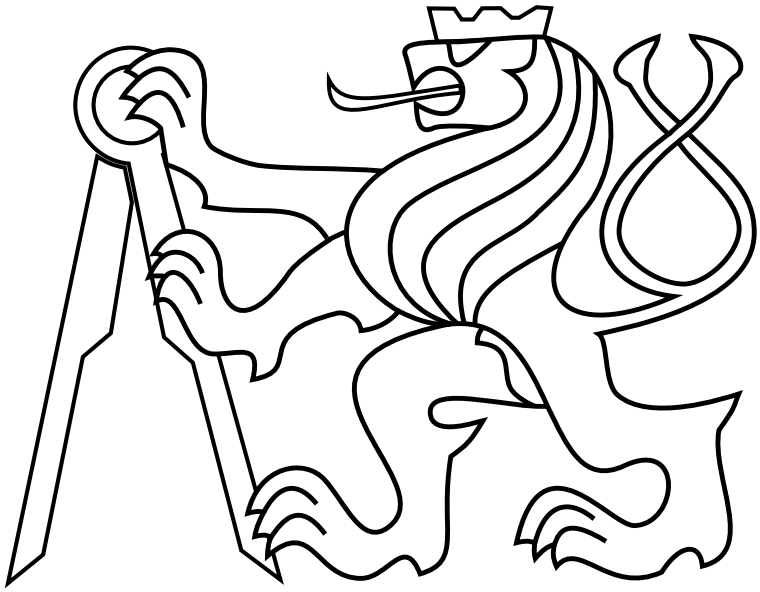
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

BACHELOR'S THESIS



Maxim Simon

Reinforcement Learning for Extended Competencies in Visual Teach and Repeat Navigation

Department of Control Engineering

Thesis supervisor: **Ing. Zdeněk Rozsypálek**

FAKULTA ELEKTROTECHNICKÁ
FACULTY OF ELECTRICAL ENGINEERING
Technická 2
166 27 Praha 6

DECLARATION

I, the undersigned

Student's surname, given name(s): Simon Maxim
Personal number: 510127
Programme name: Cybernetics and Robotics

declare that I have elaborated the bachelor's thesis entitled

Reinforcement Learning for Extended Competencies in Visual Teach and Repeat Navigation

independently, and have cited all information sources used in accordance with the Methodological Instruction on the Observance of Ethical Principles in the Preparation of University Theses and with the Framework Rules for the Use of Artificial Intelligence at CTU for Academic and Pedagogical Purposes in Bachelor's and Continuing Master's Programmes.

I declare that I used artificial intelligence tools during the preparation and writing of this thesis. I verified the generated content. I hereby confirm that I am aware of the fact that I am fully responsible for the contents of the thesis.

In Prague on 23.05.2025

Maxim Simon
.....
student's signature

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Simon Maxim** Personal ID number: **510127**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Control Engineering**

Study program: **Cybernetics and Robotics**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Reinforcement Learning for Extended Competencies in Visual Teach and Repeat Navigation

Bachelor’s thesis title in Czech:

Posilované učení pro rozšíření kompetencí ve vizuální navigaci

Guidelines:

1) Get acquainted with latest methods in reinforcement learning and visual teach and repeat navigation.
2) Use reinforcement learning and simulated environment to train controller for visual teach and repeat navigation.
3) Implement simulated training scheme for collision avoidance in visual teach and repeat navigation.
4) Train a controller with extended competencies.
5) Conduct an experiment with real-world robot to test the viability of trained model

Bibliography / sources:

[1] Schulman et. al., Proximal Policy Optimization, 2017
[2] Rozsypálek et. al., Contrastive Learning for Image Registration in Visual Teach and Repeat Navigation, MDPI Sensors 2022 vol. 22

Acknowledgements

I would like to express my appreciation to Ing. Zdeněk Rozsypálek, from whom I have learned much and hope to continue learning more. Furthermore, additionally and moreover I thank my friends Bc. Anita Hájková and Jáchym Jakl for their company during the further presented work. Lastly, I am grateful to my dad Vít Simon for evening conversations, which provided useful ideas, and for first introducing me to programming.

Abstract

Navigation is the basic requirement for any robot autonomously roaming an area. Oftentimes, the structure of the space is previously known, relieving the need for exploration performed by the robot. Visual teach and repeat (vt&r) navigation relies on a human operator, or other third party, teleoperating the robot to 'teach' it a path. The robot records sensory data during this teach phase and is then capable of autonomously traversing the path. However, any elements changing between teach and repeat phases present a problem for vt&r navigations as they commonly repeat the saved trajectory without any additional autonomy to tackle unforeseen events. In this thesis, I train a neural network acting as a control policy of a vt&r system. The use of reinforcement learning in the development of the control policy allows for extending the capabilities of the resulting vt&r method by expanding the training setup. The additional functionalities could tackle various unexpected scenarios arising from the dynamic nature of environments encountered in a real-world deployment. First, I train a control policy capable of autonomously traversing apriori known paths. Second, I enhance the reinforcement learning scheme to train a control policy capable of avoiding obstacles as it repeats a taught path. Both resulting vt&r navigation methods are experimentally evaluated. This thesis provides a detailed description of the vt&r navigation system and the reinforcement learning setup used for its development. Moreover, the advantages of using reinforcement learning for the development of a vt&r navigation, and possible further extending of its capabilities are discussed.

Keywords: visual teach and repeat, object avoidance, reinforcement learning, sim-to-real

Abstrakt

Navigace je jednou ze základních funkcí robota, který je schopen autonomního pohybu po oblasti. Často je struktura tohoto prostoru známa předem a není potřeba, aby ji robot autonomně prozkoumával. Vizuální teach and repeat (vt&r) navigace spoléhá na člověka, nebo jinou třetí stranu, který robota ručním ovládáním provede cestou. Robot během této 'teach' fáze nahrává data ze senzorů a později je schopen cestu autonomně projet—'repeat' fáze. Avšak změny v prostředí mezi teach a repeat fázemi představují problém pro vt&r navigace, jelikož opakování cesty je obyčejně prováděno bez dalších autonomních schopností řešit nepředvídatelné události. V této práci trénuji neuronovou síť, aby vykonávala kontrolní řízení vt&r systému. Využití strojového učení ve vývoji řídící funkce dovoluje, vylepšením trénovacího rozhraní, přidání kompetencí výsledné vt&r metodě. Přidané funkce mohou řešit neočekávané situace způsobené dynamickou povahou prostředí. Nejdříve vytrénuji řídící funkci schopnou provést robota naučenou cestou. Poté rozšířím schéma strojového učení tak, aby vytrénovalo řídící funkci schopnou vyhýbat se překážkám během autonomního opakování trajektorie. Obě výsledné vt&r navigace jsou experimentálně vyhodnoceny. V této práci poskytuji detailní popis vt&r navigačního systému a struktury strojového učení, které bylo použito pro jeho vývoj. Dále diskutuji výhody použití strojového učení pro vývoj vt&r navigační metody a možné budoucí rozšíření jejích kompetencí.

Klíčová slova: vizuální teach and repeat, vyhýbání se objektům, strojové učení, sim-to-real

Contents

List of Figures

List of Abbreviations

Abbreviation	Meaning
6DoF	6 degrees of freedom
3DoF	3 degrees of freedom
vt&r	Visual teach and repeat
NNVTR	Neural network visual teach and repeat
CAVTR	Collision avoidance visual teach and repeat
Bearnav	Bearing (only) navigation
RL	Reinforcement learning
PPO	Proximal policy optimization
CNN	Convolutional neural network
lidar	Light detection and ranging
SLAM	Simultaneous localisation and mapping
ORB	Oriented FAST and rotated BRIEF
FAST	Features from accelerated segment test
BRIEF	Binary robust independent elementary features
SURF	Speeded up robust features
ROS	Robot operating system
GPS	Global positioning system
sim-to-real	Simulation to real world

List of Appendices

1. Zdeněk Rozsypálek, Tomáš Rouček, Maxim Simon, Tomáš Musil, Martin Saska, and Tomáš Krajník: Learning control policy for visual teach and repeat

Chapter 1

Introduction

As first described in [1], robots exist to ease the life of humans by performing manual labour. Today their purpose is expanded by robots performing tasks both intellectually and physically difficult or too dangerous for humans. Stationary robots, e.g., in factory lines, have a limited amount of functions as their space of reach is confined. Most human endeavours take place in several points in the world, often spaced far apart. Matter and information need to be transferred between the points. For robots to perform at least all useful labours they need the ability to traverse the world, much like humans do.

In mobile robotics, non-stationary robots move in space. To do so autonomously, they need information about their surroundings. Otherwise, their movements in the environment would be random. To traverse from point A to point B, either the absolute positions of the points or the relative position of A to B must be known. Additionally, the robot needs information about its position—absolute or relative—within the space to know in which direction it should move and to recognize if it has reached its goal point B.

Various navigation systems exist to tackle the problem. Complete and absolute navigation consists of having a form of a map of the space in question and continuous localisation of the robot's position within said map. Full localisation determines 6DoF of the robot—position (x, y, z) and rotation (θ, ϕ, ψ). To move between points, which can now be found in the map, path planning is used to find a traversable trajectory between the points. For example to avoid a wall.

This ideal exhaustive navigation system is complicated to create, requiring complex algorithms, a lot of sensory data to create the map, and significant computational resources. However, the navigation can be simplified by applying restraints to the problem for which a system is developed or by not determining redundant information.

For ground vehicles deployed in an environment with low elevation gain, e.g., road travel, the traversed space can be locally approximated as a plane, instantly losing 3DoF: z position, roll θ and pitch ϕ. Further, the remaining 3DoF (x, y, θ) can be determined about an absolute coordinate frame or relative to objects or positions of interest. The robot can track its odometry readings and keep track of its current position relative to its starting position; intuitively, it leaves a path of breadcrumbs. Similar approaches that do not perform explicit localisation can be preferred in less complex, straightforward navigation problems due to their lower sensory and computational power requirements.

A map of the environment can not be created without first observing it. The robot acquires a map either by autonomous exploration, a technique used by SLAM algorithms [2], by human teleoperating the robot to gather information about the environment—non-autonomous exploration, or by receiving the map from a third party. For tasks that do not require traversal between arbitrary points but rather between predefined points A, B, C, etc, a global map is not needed, only the trajectories between the desired points. Teach and repeat navigation systems rely on third party teleoperating the robot across a trajectory, relieving the need for path planning to be included in the navigation system. The robot saves information about the traversal and can later repeat it autonomously.

Visual teach and repeat navigation (vt&r) relies on images taken by a camera to achieve this. As shown in [3, 4], ground robot can successfully traverse non-straight apriori known paths by repeating

movements recorded during teach phase (for example by replaying velocity commands) and using image comparison solely to adjust heading of the robot. There exist several vt&r systems employing these sensor data in varying ways, and some employing additional sensors, e.g., depth sensors. In general, the path is traversed by employing odometry data collected during the teach run, and adjusting its heading to steer toward the desired path based on the difference in alignment of current camera view and saved pictures from teach run. Based on the difference in their alignment, a correction to steer toward the desired path is made. There exist systems that rely solely on visual data [5, 6]. However, NNVTR, the system that is the subject of this thesis, employs both odometry data and camera images. An overview of the current state of various vt&r systems is in section 2.

Comparison of two pictures of the same place is more complex the less the pictures resemble each other. If something is only in one picture, it is impossible to compare it to anything, and therefore, no offset can be calculated. This dissimilarity is caused by dynamic factors in the environments through which the robot navigates. These can include weather conditions—snowfall, rain; changes in lighting—clouds, different times of day; moving elements—walking people, cars. In summary, image comparison is difficult in dynamic environments due to elements that appear only in one of the two considered images.

Feature detection and matching is the general approach for calculating the offset. Oftentimes, the current camera picture is additionally compared with multiple ones taken during teach phase, and the one that is judged to correspond with the current camera view the most, is used for correction. The robot presumes it is at the point along the path where this picture was taken and correction to longitudinal—forward, backward direction—offset is applied. Various approaches, described in section 2.2, have been taken to make the image matching more robust to environmental change.

The vt&r system used in this thesis employs fully connected convolutional neural network (CNN) for this task. The camera pictures are first processed in various ways and the error in their alignment is calculated by comparing the resulting representations. More precisely probabilities of all possible offsets are calculated. This high dimensionality vector of probabilities together with odometry data serve as an input into another neural network which acts as the control policy steering the robot. Both odometry data and the afformentioned vector of probabilities, which represents the information about the robot's offset, serve as its input and the output is vector $[\omega, \Delta d]$, where ω is adjustment to angular velocity—it steers the robot towards the correct path—and Δd is adjustment to estimated traveled distance—it corrects the robot's estimation of how far along the path it is located.

For the work of this thesis, the CNN visual model is deployed already pretrained as described in [7, 8, 9]. However, the control policy model is trained inside HARDNAV simulation [10] using an implementation of proximal policy algorithm. The entire system is publicly available at [11]. HARDNAV is an open-source simulation designed for testing of mobile robotics software. It is implemented in Unity game engine. Before training the control policy, the HARDNAV simulation is altered to tailor it more specifically to a vt&r navigation development. The system learns on a simulated Jackal robotic platform with a ROS Camera.

Reinforcement learning is used for the development of the control policy, making it more moldable than policies defined by generic algorithms. This suggests that in addition to the model being able to traverse a previously taught path, other functions could be added by exposing the system to particular scenarios and designing a suitable reward system. As long as the problems presented are solvable by employing only the available sensor data and actions it should be possible to train a vt&r system with additional functionalities.

One of the fundamental problems of navigation, or rather most of mobile robotics, is not to crash into anything. Vt&r scenarios usually presume obstacle-free trajectory. In dynamic environments, however, an unexpected obstacle can get into the path between teach and repeat phases. I train the model to avoid previously unseen solid obstacles that are in the way.

To avoid an obstacle the system needs to be able to detect it, which is a difficult and computationaly expensive problem if only data from one monocular camera are used. Especially when an explicit localisation within the environment is not performed, only within the traversed path. A depth sensor is much more suitable for object detection as it provides information about the relative location of the obstacle to the robot and furthemore it does not rely on seeing the obstacle beforehand, which

is an important aspect when dealing with dynamic environments. For these reasons, a 2D lidar is mounted on the robot for autonomous traversals.

The model is trained in the HARDNAV simulation and the resulting control policy agent is deployed on a real-world robotic platform Jackal in a local park. To evaluate the vt&r system number of varying paths are recorded and several different obstacles are placed in the trajectories before the autonomous repeat. Additionally, an initial offset is introduced in order to examine the convergence capabilities of the navigation. Results of the conducted experiments are presented in section 6.

In this thesis I go over the structure of our previous work [12] and additionally present a newly trained model which is capable of avoiding obstacles introduced into the path after the teach traverse.

The thesis is structured as follows. First, an overview of state-of-the-art in visual teach and repeat is provided in chapter 2. Definition of the reinforcement learning problem as employed in the control policy development is given in chapter 3. Afterwards, the execution of the main tasks of this thesis is described in the method chapter 4, which is divided into three sections. Section 4.1 describes in detail the vt&r and RL system as presented in Rozsypalek et al.'s work. The work is then reproduced in section 4.2 by training NNVTR control policy capable of autonomous traversal of a taught path. Final section 4.3 in the method chapter describes the development of a new CAVTR control policy with an additional capability of collision avoidance while autonomously traversing a path. The two trained control policies of the vt&r system are deployed outdoors on a Jackal platform for experimental evaluation and comparison against the Bearnav method [4]. The chapter 5 describes the experimental setup. Results are then presented and discussed in chapter 6. As CAVTR failed to avoid obstacles placed in its path a comprehensive discussion of future work on the system is in chapter 7. The results and contributions of this thesis are summarized in the conclusion 8.

Chapter 2

Related Work

In this section, I go over current vt&r systems and differences in their approach to both learning a path and its consequent autonomous repetition. Then I discuss developed solutions to frequent problems that arise in practicle deploynments of a vt&r navigation and further the usage of reinforcement learning in solving said problems.

2.1 Visual Teach and Repeat State-of-the-Art

In [4] Krajník et al. introduced a vt&r system dubbed Bearnav, which employs solely odometry and monocular camera data to repeat apriori known path. It does this without building a map and therefore without performing explicit localisation. During teach phase the robot saves forward and angular velocities and indexes them by distance travelled so far. Additionally, it collects camera pictures every 0.2 meters and indexes them in the same way. During repeat phase the taught path is traversed autonomously by dead reckoning—repeating saved forward and angular velocities. Due to inaccuracies in odometry sensors and errors caused by wheel slipping, irregularities in terrain etc, the robot would drift away from the desired trajectory. However, this deviation is corrected using the pictures taken. As the robot traverses the taught path it takes pictures at the same distance intervals and matches them to the corresponding images from the teach phase. The difference in horizontal alignment of the images is calculated and used for adjusting the robot's angular velocity—it's heading is modified to steer the robot towards the correct path. Distance traveled—i.e. position in the trajectory—is determined solely by odometry readings, relying on the fact that odometry sensors usually display larger error in rotation than forward traversal.

A similar approach was introduced before by Birchfield et al. in [13, 14], but their system uses both odometry and camera data to build a map representation of the traversed trajectory. During the autonomous repetition, it additionally employs image matching to correct the robot's estimation of its position.

Barfoot et al. developed a map-based vt&r system designed for long-range navigation [5, 15]. They do not collect odometry data and only use stereo camera for both building the map and then repeating it. During teaching a 3 dimensional map is built, which is then projected on a 2 dimensional plane. It is shown that this approach provides a map locally consistent enough for successful autonomous traversal of paths several kilometers long.

The use of only a camera for the entire navigation task is also done in [6], where pictures are taken during teach phase by a monocular camera. These images are fed to a convolutional neural network, which determines the robot's horizontal offset. Further, a particle filter is employed to recognize where along the path the robot is located. In contrast to this mapless vt&r system, its authors presented ORB-SLAM2 [16] based system [17], which builds a map during teach phase and simultaneously localizes the robot in it.

Explicit correction of lateral error along the path is performed in bio-inspired system [18].

This vt&r system's general approach resembles afformentioned Bearnav in that it uses odometry and monocular camera data only. However instead of repeating velocity commands it builds a map by saving odometry positions and corresponding images. To repeat the trajectory it tries to follow saved positions and simultaneously corrects its heading based on camera data. To make the image matching more robust to environmental changes and poor lighting conditions the camera pictures are first processed and downscaled. Furthermore the current image taken during repeat phase is not matched only with the teach image at corresponding position, but rather with a small range of images around it. Image with the strongest correlation is deemed to be closest to the desired robot position and the next odometry pose goal is moved further or closer, causing the robot to alter its forward velocity to correct this longitudal error.

Trained fully convolutional neural network presented in [7] and [8] creates dense representations from pictures taken by an onboard camera and then calculates horizontal displacement between them. This vision module was further enhanced by a particle filter [9] and deployed with an altered version of the aforementioned Bearnav vt&r. The particle filter is used to correct the robot's estimate of its position along the path. Moreover it takes into account the relation between consecutive images and derives from them information about shape of the path. The presented vision module comprising of both the CNN network and the particle filter is part of a vt&r system that is the subject of this thesis and its detailed description is in section 4.

2.2 Challenges of Dynamic Environments

Most of the afformentioned systems focus on the core problem of a teach and repeat navigation, autonomous traversal of a taught path. In practical deployment, however, a navigation system encounters a number of problems. Repeating a path is easiest when it does not change after it is taught to the robot. This of course does not happen in real environments, which are either more or less dynamic.

Changes in appearance present an issue for a vt&r navigation as it heavily relies on comparing the current camera view with camera view from teach phase. Basic calculation of the difference in horizontal alignment of images is done by first extracting features from the 2 images using techniques like ORB [19] or SURF [20], and subsequently matching corresponding pairs of features and calculating their horizontal displacement. This approach is used for example by most Bearnav-based systems [3, 4, 21]. Bio-inspired vt&r system [18] and its enhanced version [22], which uses the same vision module but a more precise control module, preprocess images before deploying feature detection algorithms to make the relevant data more robust to environmental or other changes. Often used is also the deployment of a CNN network directly on camera pictures to calculate their offset [6, 8] or for more robust feature detection [23]. Another approach is to use additional sensors, which are less susceptible to appearance change than just one monocular camera. These usually involve depth sensors like stereo camera [5], lidar and radar [24] or a laser scanner [25]. Finally, the vt&r system presented in [26] tackled seasonal appearance change in traversed environments by updating its map during autonomous traverses of the taught path.

Apart from visual appearance changes, solid objects often change their position in dynamic environments, such as people or cars. As the robot generaly traverses the path with non-zero error, it can additionally encounter obstacles that are positioned close to the taught trajectory but during repeat they are directly in the way. Vt&r systems usually presume that if a trajectory was traversable during teach phase, it will be traversable during repeat phase and therefore fail if there is something in the way forcing the robot to deviate from the path.

The issue is addressed in [27], where the robot is equipped with 2 lidar sensors for creating an elevation map of its surroundings and the vt&r system uses a locally reactive controller to plan a path around an encountered obstacle. The controller is aimed at safe autonomous traversal. Lidar for obstacle detection is also used in [28], where a novel controller for their avoidance is introduced. Its aim is to circumvent a detected obstacle while minimally deviating from the traversed trajectory.

In this thesis, I present a vt&r system, with a neural network trained by a PPO scheme [29]

acting as a control policy capable of repeating a taught path while avoiding obstacles introduced after teach phase. This work builds on the previous work of Rozsypalek et al. [12].

Chapter 3

General Reinforcement Learning Overview

Reinforcement learning of policies can be divided into two main categories, value-based and policy-based. Value-based algorithms, such as Q-learning [30], are generally used for training deterministic policies $\pi(a|s)$—output an action a based on the given state s. Policy-based algorithms, such as Proximal Policy Optimization (PPO) [29], which is used in this thesis, create stochastic policies $\pi(a|s,\theta)$—output an action a based on the given state s and the parameter θ.

Deterministic policies are well suited for discrete state spaces with low dimensionality. The policy $\pi(a|s)$ gives identical actions for the same states. Mind that both a and s can be vectors and with their rising dimensionality the number of connections between states and actions the policy needs to remember, and more importantly learn, grows. In the vt&r problem of this thesis the action is a 2 dimensional vector, but the state is the observation passed to the neural network and its dimension is in the range of thousands. Moreover, as new capabilities can be added, both the observation and action space may grow, making a value-based reinforcement learning not suitable for training of the control policy neural network.

Stochastic control policies [31] on the other hand are designed to handle continous, and therefore often high-dimensionality, spaces. The policy $\pi(a|s,\theta)$ is a probabilistic distribution of action space over the given state. In contrast to value-based reinforcement learning, which during training chooses the action with the highest expected reward, policy-based learning samples actions from this distribution. providing an exploration-exploitation balance. The θ parameter, e.g., weights of a neural network, defines the probabilistic distribution of action space. It is the value the policy refines during training.

The expected reward is determined by various factors, depending on the implementation of the RL scheme, however, it always includes a reward function. The reward function defines the desired behaviour towards which the policy is trained. The actions the system learns to undertake are chosen based on the reward that is expected to be received in the future. Intuitively it tries to act in a way that maximizes the reward function [32].

As the reward function is defined by a human, but the system learns by itself on the deployed dataset, an RL scheme takes away most of the *how* off of the method's developer and rather relies heavily on an accurate definition of *what*. The vt&r problem considered in this thesis consists of a substantial amount of rather complicated inputs comprising depth sensory data, odometry data and several histograms encoding various offsets, as described in the method chapter 4. All these inputs, however, do not contain an unreasonable amount of numerical data. Reinforcement learning scheme is able to learn how to efficiently employ these data for a vt&r problem while hand-crafting an algorithm that takes advantage of all the available information would be more complex to design than creating the RL scheme.

Performing an action and receiving the appropriate reward is called a training step. Episode is an attempt of one task, in our case an autonomous traversal of a path. After one epoch, defined in

our system as 1000 training steps, the parameter θ of the policy is adjusted based on the gathered rewards.

For training of the control policies in this thesis, a PPO algorithm presented in [33] is used. A PPO algorithm is a version of policy-based reinforcement learning. The function of expected reward is clipped at a constant value if the adjustment of the policy is too large. As the expected reward no longer grows, the system loses the incentive to further change the policy. With problems comprising of a large number of possible scenarios, or more precisely states, it can happen that an experience gathered during an episode does not reflect on the problem as a whole, causing the system to take too large a step in the newly learned direction, worsening the overall performance of the control policy. The PPO clipping design is for the purpose of taking small incremental learning steps for more stable development of the policy.

All of the concepts mentioned in this section are presented and described in detail in [34].

Chapter 4

Method

The subject of this thesis is threefold. First, I train the control policy to work as part of a vt&r system, effectively reproducing the yet unpublished work of Rozsypalek et al [12]. Afterwards I modify the reinforcement learning scheme to train a new enhanced control policy, which is capable of avoiding obstacles during the autonomous traversal of a path. Finally, the new vt&r system is deployed on a 4-wheeled robotic platform and experimentally evaluated.

4.1 Specifications of the VT&R System

In this section, I provide a comprehensive description of the vt&r system developed by Rozsypalek et al. It is based on Bearnav [4], description of which is provided in section 2. This basic vt&r paradigm is enhanced with two neural networks, one acting as a visual module, the second as a control policy. An overview of both of these modules is presented in sections 4.1.1, 4.1.2 respectively. The control policy is trained inside a HARDNAV simulation [10], which is described in 4.1.3

4.1.1 Visual Module

The visual module is responsible for processing pictures taken by an onboard camera and providing information to the control module about the robot's displacement from the taught path.

Feature matching algorithms focus on detecting edges to provide a sparse representation of the examined image. Representations of two images can be compared by matching edges appearing in both of the pictures. This approach is robust to change of position from which an image of a place is taken, due to the objects appearing in both pictures providing a sufficient number of matchable edges. However, if the environment changes in appearance even though the structure remains the same, e.g., leafs falling off a tree, feature matching experiences problems as the detected edges in one picture are not in the second matched picture.

The visual module of NNVTR is trained to provide a dense representation of an image, ensuring the relations between closely clustered parts of the picture can be exploited. Therefore, it does not match sparsely distributed appearance features but rather structural landmarks of the environment. This approach of image matching is less affected by dynamic factors, because if any parts of the environments structure change, they will simply be omitted. The matching focuses on stable, constant structural features.

The Siamese neural network [7, 8, 9], which acts as the visual module, is pre-trained on Nordland [35] and EU long-term [36] datasets. These datasets provide views taken from similar positons and angles but a long time apart providing the neccassry appearance change for development of neural network focusing on strctural definition of an environment.

The pictures from the datasets were indexed by GPS coordinates to match images of the same view taken at different times. Training consisted of feeding a 56-pixel cutout of an image into the backbone of the CNN, formed based on the architecture presented in [37], which outputs a 7-bin wide

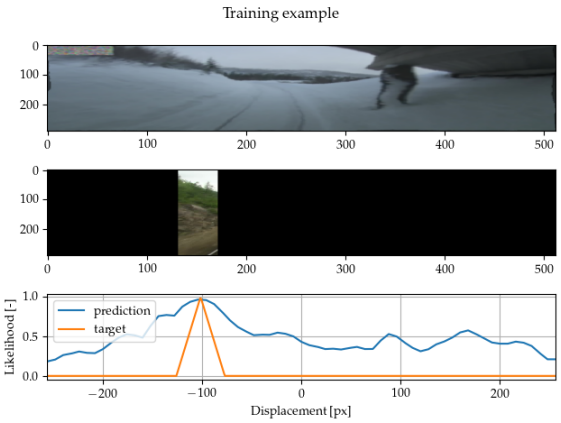


Figure 4.1: An example from the training of the Siamese neural network. First, it determines the likelihood of structural similarity (blue line). It uses that information to determine the position of the cutout in the compared image (orange line). This offset determination was performed during training of the Siamese network but when deployed as visual module of NNVTR only the likelihood is calculated. Notice the appearance change caused by the different seasons in which pictures were taken. Source: [8]

representation of the cutout. A corresponding picture taken at different time is then given to the neural network, which based on the representation created by the backbone tries to determine where in the corresponding image the cutout is located. One training example is shown in Figure 4.1. The representations are created in such a way, that their cross-correlation provides information about structural similarity of the teach and repeat images, based on which an offset of the robot can be deduced. The advantage of handling representations rather than raw camera data is increased robustness to appearance change of the traversed environment, e.g., snow cover.

During deployment of the trained neural network, its task is not to explicitly calculate the horizontal offset of two images but rather to determine probabilities of two images being horizontally offset by a certain value, further denoted Δs as in shift. The original pictures have size $512 \times 384 \times 3$ and their resulting representations are of dimension $64 \times 6 \times 16$. One picture is always the latest image taken by an onboard camera during the autonomous traversal. Against which images this view is compared is determined by the robot's estimate of distance travelled so far, which will further be marked d. The robot assumes it is located at distance d, measured along the path's trajectory, from the starting position. Therefore, the image with its distance index closest to d is assumed to be most likely to correspond with the current view and thus provide the most accurate information about the robot's horizontal offset from the traversed path.

Calculation, within the CNN, of likelihood that images I_a and I_b are horizontally offset by Δs can be denoted as follows:

$$\mathcal{L}(I_b, I_a|\Delta s) = r_{\Delta s}(\mathbf{R}_a) \star \mathbf{R}_b.$$

$\mathbf{R}_a$ and $\mathbf{R}_b$ are representations of the images created by the backbone of the visual module CNN. $r_{\Delta s}$ denotes the function applying roll padding in the horizontal dimension by the value Δs to one of the representations. This shifted representation $\mathbf{R}_a$ is then cross-correlated with the other representation $\mathbf{R}_b$ to determine the probability of the original images I_a and I_b being horizontally offset by value Δs. Note that the images do not have to be of the same place, e.g., when the robot is lost. In such a case, an ideal agent would determine the probability to be zero.

This process is repeated for all possible values of Δs and the result is a vector of probabilities of all possible horizontal offsets between the compared images. Such vector is further denoted as $\mathbf{h}_{I_b}^{I_a}$. The number of possible values of Δs is limited by 50% of the images' width.

Additionally, to the teach image closest to the distance index d, several following and preceding images are compared to provide information about possible inaccuracy in the value d. If, for example,

the immediately next image has higher likelihoods for the various offsets from the repeat image, the robot is probably located at a distance closer to that image's distance index than at d. The number of pictures in each direction that are considered is an adjustable parameter. NNVTR uses two in each direction because it provides a good trade-off between the information provided and computation speed. The resulting vectors of probabilities are concatenated into one vector:

$$\mathbf{z}_L = [\mathbf{h}^{I_L}_{I_{i-2}}\ \mathbf{h}^{I_L}_{I_{i-1}}\ \mathbf{h}^{I_L}_{I_i}\ \mathbf{h}^{I_L}_{I_{i+1}}\ \mathbf{h}^{I_L}_{I_{i+2}}],$$

where I_L is the current live image and i is a number index of the image closest to the current value of d.

In addition to comparing camera view during repeat phase to images taken during teach phase, the visual module matches consecutive images recorded during teach phase. By doing so it provides the control module with information about the shape of the path. For example images recorded during a right turn will have a steady right-leaning difference in their horizontal alignment. Knowing this can cause the robot to steer more aggressively, helping it converge to the path faster. Again, two images forward and two backwards are taken into consideration, and the resulting vector of this consecutive image comparison takes the form

$$\mathbf{z}_m = [\mathbf{h}^{I_{i-2}}_{I_{i-1}}\ \mathbf{h}^{I_{i-1}}_{I_i}\ \mathbf{h}^{I_i}_{I_{i+1}}\ \mathbf{h}^{I_{i+1}}_{I_{i+2}}].$$

The output of the visual module is vector $\mathbf{z} = [\mathbf{z}_L\ \mathbf{z}_m]$ and it serves as an input of the control module.

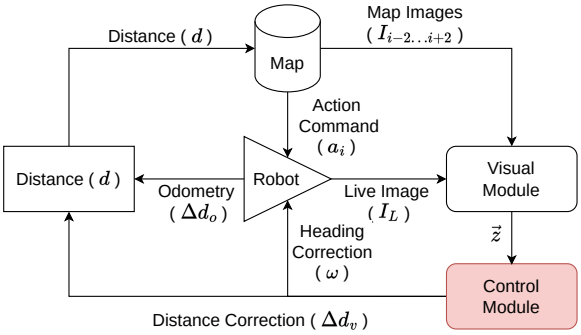


Figure 4.2: Diagram of NNVTR navigation process. Map contains list of actions a_i and map images I_i, where i indexes are determined by distance d. Robot, which contains the core of the vt&r system, gathers live odometry readings Δd_o and images I_L. The live images are passed into Visual Module neural network, which fetches map images $I_{i-2...i+2}$ for comparison. The output vector $\vec{z}$ is passed into Control Module neural network, which outputs a heading correction ω to steer the Robot and distance correction, which combined with odometry readings provides a new value of d based on which the map data is fetched. Robot combines action a_i with heading correction ω to move. Red Control Module square is the trained neural network control policy. Source: [12]

4.1.2 Control Module

The goal of both this thesis and Rozsypalek et al.'s work is to train the control policy. It is composed of a feed-forward neural network which takes the $\mathbf{z}$ vector as its input and generates an output vector $[\omega\ \Delta d]$. Δd is an adjustment made to d to correct the robot's estimation of its position within the traversed trajectory. Based on the newly corrected d an action with the corresponding distance index saved during the teach phase is fetched. The action has form of forward and angular velocity which the robot had at that place of the map during teach traversal. The forward velocity is unchanged, but the angular velocity is adjusted by ω to converge the robot to the path. The entire process of NNVTR employing both the visual and the control module is in Figure 4.2.

The control policy is trained using a proximal policy approximation training scheme. The vt&r system is deployed in a simulation and is given several various maps which it attempts to autonomously repeat. A detailed description of the training process including how rewards are distributed, is in section 4.2.

4.1.3 Simulation

If training the control policy agent in real world on actual robotic platform was logisticaly feasable, it would most likely prove to be far superior to anything trained in a simulation. This is not possible of course as current RL systems improve slowly per amount of data available, meaning training runs require minimally thousands of episodes and millions of experiences [38]. In the case of vt&r navigation, the neural network needs thousands of attempted autonomous traversals to result in a working control policy, which would take weeks or months. Additionally, the robot fails most of the traversals and needs to be positioned back to start of another path. Then crashes would need to be detected before the robot actually crashed into anything and damaged its hardware. All these factors and more, which I will not list, contribute to a large strain on resources.

The training in simulation can run continuously on a computer, no robot is needed only software. The robot can be teleported within the simulation minimizing time when it does not traverse anything and the CNN is not learning. Moreover, simulation time can be faster than real time, thus realizing more training in less time. The goal of the simulation is therefore to simulate the world as closely as possible to train a well-working agent, while shortening the time and other resources, it takes to do so. To do this it is important to keep in mind in what way the vt&r navigation system interacts with the world and ideally simulate only that, which is detectable by the employed sensors.

The trained vt&r system uses only three sensors: odometry, a monocular camera and a lidar. Lidar was added to the system as part of the work presented in this thesis. It's implementation is described in section 4.3.1. The control policy is both in this thesis and in original Rozsypalek et al.'s work trained inside the HARDNAV simulation [10], which is implemented in Unity game engine.

Odometry is easy to simulate as the simulation naturally knows exactly how its elements—including the robot—move. For simulating the real world adequately in the context of camera data, the environments need to be visualised. However, as the visual module exploits the stability of the environment's structure for image matching rather than explicit pixel values, high resolution, i.e. high visual resemblance to the real world, is not strictly needed. The Siamese neural network creates dense representations which are further compressed into likelihoods of structural similarity that are passed to the control module. Therefore the camera view's visual quality needs to be consistent rather than high. HARDNAV does this well and the trade-off between visual appearance, variability of environments and computational speed is adequate. The lidar sensor shoots rays which need to collide with an obstacle to detect it. For this the HARDNAV simulation is well suited as its implemented in a game engine which offers high support for ray interactions on the account of games often containing projectiles such as lasers or bullets.

In the following sections, I provide specifications of the altered HARDNAV simulation used in Rozsypalek et al.'s work. First, a general overview of HARDNAV implementation and its features is provided. Afterwards I describe the environments within the simulation. Then I explain the robot's and the camera's implementations.

HARDNAV Simulation

The HARDNAV simulation [10] was designed specifically for the development and testing of long-term navigation systems for mobile robots. It is implemented in Unity game engine and all its source codes are publicly available. HARDNAV aims to simulate environments spanning several kilometers wide. Being an open-source implementation in a game engine it is relatively easy to modify, which is desirable as the vt&r system can be enhanced with additional functionalities in the future, for which requirements not yet implemented in HARDNAV may be neccassary. Furthermore it allows simple randomization of environmental conditions, e.g., fog, rain, wind and lighting, which is advan-

tageous for the purpouse of creating diverse database on which the neural network will be trained, in order to result in a more robust control policy.

The simulation after alterations made for Rozsypalek et al.'s work comprises of two scenes, even though only one (forest) was used. Both were used only in the work of this thesis. Scenes are worlds created within Unity, which are independent of each other. Each contains its own objects, functions, definitions etc. They are compiled as part of one project and then launched one at a time, allowing switching between natural and structured environments during the training of the control policy.

GameObject is a basic building block of a Unity project. Except for special cases like Assets—a form of a function library within Unity—all GameObjects have a position in the scene, even when they are invisible or uninteractable with. Both the appearance and physics of GameObjects can be defined. In addition to having position, physical shape and appearance a GamObject can contain a C# script, allowing for a programable behaviour. An example of such game object is a white cube (appearance) which is solid for all other GameObjects (physics), placed on the ground (position) and when something touches it it turns red (behaviour). Finally, GameObjects can be nested, for example, a small blue square on a side of the cube.

Each scene is composed of several GameObjects and explaining every one is beyond the scope of this thesis. Notable GameObject, which is in both the forest and the mall scene is VisSky.

VisSky is responsible for the appearance of the sky. It defines the day cycle within the simulation, displaying the sky, stars, sun and clouds and moving them according to the set speed of simulation time. Additionally, it handles natural lighting, including shadows thrown by clouds. Finally, the intensity of rain and the wind speed are determined by this GameObject.

Other GameObjects of significance, that are part of both scenes, are Jackal, which defines how the robot is simulated, and ROS camera, which is part of the Jackal GameObject.

Description of the Environments

As shown in [39] vt&r systems generally perform better in structured environments which offer stable, well-distinguished features in the forms of human-made objects—buildings, windows, pavements etc. Worse performance is observed in natural environments, which offer more dynamic and less recognisable edges. The HARDNAV implementation presented in [10] offers two worlds: forest and sci-fi. In Rozsypalek et al.'s original work, the forest scene is used as it represents a natural environment well. However, instead of the sci-fi world, Roucek, a co-author, implemented a new shopping mall scene to simulate structured environments.

Forest scene's view from the robot's camera is in Figure 4.3. Most of its appearance is handled by Terrain, a GameObject with predefined functions, which allow easily randomized rendition of objects—convinient for simulating nature. It defines the general appearance of the forest scene, including the ground, trees, other vegetation and water surfaces. Other objects scattered through the scene, e.g. a hut, are defined separately.

Trees are added to the scene by a Unity predefined function called 'Paint Trees'. Vegetation added in this way can not interact with rays, which is a necessity for lidar detection. To rectify this I used a publicly available Asset called VegetationSpawner. The description of the process is detailed in section 4.3.1.

Mall scene's view from the robot's camera is in Figure 4.3. As it is a structured environment it is less random than the forest. That is why the Terrain GameObject is not used, but the scene's structure and objects are handled as individual or grouped GameObjects. These include ground, buildings, stairs, walls, shops and glass ceilings.

The forest scene contains only natural light coming from the sun however the mall simulates artificial lighting as well. Light sources are the display windows of shops. They are defined as Point Light GameObjects with parameters such as range, intensity or shadow type.

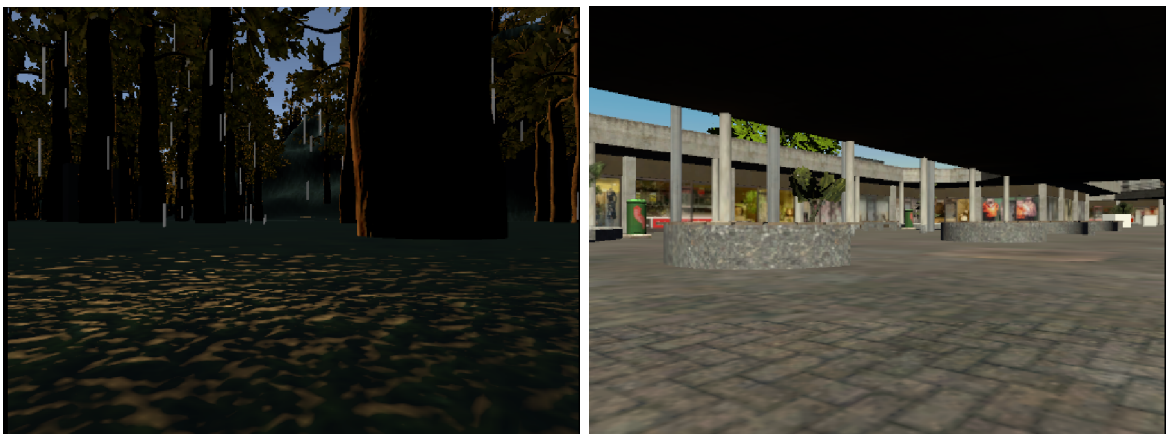


Figure 4.3: Camera view inside the HARDNAV simulation. Forest scene on the left. Mall scene on the right.

Simulated Robot

The robot's simulation is defined by GameObject called Jackal, containing several embeded GameObjects. Its first layer is made up of two GameObjects: Plugins and chassis_link.

Plugins defines connections between all parts of the robot. **Chassis_link** then contains all parts of the robot. Most are physical parts of the robotic platform Jackal, e.g. left front wheel, mid-mount. I do not list all the GameObects that are part of the Jackal GameObject as they are not important for the subject of this thesis. The important ones are: ROS Camera, Lidar, which is described in 4.3.1, and draw_line together with render_line, which are supporting GameObjects for placing obstacles as described in section 4.3.2.

ROS Camera

ROS Camera GameObject is the only sensor for steering the robot back to the traversed path when it diverges. Its specifications are available in the HARDNAV publication [10]. Most important for our purposes are the adjustable parameters like width and height—for the image to have correct dimensions as the visual module CNN's input. All data gathered by the camera are published on a ROS topic, from which they can be read the same way as they would from a real-world onboard camera. This feature allows me to use the same ROS workspace for both the simulation and the deployment in the real world.

4.2 Training VT&R Control Policy

As the training of a generic vt&r control policy for NNVTR has already been done in Rozsypalek et al.'s work, the system and the simulation were mostly ready for the task. I merely recorded new maps in both the forest and the mall scene. This section describes the state of the system as the training took place.

4.2.1 Creating Maps

The neural network learns by autonomously repeating previously traversed paths, i.e., by performing a classic vt&r scenario. I have recorded 20 maps in the simulation, 13 in the forest scene and 7 in the shopping mall scene.

Each map was recorded manually. The robot was first teleported to an arbitrary place in the scene and then teleoperated via a LogiTech controller along a trajectory. The simulated robot took

camera pictures roughly every 1 meter and recorded its odometry readings together with action commands for future dead reckoning. The distance between image capturing is affected by the curvature of the path, not only by the set value of 1 meter. Example images of the robot's view from both forest and mall scenes are displayed in Figure 4.3.

All maps were created with diversity of the learning dataset being the main goal. Shapes of the paths are arbitrary. Maps were recorded at different times of the simulated day, and a few with different speeds of time for less consistent lighting conditions as the sun moved across the sky. Few paths lead through dense vegetation to obscure the robot's view, others through densly forested areas with tight manouvering space. In contrast, there are paths traversed through environments favourable for a vt&r system, such as structured areas with even ground.

In 15 out of the 20 maps I have placed obstacles, which is discussed in section 4.3.2, therefore the NNVTR control policy was trained only on 5 maps. These maps were 3 in forest and 2 in mall. Lastly, I had to add a functionality to switch between the Unity scenes when starting a new map, as the original system was trained only in the forest scene and was unable to load different scenes.

Map in the context of this vt&r system is a folder containing a rosbag with two recorded ROS topics, odometry and action commands. Additionally, there are representations created by the visual module from the recorded camera pictures.

4.2.2 Training Process and Reward

When started, the training process takes place as follows. A map is chosen at random and the scene in which it was recorded is loaded. Afterwards, the robot's initial position is generated. First, a starting position at the first half of the trajectory is chosen randomly to create further variety in the traversals. The robot is then teleported to a random position within a few meter sqaure around the starting point to introduce an artifical offset from the traversed path which the system should correct. Information about how far along the path it was teleported is given to the robot, however the artificial offset is not.

The robot then attempts to traverse the path autonomously, employing both the visual and the control module as described in sections 4.1.1, 4.1.2. Active corrections are made approximately every 0.2 meters. For each action t the robot takes, a reward is administered. The function for calculating this reward is defined as

$$R = \Delta d - \Delta d_{err} - \Delta o_{err}.$$

Δd is the length of the path traversed between actions t and $t-1$. It is taken between the two points on the path closest to the robot after each action. This component keeps rewarding the system for not failing the traversal. The further the robot advances along the path before failing the higher the reward it cumulates. Δd_{err} is a difference of errors of the robot's distance estimates between actions. If the action t makes the distance estimate more precise, the received reward is positive and vice versa. Similarly Δo_{err} is a difference between the robot's offsets from the path—distance of the robot's position from the closest point on the taught path. Again, the reward is increased if the action gets the robot closer to the correct trajectory.

A successful repetition of a path is considered only if the robot reaches the end. The traversal is terminated sooner and an additional reward of -3 is given if one of the following happens.

Invalid Distance Estimation: If the estimation of the distance travelled is off by more than 3 meters, i.e., $d_{err} > 3$. The value of 3 meters was chosen because teach images are spaced approximately 1 meter apart and 2 images forward and backwards from the current index of distance travelled are matched. Therefore with a distance offset higher than 3 meters the teach image that actually corresponds to the current camera view is not within the 5 images taken into consideration and the offset is impossible to calculate precisely.

Too Far from the Map: If the robot's distance from the path is greater than 5 meters, i.e., $o_{err} > 5$. The value of 5 meters was chosen arbitrarily.

Not Moving: If the robot does not move for more than 5 seconds, it is considered stuck and the traversal is terminated as the robot is incapable of travelling any further. This happens for example when the robot hits a wall and keeps driving forward.

Flip: If the robot flips on its back it is not able to continue the traversal in addition to it not being a desirable behaviour.

To prevent overfitting to certain scenarios the reward is contained within $(-3, 3)$ interval. The rewards are backpropagated through the neural network to adjust its policy to prevent the situation in the future. However, as the system has no memory it affects only a limited number of last executed functions. This design serves the purpose of not affecting the entirety of the traversal. In the case of the -3 failure reward it only prevents the robot from performing actions bringing its state over the failure threshold without affecting the rest of the actions taken during the traversal which may have been correct. For example if the robot's current offset is 4.9 meters, its policy may still guide it to turn even further away because it assumes it will get closer to the path in 10 actions, a common scenario during turns. The goal however is to follow the taught path not just to arrive at the end position, making this behaviour undesirable.

The training process took approximately 3 computational days using NVIDIA GeForce RTX 3080 Ti GPU, AMD Ryzen 9 3950X 16-Core processor and 128 GB RAM. The resulting control policy agent is experimentally evaluated in section 5.

4.3 Training VT&R Control Policy with Obstacle Avoidance

The vt&r system is determined by a visual module and a control policy. The control policy is created by an RL training scheme. Therefore a new control policy, and thus a new vt&r system, can be created without rewriting the entire structure of the algorithms, but merely by changing the training data, input or output of the neural network and possibly the rewards administred during training. Relatively easy adjustments like these allow for creating a vt&r navigation system with additional functionalities, which would otherwise be complicated to define by an exact algorithm [8, 37].

In this thesis, I attempt to train a vt&r control policy capable of avoiding obstacles. For such task a neccessity to detect the obstacles arises, which is a rather complicated problem for one monocular camera alone, but an easy one for a depth sensor. The new vt&r system—CAVTR—is designed for a robotic platform carrying both a camera for heading corrections and a lidar for obstacle detection.

This section describes alterations made to the system for the development of the enhanced control policy. First, I detail the implementation of a simulated lidar for obstacle detection. Then I describe alterations made to the simulation. Afterwards, I move on to the necessary adjustments made to the RL scheme, such as including the lidar readings in the observation space. The resulting vt&r navigation with additional lidar sensory inputs is capable of traversing a previously known path, however, as shown by the experimental evaluation 6, the collision avoidance ultimately fails. Reasons why it is so and possible rectifications are discussed in the results and the future works chapters 6, 7.

4.3.1 Lidar Implementation

The main alteration made to the simulation was the lidar sensor. As the obstacle avoidance is the first additional functionality of the vt&r system, I have chosen to simplify the problem of collision avoidance by presuming all encountered obstacles to have a uniform enough shape to be avoidable based on the detection by a 2D lidar. Therefore any hurdles in the path, which appear under the lidar's laser scan are not considered. Additionally, a 2D lidar sensor's output has low dimensionality, making it easier for the neural network to effectively act based on its readings.

The implementation of the lidar into the RL scheme as a whole comprised three parts. Firstly, the simulated sensor itself was integrated onto the robot inside the Unity HARDNAV simulation. Secondly, the processing of the sensory data in the ROS workspace, which integrates the vt&r system with the RL scheme, together with the subsequent passing of the lidar readings into the observation

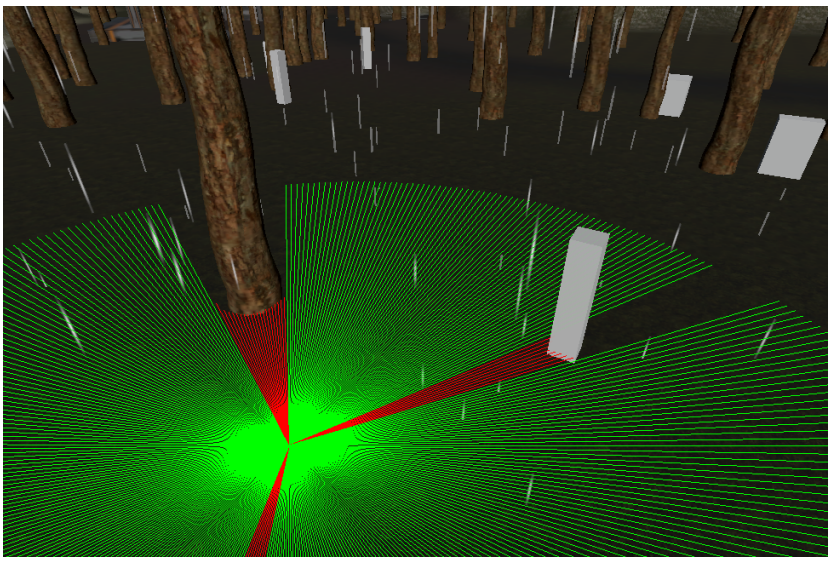


Figure 4.4: Visualisation of simulated lidar detecting distance of objects. The robot with the lidar mounted on top is located in the centre of the rays. Green rays reached their maximum range of 20 meters without detecting anything. Red rays collided with an object detecting its distance. The white slabs are obstacles placed in a few of the recorded paths. Light streaks are simulated rain.

space of the control policy neural network were implemented. Thirdly, trees in the forest scene used in Rozsypalek et al.'s work were not defined to be solid, making them undetactable by the lidar's rays. A description of solving this problem is in the section on altering the simulation 4.3.2.

Simulated Lidar

The lidar is a new GameObject added to the chassis_link, which is the main body of the simulated Jackal robotic platform. It is invisible and does not contain any physical body as there is no necessity for either. By adding the lidar as a nested GameObject of the chassis_link it stays attached to the robot and is on the same level in the robot's GameObject hierarchy as the camera. A C# script defining the sensor is attached to the lidar GameObject.

The simulated lidar fires rays at 10 Hz in a 360 degree span with 1 degree between each ray. These values, which are of course tunable parameters, were chosen because they closely resemble the parameters of our real-world lidar mounted on the actual Jackal platform. Each ray of the simulated sensor has maximum range set to 20 meters. As the paths measure approximately 50 meteres in length this value is considered to be substantial and is further scaled down during processing of the data by the vt&r system itself. Visualisation of one fire of the simulated lidar is in Figure 4.4.

The sensor data from the simulation lidar are published to a ROS topic */robot1/lidar* as a LaserScan message, which contains the distance at which each ray collides with an object. With the lidar parameters mentioned previously, the data have the form of an array with 360 float values. Publishing the lidar readings to a ROS topic in this way allows the implementation of their handling by the vt&r system to be identical for deployment in both the simulation and the real world, as the real-world lidar also publishes its measurements in the form of a LaserScan message on a ROS topic.

Processing the Lidar Data by the VT&R System

For processing the lidar readings a separate ROS node was created. It contains a subscriber and a publisher.

The subscriber reads the detected distances from the */robot1/lidar* topic and normalizes them

directly in the callback. The neural network's observation space is an array of float values in the range of $(-1, 1)$. This normalization is necessary for encoding the context of an observation together with its numerical value. For example, the proportion of green in a pixel and a distance of an obstacle can both have the numerical value 2, even though its meaning is entirely different for each. The normalization allows the neural network to use the same mathematical operations for both, without the need to include their context in said operation, because it is already included in the normalized value.

Additionally, all distance readings were cut to have a maximum value of 5 meters, to more closely resemble a real-world lidar sensor, which can provide imprecise readings after this distance. The original 20 meter range of the simulated lidar served the purpose of measuring the distance of the closest object beyond the 5 meter value.

These normalized readings are published to */lidar_processed* ROS topic as a custom ROS message which additionally contains the distance of the closest detected object. All other nodes of the vt&r system then work only with this topic. The distance of the closest detected object is not normalized, because it is not passed as an input to the neural network and therefore human-readable format is preferable.

As the goal of the new training scheme is obstacle avoidance, collision was added as a new reason for traversal termination. For training NNVTR the task was solely to repeat the path as closely as possible and it was assumed, that no obstacles are in the way as there were none during teach phase. Therefore terminating the episode prematurely would unnecessarily prevent the model from training on the rest of the path.

For the current task however it is not only undesirable for the robot to crash into anything, but even to get too close to a solid object. Training the system to leave a small gap ensures safer real-world deployment. Hence an autonomous path traversal is determined as a failure if the robot is closer than 0.3 meters to an object. The implementation of this condition employed the distance to the closest detected object, which the lidar ROS node publishes together with the normalized lidar readings.

4.3.2 Altering the Simulation

The forest scene in the simulation used for training of NNVTR policy contained trees which were not solid. As stated previously, this did not present a problem but rather an advantage because even if the robot hit a tree, it went through and then collected further training data for the rest of the episode. However, for the lidar to detect trees by the fired rays a collision mesh was neccassary to add to all trees in the simulation. Collision mesh defines the physical shape of an object within the simulation.

As stated in section 4.1.3, the trees were created by a Unity predefined function of a Terrain GameObject called 'Paint Trees'. It allows coverage of large areas with randomly spaced and variously shaped trees. However, the trees themselves are not GameObjects and can not possess a collision mesh.

As there are hundreds of trees in the simulation it was not feasible to replace them all by hand. I used a publicly availbale Asset called VegetationSpawner, which automaticaly converts all selected objects created by the Terrain's 'Paint' functions into GameObjects. Afterwards, a collision mesh could be added to all the new GameObject trees at once, making them detectable by the lidar.

Adding Obstacles

To present the system with scenarios in which it is forced to avoid a previously unseen obstacles I manually added several objects into the trajectories, which were traversed during teach phases. The creation of the maps is described previously in section 4.2.1.

The paths were created arbitrarily and the obstacles needed to be located directly in the robot's way. Before recording the maps I have added two new GameObjects to the chassis_link. These GameObjects have no visual nor physical form and are merely functions.

First GameObject called line_rendering recorded points spaced 0.1 meters apart through which the robot went during teach phase. It then saved these points into a file. The second GameObject

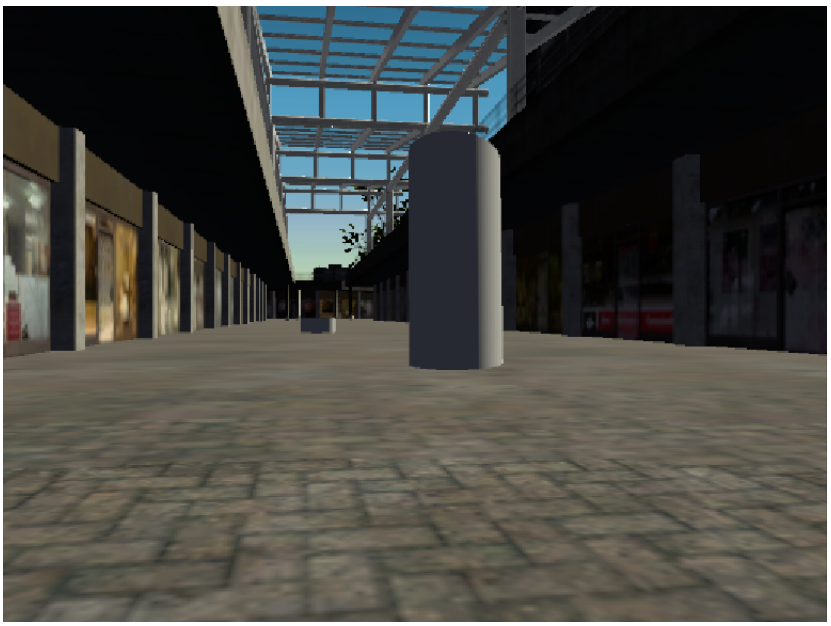


Figure 4.5: Camera view inside the simulation in the mall scene of two white cylinders placed as obstacles in the traversed trajectory.

called draw_line then loads a file with the name given as a parameter and plots the points in the Unity Scene view. The Scene view is a mode in which all editing of the simulation is done.

Such setup allowed me to see all paths and place obstacles of various shapes, sizes and spacings into 15 of the taught paths. As with recording the maps the obstacles are placed in a way that creates a ranging spectrum of difficulties for the autonomous traversals. An example of the robot's view of obstacles is in Figure 4.5.

4.3.3 Adjustment of the Reinforcement Learning Scheme

Adjustment of the RL setup had two parts. First, the normalized lidar data were added to the observation space of the control policy neural network. This was rather straightforward and comprised of fetching the data from the */lidar_processed* ROS topic and adding the array into the observation tensor.

Secondly, the reward function specified in section 4.2.2 needed to be altered to encourage the robot to avoid obstacles. We have discussed several approaches with my supervisor. The main two functions considered were an absolute reward and a difference reward.

The absolute approach would distribute rewards in direct proportion to the distance of the closest object. However, an ideal function would give a reward of zero for driving by an object located next to the path. Such objects are inevitably detected by the lidar. Lowering the range of the lidar is not feasible because the robot needs to detect an obstacle before it is located at the failure threshold distance of 0.3 meters to have time to steer away. For example driving along the desired path, which has an object located 0.5 meters next to it, is not an undesirable behaviour and therefore should not be punished by a negative reward. The absolute reward function does not ensure this and was therefore rejected in favour of the difference approach.

The implemented difference reward is calculated similarly to the distance estimate and the offset rewards. It is the difference between the distances of the closest objects of actions t and $t-1$. If an action brings the robot closer to the closest object a negative reward is given and vice versa. In the case of driving by an object located near the path, or even in the case of successfully avoiding an obstacle, the cumulative reward would be approximately zero, as it receives a negative reward when

approaching the object but then the same value only positive is awarded for travelling away from the object and continuing the path traversal.

In addition to distributing reward based on the lidar data I introduced an element to the reward function which is designed to more aggressively force the control policy to correct its estimation of distance traveled so far. The already included element Δd_{err} only rewards the robot when the distance estimation error changes. It is a difference reward for the distance estimation to encourage the robot to rely primarily on odometry readings and use images for sparse corrections of this estimate. However, as obstacles are added to the traversed path, the robot's view changes making it more difficult to effectively use camera data. Furthermore, when the robot avoids an obstacle, its distance estimate based on the odometry data is affected because it actively traverses a different trajectory with a different length than the taught path. For these reasons, I added an absolute reward element d_{err}, equal to one hundredth of the current error of the distance estimate, to punish the robot whenever the estimate is wrong. It is designed to force the system to actively correct this error to prevent its accumulation as obstacles are avoided.

The resulting reward function has the form

$$R = \Delta d - \Delta d_{err} - \Delta o_{err} - \Delta c - d_{err},$$

where Δc is the difference in the distance of the closest object between actions t and $t-1$.

As stated in the introduction of this section, the trained control policy failed to circumvent objects in its path. There are several problems with the newly introduced elements of the reward function, which are described in the results chapter 6 and their possible rectifications are discussed in the future work chapter 7.

Chapter 5

Experiment

The vt&r system with the newly trained control policy agent was deployed on a Clearpath Jackal robotic platform in a small backyard in daytime. The Jackal was carrying a Basler ace 2 camera and a LD06 lidar sensor.

Figure 5.1: Jackal platform with camera (on the left), lidar and a prism for tracking its position by Leica Total Station during traversal of a path (on the right). All evaluated vt&r system's were deployed on the Jackal platform.

The lidar fires 450 rays in a 360 degree span at a 10 Hz frequency. The number of rays fired each time varies slightly. The control policy neural network accepts lidar readings in a format of 360 wide array, therefore I had to scale the real-world lidar's array of distances down.

For measuring the ground truth of the robot's position a Leica Total Station TS16 R500 is used. It fires a laser at a tracking prism, which is mounted on the top of the Jackal platform to be visible from all sides. The position of the crystal is measured in a 3D space with 5 mm accuracy, however, as the traversed space was flat only two components, x and y, are used for evaluating the vt&r methods. It does not measure its orientation. The setup of the Jackal platform and the Leica Total Station tracking its position can be seen in Figure 5.1.

5.1 Experimental Scenario

An example of the robot's camera view of the experimental environment is in Figure 5.2. The ground was covered with grass and there were few trees on one side of the backyard. The other three sides of the space were walls of a building with windows and air conditioners. It was a rather structured

Figure 5.2: Camera view at the start of traversals. On the left is the view from the starting position of teach phase, in the middle from the start of the autonomous traversal with 2 meter offset, and on the right with an initial 4 meter offset.

environment with visual landmarks located near, which is favourable for a vt&r navigation system. The challenges of the environment consisted of repeating patterns on the walls, which can be difficult to distinguish and therefore to orientate by, and the relatively narrow manouvering space compared to sharpness of the turns in the short circle path.

In addition to deploying the newly trained CAVTR control policy, we have deployed NNVTR with control policy trained by me, as described in section 4.2, and Bearnav system [4] for comparison against a system with a control policy that does not consist of a neural network. All three vt&r navigation systems use the same visual module described in section 4.1.1 but a different control policy—CAVTR, NNVTR and Bearnav. Switching between these policies is parameterized.

The three systems were compared on a circle path. The robot was manually teleoperated across the trajectory, which is visualised in Figure 6.1. The one resulting map from this teach traversal was used by all three of the vtr&r systems, which is possible as only the visual module takes part in teach phase. The robot was then offset from the starting position and the autonomous traversal was initiated. In the following section 6, I present the results of runs with two different offsets. The first offset is approximately 2 meters—1.5 to the left and 1 forward from the starting position, and the second is roughly 4 meters—1 to the right and 4 forward from the start. The second offset was much larger and more difficult to correct because if the system did not converge fast enough, as was the case with Bearnav, it ran into a tree obscuring its vision.

To test the obstacle avoidance a straight path was recorded. Then a wooden pillar of roughly 30 centimeters in diameter and 1.5 meters in height was placed in the middle of the path. Autonomous traversal using the trained vt&r agent was initiated to see if it is capable of avoiding the previously unseen obstacle in its path.

Chapter 6

Results

In this section, the results of real-world experiments are presented and discussed. I start by describing the metrics used to calculate errors of the traversals—section 6.1. In section 6.2, graphs showing the evolution of the errors during the repetition of the circular path are presented, and problems in both performance and training of the newly trained vt&r systems are discussed. Afterwards, in section 6.3, I briefly show the results of the failed tests of obstacle avoidance during an autonomous traversal. Subsequently, I suggest possible causes of this failure, which concludes the results of the real-world experiment. A short explanation of evaluating the quality of a training run is given in section 6.4.

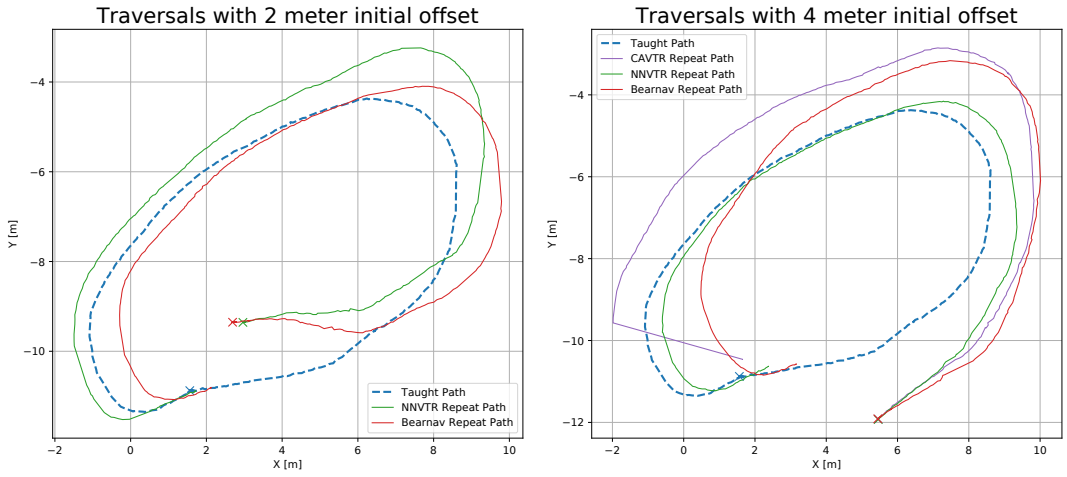


Figure 6.1: Visualisation of traversals of a taught circular path by the evaluated vt&r methods. The left graph shows runs with the 2 meter initial offset, the right graph with the larger 4 meter offset. Starting positions are marked with crosses.

6.1 Discussion of Evaluation Metrics

There are several metrics commonly used for the evaluation of a vt&r system's capabilities. I chose to plot the evolution of an error during the autonomous traversal to visualise both the speed of convergence of the vt&r method and its accuracy in following the taught path after it has converged. The error is calculated in two ways.

First metric, further called lateral error, is calculated as the distance between the robot's position and the closest point on the taught path. It provides simple and clear information about the navigation's ability to repeat the path accurately. However, one problem with this metric is that it

drops to zero when the robot crosses the taught path without being converged. Moreover, as the vt&r systems correct their heading they often quickly adjust for the initial offset in the perpendicular direction to the path but not in the longitudal, along the path, direction. This leads to the robot traversing with a wrong estimation of distance travelled so far and therefore wrong estimation of its position along the path, resulting in turning late or prematurly the next turn. This is visible in lateral error graph plots in the form of the error dropping to zero temporarily as the robot crosses the path, but then again increasing as the next turn arises.

The second metric, further called absolute error, is calculated as the distance between the robot's position and the position on the taught path with the same distance index. In other words, it is the difference between the robot's current position and the ideal position where it would be located, if it repeated the path perfectly with no initial offset. This metric does not drop to zero when the robot crosses the path, however, it does not account for the correction of the distance index within the vt&r system, which is performed by both NNVTR and CAVTR. It can therefore happen that the error will not drop to zero even if the robot follows the path perfectly because the distance index is adjusted internally, which the metric does not account for. If the robot does correctly adjust its distance estimation it should not diverge from the path even as it traverses the next turn and thus the lateral error will remain low, which is the reason, why two metrics are presented.

6.2 Results of the Experiment

In this section I present the results of the conducted experiments on three vt&r navigation methods:

CAVTR is the only method employing lidar sensory data. It was trained to be able to avoid obstacles.

NNVTR is the method taught using the same RL scheme as in Rozsypalek et al.'s work.

Bearnav as presented in [4] with the exception of using the same CNN visual module as CAVTR and NNVTR.

Results of repetitions of the path with two different starting artificially introduced offsets are presented.

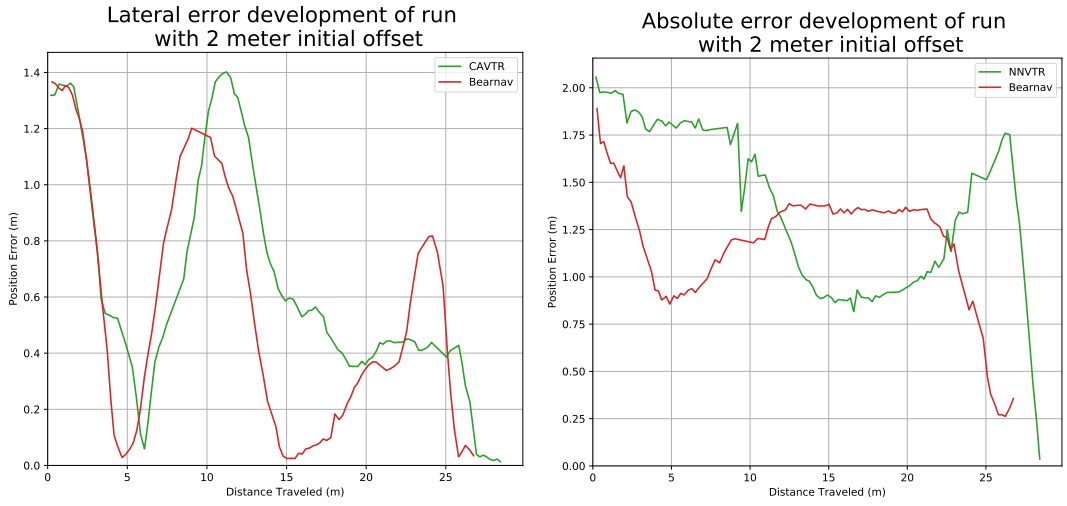


Figure 6.2: Development of lateral error in the left graph and absolute error in the right graph across distance traversed during an autonomous repeating of a circular path by two vt&r systems: NNVTR and Bearnav. Lateral error is the robot's distance from the path. Absolute error is calculated as the difference between the robot's current position during repeat traversal and the closest point on the taught path. The initial artificially introduced offset from the starting point was approximately 1.5 meters to the left and 1 meter forward.

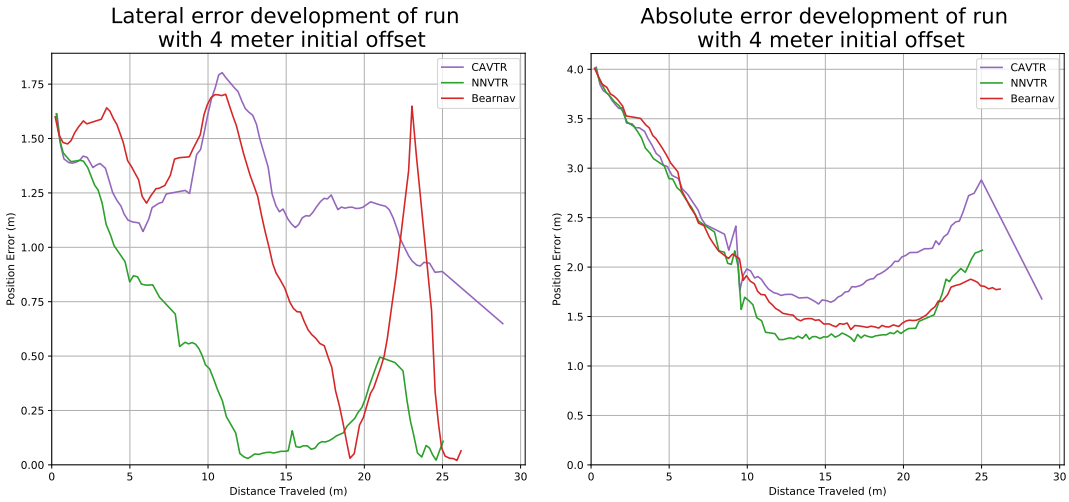


Figure 6.3: Development of lateral error in the left graph and absolute error in the right graph across distance traversed during autonomous repeating of a circular path by three vt&r systems: CAVTR, NNVTR and Bearnav. Lateral error is the robot's distance from the path. Absolute error is calculated as the difference between the robot's current position during repeat traversal and the closest point on the taught path. The initial artificially introduced offset from the starting point was approximately 1 meter to the right and 4 meters forward.

The first offset was 1.5 meters to the left and 1 meter to the front from the starting position. With this offset, only Bearnav and NNVTR were deployed. Their developments of errors during the traversal are displayed in Figure 6.2. In the lateral error graph the phenomena of the the error dropping to zero and then again increasing as the robot crosses the path can be seen. However, the overall trend of convergence is visible as well. The absence of the second spike in the NNVTR, but its presence in Bearnav's error, speaks to the correction of the distance estimate performed by NNVTR. Bearnav, which does not adjust its internal distance index, diverges at the last turn when the NNVTR has corrected part of its initial forward offset.

The second initial offset was approximately 4 meters forward and 1 meter to the left of the path's starting point. Developments of both lateral and absolute errors for CAVTR, NNVTR and Bearnav are shown in Figure 6.3. This larger forward offset more markedly shows the distance estimation factor of each of the vt&r methods. In the lateral offset, the spikes during the first turn are seen again for Bearnav and CAVTR. However, NNVTR does not show this as its distance estimation is more precise than CAVTR's. I suspect this is due to the absolute d_{err} element introduced into the reward function, which the training of NNVTR did not include. The lateral error of NNVTR has a lower spike in the end, containing the offset under 0.5 meters, whereas Bearnav, which does not perform explicit along the path correction spikes because it starts turning too soon due to its wrong distance travelled estimate as seen in Figure 6.1.

The fact that CAVTR has the highest absolute error supports the fact that it corrects its distance estimation too aggressively due to the d_{err} element. The absolute error shows this as it does not account for the internal correction, but only for the actions taken. Bearnav's and NNVTR's absolute error development behave similarly, which can be explained by NNVTR only adjusting its distance estimation when it is necessary, and therefore more scarcely than CAVTR. This would likewise explain its faster convergence in the context of the lateral error as the initial forward offset was 4 meters, which triggered the overshot correction of CAVTR.

Even though based on the 4 meter offset traversal, the accuracy of CAVTR is the worst of the three methods, it appears to behave more steadily—precisly—than Bearnav. I suspect the low accuracy is due to the too-aggressive distance estimation caused by the absolute d_{err} element in the

reward function in addition to training it for obstacle avoidance which may have negatively affected the path following by developing non-ideal strategies. However, the high stability of the method suggests that refining the d_{err} and introducing sequential learning into the process could provide a highly precise method. Sequential learning would separate the training of strategies for path following and for obstacle avoidance possibly resulting in a more precise vt&r control policy.

It should be mentioned that in Figure 6.3 Bearnav's and NNVTR's absolute error increases because of the final turn. They both started to turn too soon, due to the initial large forward offset which moved them in that direction relative to the path—backwards relative to the last turn. However, the CAVTR method again too aggressively corrected its distance estimation and thus overshot the last turn as seen in Figure 6.1. Similar behaviour is displayed by NNVTR in the 2 meter initial offset traversals in Figure 6.1, where, however, the forward offset was lowered more efficiently. The overshot of CAVTR caused it to traverse too close to the Leica Total Station which lost sight of the prism and was manually guided to measure the robot's final position after it had stopped, resulting in the straight line at the end of the plotted traversal in Figure 6.1 and the corresponding straight lines at the end of the CAVTR's error developments in Figure 6.3.

In a few traversals, which are not displayed here and had different offsets introduced at the starting point, CAVTR failed to detect the end of the path and instead kept driving forward. This was again caused by forcing it too aggressively to correct its distance estimation during training and thus it kept correcting its distance index to the second to last image instead of the last image, resulting in never determining it has reached the end of the traversal.

6.3 Obstacle Avoidance

To test if CAVTR is capable of avoiding an obstacle a simple straight path was recorded. A column was placed in the middle of the path, and the robot attempted to autonomously traverse the straight line with no initial offset. Visualisation of the traversal is in Figure 6.4.

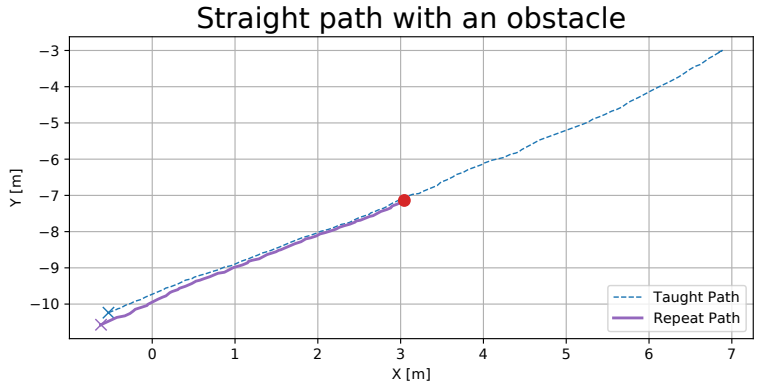


Figure 6.4: Visualisation of an attempted autonomous traversal of a straight path with an obstacle placed in the middle of it after teach phase. The obstacle is marked by a red point. Starting position of both teach and repeat traversals are marked with crosses.

The robot reacted in no way to the obstacle. It continued to traverse forward, eventually colliding with the column and the traversal had to be stopped manually.

The fact that the robot did not display any behaviour in reaction to the obstacle detected by lidar suggests the reward function to be defined in a way that does not encourage the robot to avoid obstacles. The collision element Δc of the reward is based on the difference caused by the last performed action rather than the current state of the system. The idea was for the reward cumulated during an avoidance manoeuvre to be zero as the negative part, received when approaching it, would be counteracted by the positive reward, gathered after driving away. However, it is possible this manoeuvre is too complex

or long for the current architecture of the neural network to be able to backpropagate the actions far enough to affect the actions performed at the beginning of the manouvre. This inability would cause the robot to never or scarcely achieve the positive reward, resulting in attempts to minimize the negative reward obtained as it approaches the obstacle.

Together with the other elements punishing the robot for any deviation from the traversed path, specifically the Δo_{err} and d_{err} elements, this may cause the robot to develop a policy which leads it directly into the obstacle. It does not gather the possible rewards from the rest of the path, but it does minimize the losses at the moment of detecting the obstruction as it maximizes rewards received for accurate path following. Attempting an avoidance manouvre would lead it away from the path, causing it to recieve additional negative reward to the one recieved for approaching the obstacle.

This is related to the fact that the Δc element is calculated only from the distance of the closest object. As it attempts to circumvent the obstacle the detected closest object keeps getting closer and only the angle under which the robot detects it changes. However, the angle is not taken into consideration by the reward function.

Additionally for quantitative evaluation average chamfer distances of each repeat run are shown in table 6.1. Note that this metric too is affected by crossing the path. As seen in Figure 6.1, Bearnav tends to converge by oscillating around the desired trajectory while CAVTR and NNVTR performing distance estimate correction can approach it gradually.

Table 6.1: Average chamfer distance for both the 2 meter offset run and the 4 meter offset run. CAVTR was not deployed on the 2 meter offset run.

	Chamfer Distances	
Method	2 meter offset	4 meter offset
CAVTR	x	1.16
NNVTR	0.54	0.47
Bearnav	0.47	0.86

6.4 Training Progress

During the entire training process several metrics were logged, using Wandb interface [40], to examine the process, determine its strengths and weaknesses, and to possibly terminate it in the case the agent is not learning properly. The most important metric is the cumulative reward, which is the sum of all rewards the robot has received. Cumulative reward gathered during the training of the model is in Figure 6.5. To see the progress of the training, each metric is reset after one epoch consisting of 1000 training steps. If the model manages to progressively get better at maximizing the reward function, the reward acumulated per each epoch will get larger. Therefore the expected behaviour of the cumulative reward is to gradually increase until it flatlines signalizing the model has stopped getting better and the agent is as trained as it can be with the current RL setup.

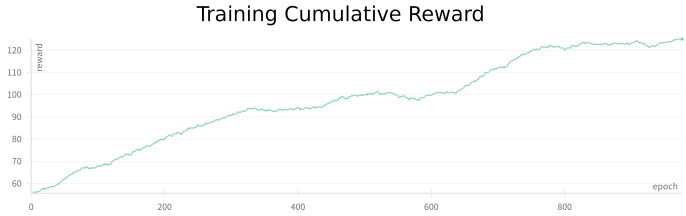


Figure 6.5: The progression of the cumulative reward per epoch acquired by the control policy neural network during its training.

Chapter 7

Future Work

As shown in the results section 6 the Rozsypalek et al.'s work of training a neural network to act as a control policy of a vt&r navigation system was successfully reproduced. The new control policy, trained to additionally be able to circumvent obstacles during the autonomous traversal of a taught path, however, failed in the collision avoidance task. The reward function used for the training of the neural network has the form:

$$R = \Delta d - \Delta d_{err} - \Delta o_{err} - \Delta c - d_{err}.$$

Each element is described in section 4.3.

The better-performing NNVTR was trained with a reward function:

$$R = \Delta d - \Delta d_{err} - \Delta o_{err}.$$

In the results section, it is shown that CAVTR is not only incapable of avoiding an obstacle, it performs worse in an autonmous traversal of a path with no obstacles as well. One of the problems is most likely in the newly introduced reward elements Δc and d_{err}.

I have previously trained a control policy without the d_{err} element in the reward function. The model is not discussed in this thesis as it was clear from only a few evaluation runs in the simulation, that it was not capable of obstacle avoidance. It however suggests that the problem is in both of the added elements, not just d_{err}.

In the results section 6, it is discussed that the d_{err} element may have caused the robot to correct its distance estimate too aggressively resulting in worse accuracy than NNVTR trained without it, leading me to the conclusion that it should be removed altogether in the future work.

Bearnav relies solely on odometry of the robot to estimate its position along the traversed path and camera data are only used to adjust the heading. It has been extensively evaluated in many works [4, 6, 39] and is a stable and robust vt&r system. Therefore I suggest for future reward systems to focus on relying heavily on the basic functionality of this approach and apply difference rewards, which encourage the robot to make corrections only when there is high probability of them being correct, instead of trying to eliminate a constantly incoming absolute negative reward.

The problems with Δc element are previously discussed in section 6.3. The angle at which the closest object is detected should be included in the reward. This would automatically solve the problem of the robot ideally receiving zero reward for driving by an object located near the taught path. When the robot detects it is located close to an object, but the current movement is not in the direction of said object, it can safely ignore it.

Furthermore, the reward should include more than one distance reading from the lidar to prevent situations where the robot avoids the closest obstacle only to run into another, or even into another part of the same obstacle if it has an irregular shape. The resulting function would likely be relatively complicated in contrast to the straightforward difference functions already included in the reward. Possible element, which could be introduced would reward difference in distances only in forward

facing narrow cutout of the lidar's field of view, thus encouraging the robot to steer away from the obstacle in order to detect the object under angle which does not lead to a crash.

The overall complexity of the obstacle avoidance problem became clearer to me during the work presented in this thesis. Another aspect of this fact is that it takes time for the control policy to learn how to traverse a path, causing the training of the obstacle avoidance itself to start late in the overall RL run and even then to not happen every run, but rather to take place inconsistently. The robot often failed the traversal before encountering any obstacle, therefore it could not train its circumvention as often as necessary. However, for learning traversal of a path it is shown that the original reward function used in Rozsypalek et al.'s work acts well. Hence I suggest employing sequential learning for the development of any further capabilities of the vt&r navigation system.

For the sequential learning not only the reward should be adjusted but the tasks presented to the neural network as well. The problem with the maps used for training CAVTR is that even with 20 maps the difficulty of the paths is not gradual enough. To say nothing of the fact they were presented all in one training run instead of gradually as the control policy developed its capabilities.

Firstly a generic vt&r system should be trained on various maps not including obstacles as shown Rozsypalek et al.'s work and in section 4.2. After the system is reliably able to traverse paths the reward function can be changed to encourage obstacle avoidance. However, a problem arises as the PPO training scheme consists of choosing actions with a degree of randomness, causing the system to fail even on the parts of the paths with no obstacles. A possible solution would be presenting the system with maps containing small obstacles close to the start of the path and positioned not directly in the way but only partly, making them avoidable with minimal deviation from the path. In this way, the control policy could learn what actions to take when encountering an obstacle, which can be made larger and more difficult to circumvent in future sequential learning runs. The datasets should always contain few paths with no obstacles to prevent overfitting to the collision avoidance problem and losing the capability of traversing an easy path without obstructions.

As stated in Rozsypalek et al.'s work additional functionalities could include not only obstacle avoidance but even more complex functionalities, such as avoidance of dynamic obstacles, kidnapped robot problem or social awarness.

Chapter 8

Conclusion

In this thesis, I presented an overview of the current state-of-the-art in visual teach and repeat navigation, discussing various approaches of tackling the problem. I then provided a detailed description of a vt&r system presented in Rozsypalek et al.'s work [12]. The system comprises two neural networks. One acts as a visual module gathering information about the surroundings and the robot's state. The other is employed as a control policy steering the robot along a previously taught path.

Afterwards I described reproducing said work by training a control policy with similar capability of autonomous repetition of a path. The training was conducted using the RL scheme developed and made publicly available by Rozsypalek et al. [11]. However, a different set of maps was used.

Furthermore I enhanced the system with implementation of a lidar sensor in the simulation environment and both the vt&r system and the RL scheme. The entire setup is currently ready for developing vt&r systems with additional functionalities employing a lidar sensor. Subsequently, I attempted to train a second control policy utilizing the lidar data to avoid obstacles during an autonomous traversal.

The two trained control policies were experimentally evaluated by deployment outside on a real-world Jackal robotic platform, confirming the successfull bridging of sim-to-real gap achieved by Rozsypalek et al. The experiments showed how the trained NNVTR outperformed Bearnav [4]. However, the CAVTR ultimately fails to avoid objects placed in its path. Possible causes and their rectifications were discussed extensively.

The main contributions of this thesis are reproducing of Rozsypalek et al.'s work, confirming that efficient vt&r system navigation can be trained using an RL scheme, following up on the work by attempting to create a vt&r system with an additional capability of obstacle avoidance and identification of crucial problems of the task. Suggestions of possible solutions were made, which we hope to subsequently implement in future works regarding the neural network trained to act as a control policy of a visual teach and repeat navigation system.

Bibliography

[1] Karel Čapek. *R.U.R.* Aventinum, 1920.

[2] Josep Aulinas, Yvan Petillot, Joaquim Salvi, and Xavier Lladó. The slam problem: a survey. *Artificial Intelligence Research and Development*, pages 363–371, 2008.

[3] Tomáš Krajník and Libor Přeučil. A simple visual navigation system with convergence property. In *European Robotics Symposium 2008*, pages 283–292. Springer, 2008.

[4] Tomáš Krajnık, Filip Majer, Lucie Halodová, and Tomáš Vintr. Navigation without localisation: reliable teach and repeat based on the convergence theorem. in 2018 ieee. In *RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1657–1664.

[5] Paul Furgale and Timothy D. Barfoot. Visual teach and repeat for long-range rover autonomy. *Journal of Field Robotics*, 27(5):534–560, 2010.

[6] Luis G Camara, Tomáš Pivoňka, Martin Jílek, Carl Gäbert, Karel Košnar, and Libor Přeučil. Accurate and robust teach and repeat navigation by visual place recognition: A cnn approach. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6018–6024. IEEE, 2020.

[7] Zdeněk Rozsypálek, George Broughton, Pavel Linder, Tomáš Rouček, Keerthy Kusumam, and Tomáš Krajník. Semi-supervised learning for image alignment in teach and repeat navigation. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, SAC '22, page 731–738, New York, NY, USA, 2022. Association for Computing Machinery.

[8] Zdeněk Rozsypálek, George Broughton, Pavel Linder, Tomáš Rouček, Jan Blaha, Leonard Mentzl, Keerthy Kusumam, and Tomáš Krajník. Contrastive learning for image registration in visual teach and repeat navigation. *Sensors*, 22(8):2975, 2022.

[9] Zdeněk Rozsypálek, Tomáš Rouček, Tomáš Vintr, and Tomáš Krajník. Multidimensional particle filter for long-term visual teach and repeat in changing environments. *IEEE Robotics and Automation Letters*, 8(4):1951–1958, 2023.

[10] Tomáš Musil, Matej Petrlık, and Martin Saska. Hardnav-simulator for benchmarking robust navigation and place recognition in large, confusing and highly dynamic environments.

[11] Zdeněk Rozsypálek, Tomáš Rouček, and Maxim Simon. NNVTR and CAVTR code. `https://github.com/Zdeeno/vtr-sim-ws`.

[12] Zdeněk Rozsypálek, Tomáš Rouček, Maxim Simon, Tomáš Musil, Martin Saska, and Tomáš Krajník. Learning control policy for visual teach and repeat. *The work is in review and is diclosed as an annex to this thesis.*

[13] Zhichao Chen and S.T. Birchfield. Qualitative vision-based mobile robot navigation. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2686–2692, 2006.

[14] Zhichao Chen and Stanley T. Birchfield. Qualitative vision-based path following. *IEEE Transactions on Robotics*, 25(3):749–754, 2009.

[15] Paul Furgale and Tim Barfoot. Stereo mapping and localization for long-range path following on rough terrain. In *2010 IEEE International Conference on Robotics and Automation*, pages 4410–4416. IEEE, 2010.

[16] Raúl Mur-Artal and Juan D. Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[17] Tomáš Pivoňka and Libor Přeučil. Orb-slam2 based teach-and-repeat system. In *International Conference on Modelling and Simulation for Autonomous Systems*, pages 294–307. Springer, 2020.

[18] Dominic Dall'Osto, Tobias Fischer, and Michael Milford. Fast and robust bio-inspired teach and repeat navigation. In *2021 ieee/rsj international conference on intelligent robots and systems (iros)*, pages 500–507. IEEE, 2021.

[19] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011.

[20] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9*, pages 404–417. Springer, 2006.

[21] Filip Majer, Lucie Halodová, and Tomáš Krajnık. A precise teach and repeat visual navigation system based on the convergence theorem. In *Student Conf. on Planning in AI and Robotics (PAIR)*, 2017.

[22] Payam Nourizadeh, Michael Milford, and Tobias Fischer. Teach and repeat navigation: A robust control approach. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2909–2916, 2024.

[23] Mona Gridseth and Timothy D. Barfoot. Keeping an eye on things: Deep learned features for long-term visual localization. *IEEE Robotics and Automation Letters*, 7(2):1016–1023, 2022.

[24] Matěj Boxan, Alexander Krawciw, Timothy D Barfoot, and François Pomerleau. Toward teach and repeat across seasonal deep snow accumulation. *arXiv preprint arXiv:2505.01339*, 2025.

[25] Colin McManus, Paul Furgale, and Timothy D Barfoot. Towards lighting-invariant visual navigation: An appearance-based approach using scanning laser-rangefinders. *Robotics and Autonomous Systems*, 61(8):836–852, 2013.

[26] Michael Paton, Kirk MacTavish, Michael Warren, and Timothy D Barfoot. Bridging the appearance gap: Multi-experience localization for long-term visual teach and repeat. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1918–1925. IEEE, 2016.

[27] Matias Mattamala, Nived Chebrolu, and Maurice Fallon. An efficient locally reactive controller for safe navigation in visual teach and repeat missions. *IEEE Robotics and Automation Letters*, 7(2):2353–2360, 2022.

[28] Jordy Sehn, Yuchen Wu, and Timothy D. Barfoot. Along similar lines: Local obstacle avoidance for long-term autonomous path following. In *2023 20th Conference on Robots and Vision (CRV)*, pages 81–88, 2023.

[29] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[30] Christopher John Cornish Hellaby Watkins et al. Learning from delayed rewards. 1989.

[31] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

[32] David Silver, Satinder Singh, Doina Precup, and Richard S Sutton. Reward is enough. *Artificial intelligence*, 299:103535, 2021.

[33] Albert Bou, Sebastian Dittert, and Gianni De Fabritiis. Integrating distributed architectures in highly modular rl libraries. *arXiv preprint arXiv:2007.02622*, 2020.

[34] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[35] Norwegian Broadcasting Corporation. Nordlandsbanen: Minute by Minute, Season by Season. `https://nrkbeta.no/2013/01/15/nordlandsbanen-minute-by-minute-season-by-season`.

[36] Zhi Yan, Li Sun, Tomáš Krajnık, and Yassine Ruichek. Eu long-term dataset with multiple sensors for autonomous driving. in 2020 ieee. In *RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10697–10704.

[37] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[38] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

[39] Maxim Simon, George Broughton, Tomáš Rouček, Zdeněk Rozsypálek, and Tomáš Krajník. Performance comparison of visual teach and repeat systems for mobile robots. In *International conference on modelling and simulation for autonomous systems*, pages 3–24. Springer, 2022.

[40] Wandb environment for logging RL training metrics. `https://wandb.ai/simonmax-czech-technical-university-in-prague/bakalarka_vtr-sim-ws`.