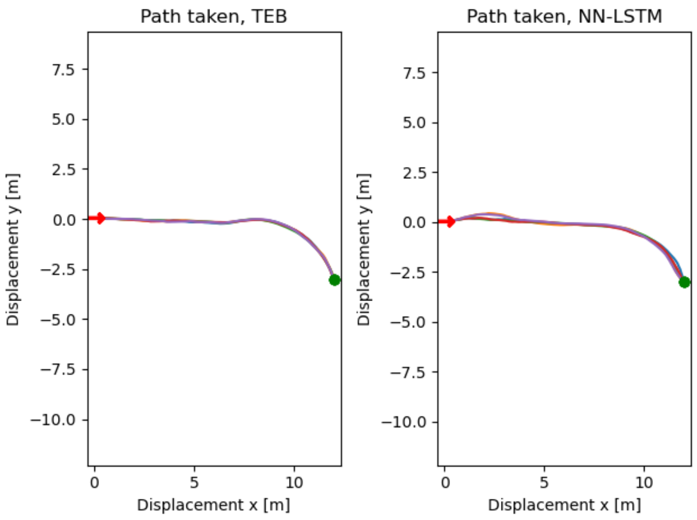


Figure C.2 The successful trajectories during testing for the local planners TEB and NN-LSTM in the validation environment. From Start 1 (red arrow) to Goal 2 (green circle), see Fig. 3.16.

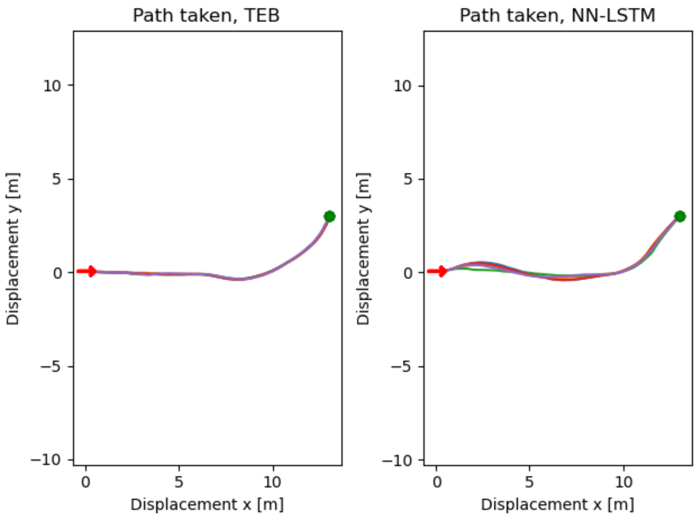


Figure C.3 The successful trajectories during testing for the local planners TEB and NN-LSTM in the validation environment. From Start 1 (red arrow) to Goal 3 (green circle), see Fig. 3.16.

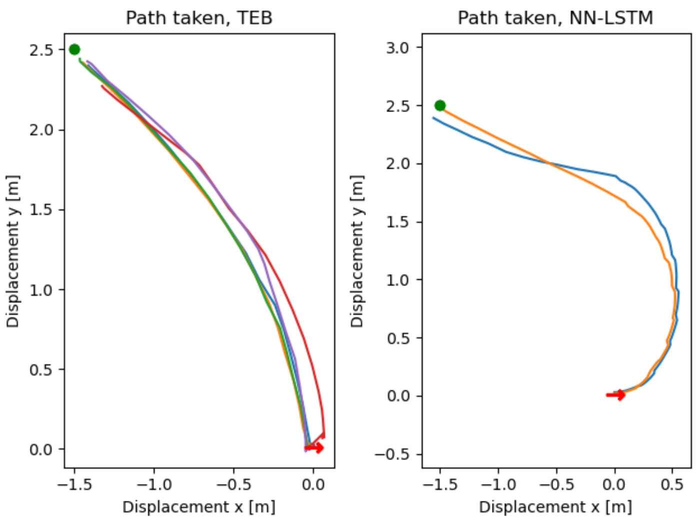


Figure C.4 The successful trajectories during testing for the local planners TEB and NN-LSTM in the validation environment. From Start 1 (red arrow) to Goal 4 (green circle), see Fig. 3.16.

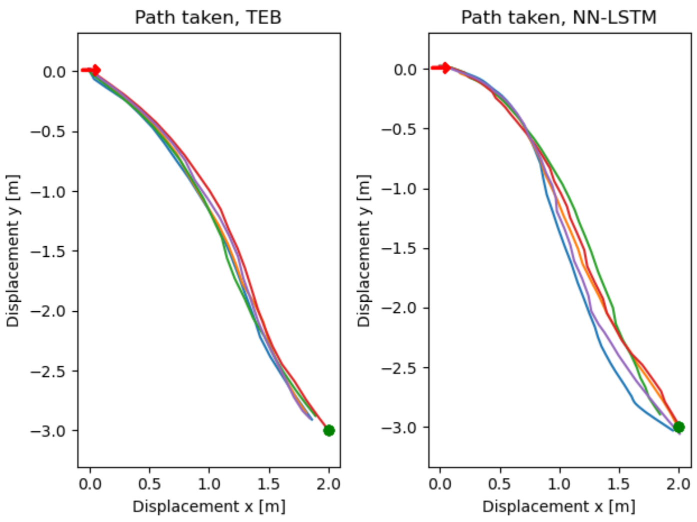


Figure C.5 The successful trajectories during testing for the local planners TEB and NN-LSTM in the validation environment. From Start 1 (red arrow) to Goal 5 (green circle), see Fig. 3.16.

C.2 Validation environment: Start 2 without obstacle

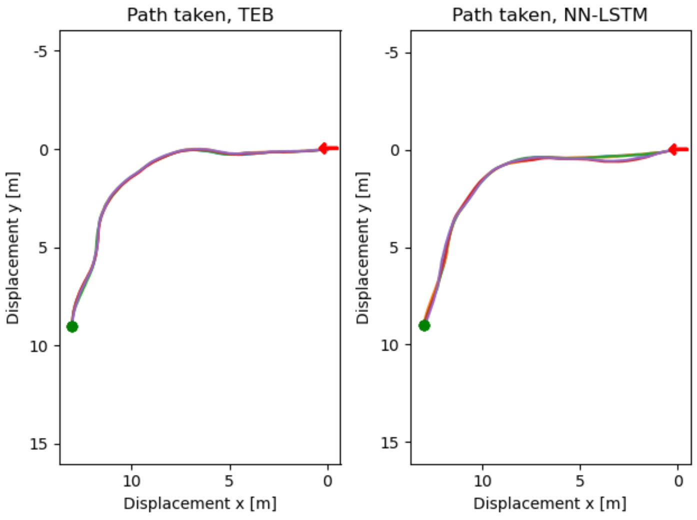


Figure C.6 The successful trajectories during testing for the local planners TEB and NN-LSTM in the validation environment. From Start 2 (red arrow) to Goal 1 (green circle), see Fig. 3.16.

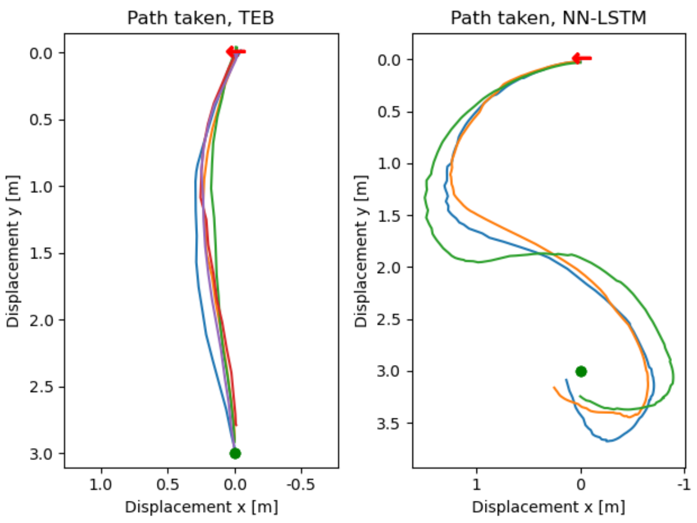


Figure C.7 The successful trajectories during testing for the local planners TEB and NN-LSTM in the validation environment. From Start 2 (red arrow) to Goal 2 (green circle), see Fig. 3.16.

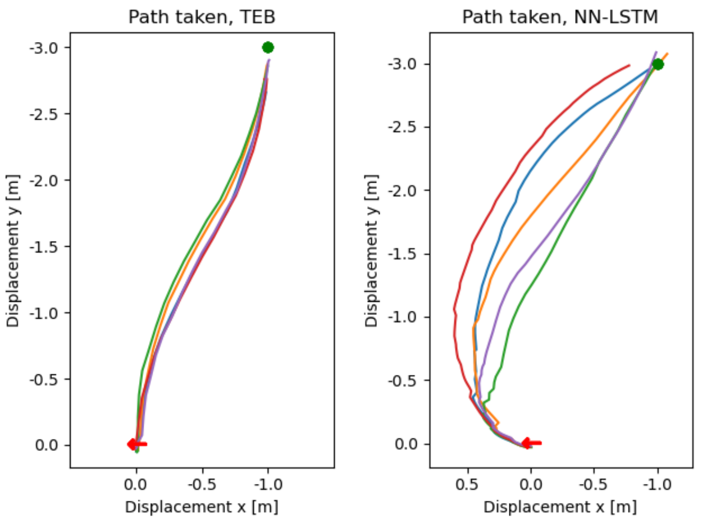


Figure C.8 The successful trajectories during testing for the local planners TEB and NN-LSTM in the validation environment. From Start 2 (red arrow) to Goal 3 (green circle), see Fig. 3.16.

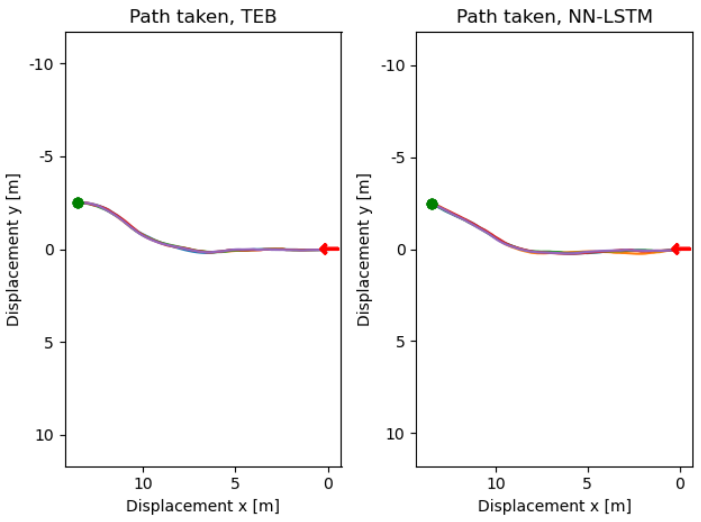


Figure C.9 The successful trajectories during testing for the local planners TEB and NN-LSTM in the validation environment. From Start 2 (red arrow) to Goal 4 (green circle), see Fig. 3.16.

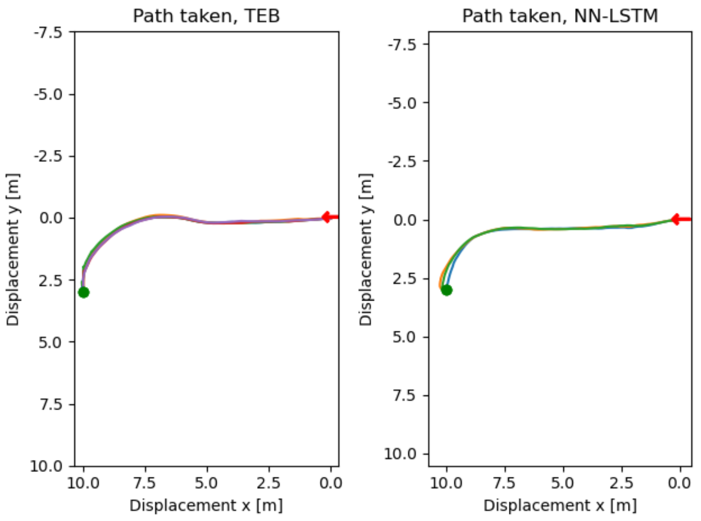


Figure C.10 The successful trajectories during testing for the local planners TEB and NN-LSTM in the validation environment. From Start 2 (red arrow) to Goal 5 (green circle), see Fig. 3.16.

Lund University **Department of Automatic Control** **Box 118** **SE-221 00 Lund Sweden**	*Document name* MASTER'S THESIS *Date of issue* June 2023 *Document Number* TFRT-6210
Author(s) Johan Henningsson	*Supervisor* Fredrik Bissmarck, Swedish Defence Research Agency, Sweden Jonas Nordlöf, Swedish Defence Research Agency, Sweden Anders Rantzer, Dept. of Automatic Control, Lund University, Sweden Bo Bernhardsson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle

Local Planning for Unmanned Ground Vehicles using Imitation Learning

Abstract

Mobile robotics is an expanding field worldwide leading to the need for advanced path-planning algorithms that can traverse various environments. Current state-ofthe- art path-planning algorithms used at the Swedish Defence Research Agency, FOI, tend to be inflexible and parameter dependent. The parameters might need to be tuned for each new environment, which is a very labor-intensive process.

This thesis investigates the possibility to replace computationally heavy pathplanning algorithms with neural networks using Imitation Learning. Neural networks with and without Long Short-Term Memory (LSTM) layers were trained and evaluated. The network without LSTM failed to capture the temporal dependency of the input data, which lead to poor performance. Using LSTM layers performed close to the imitated algorithm in the training environment and in certain situations, the trained neural network outperformed the algorithm by a big margin. In conclusion, neural networks are, after training, able to replace path-planning algorithms and in certain scenarios, the network outperforms the algorithm. Further work is needed to get a robust local planner with a neural network as a base.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title 0280-5316		*ISBN*
Language English	*Number of pages* 1-59	*Recipient's notes*
Security classification		

http://www.control.lth.se/publications/