University of Tartu

Faculty of Science and Technology

Institute of Technology

Robert Allik

Validation of NoMaD as a Global Planner for Mobile Robots

Master's thesis (30 EAP)
Robotics and Computer Engineering

Supervisors:

Arun Kumar Singh, PhD
Karl Kruusamäe, PhD

Tartu 2024

Abstract/Resümee

Validation of NoMaD as a Global Planner for Mobile Robots

As autonomous mobile robots are becoming increasingly common in real-world applications, like warehouses and self-driving cars, so is the need for robust navigation methods growing. NoMaD, a vision-based navigation architecture, was recently presented and showed very good metrics in obstacle avoidance tested in ”challenging environments” but the exact level of challenge was unclear. This thesis aimed to measure that performance in a standardized way and implement an improvement by augmenting it with a LiDAR sensor. This new solution is based on ROS Navigation, where NoMaD acts as the global planner guiding the robot, leaving the obstacle avoidance to the local planner. Both NoMaD and the new solution were tested in environments inspired by the standardized navigation environment dataset BARN. While NoMaD reached the proclaimed success rate (90%) in simple baseline tests, it failed to do so in more complex environments, even when the hardware limitations of the setup were compensated for, with success rates ranging from 3.3% to 53.3%. The new solution, however, achieved all-around good results (83%) with no collisions. While it has its own drawbacks the new approach shows some merit.

CERCS: P176 Artificial intelligence; T125 Automation, robotics, control engineering.

Keywords: motion planning, ROS, diffusion models

NoMaD'il põhineva mobiilsete robotite globaalse planeerija valideerimine

Kuna autonoomsed liikurrobotid on muutumas üha tavalisemaks igapäevastes rakendustes, nagu näiteks laorobotid ja isesõitvad autod, kasvab ka vajadus robustsemate navigatsioonimeetodite järele. Hiljuti esitleti arvutinägemisel põhinevat navigatsiooniarhitektuuri NoMaD, mida katsetati "väljakutsuvates keskkondades" ning näitas väga häid tulemusi, kuid nende keskkondade täpne raskusetase oli ebaselge. Selle töö eesmärk oli mõõta NoMaD'i standardiseeritud viisil ja arendada sellest uus lahendus täiendades NoMaD'it LiDAR sensoriga. See uus lahendus põhineb ROS Navigation'il, kus NoMaD tegutseb robotit juhtiva globaalse planeerijana, jättes takistuste vältimise kohalikule planeerijale. Nii NoMaD'it kui ka uut lahendust katsetati standardiseeritud navigatsioonikeskkonna andmestikust BARN inspireeritud keskkondades. Kuigi NoMaD saavutas küllalt hea edukuse määra (90%) lihtsates testimis keskkondades, ei suutnud ta seda teha keerukamates keskkondades, isegi kui riistvaralisi piirangud kompenseeriti. NoMaD saavutatas nendes keskkondades 3,3% kuni 53,3%. Uus lahendus andis aga üldiselt häid tulemusi (83%) ja ei põrkanud kordagi kokku ühegi takistusega. Kuigi sellel uuel lahendusel on omad puudused, näitab see potentsiaali.

CERCS: P176 Tehisintellekt; T125 Automatiseerimine, robootika, juhtimistehnika .

Märksõnad: rajaplaneerimine, ROS, difusioonimudelid

Contents

List of Figures

List of Tables

List of Abbreviations

DWA - Dynamic Window Approach

LiDAR - Light Detection and Ranging

ML - machine learning

NoMaD - Navigation with Goal Masked Diffusion

ROS - Robobotic Operating System

RRT - rapidly exploring random trees

ViNT - Visual Navigation Transformer

1 Introduction

Mobile robots have already become everyday components of various real-world applications, ranging from autonomous cars [1] to warehouse logistics [2] and even warfare in the form of drones and ground vehicles [3]. In structured environments such as grid-based warehouses, traditional navigation methods using algorithms like A* or Djikstra's to compute paths tend to work well. But in places, like the outdoors or buildings designed for people, which can unpredictably change over time and have very complex shapes, these methods can have difficulty with localization [4], e.g. if the world no longer matches the map given to the robot, and be inefficient [5].

Machine learning (ML) based approaches, like the Diffusion Policy [6] based NoMaD [7], aim to learn behaviours that avoid obstacles and navigate based on vision. While diffusion is nowadays commonly used in text-to-image generation, it can be used to generate other kinds of data as well, in NoMaD's case, using images as inputs and producing navigable paths.

The authors of NoMaD consider it to be the first successful action diffusion model that can do both exploratory and goal-conditioned navigation by reporting a very high success rate, 90% to 98%, tested in "challenging indoor and outdoor environments". However, it is unclear exactly how challenging the environments really were.

1.1 Motivation

As mentioned, the exact difficulty of the environments was not discussed by Sridhar et al., but from the published images in the paper [7] and videos [8] showcasing NoMaD we can get some hints. They show their robot moving in typical human environments like corridors, sidewalks, and office buildings at a slow speed. This raises some questions: how well does NoMaD work in more cluttered environments, how much does velocity impact the performance, and can NoMaD be further improved?

1.2 Objective

The main goals of this thesis were to first evaluate the effectiveness of the NoMaD navigation architecture in moderate to highly cluttered real-life standardized environments. Then in an effort to improve the architecture, it was augmented with LiDAR sensing. To this end, NoMaD was integrated into ROS Navigation as a global planner leaving it to direct the robot, but having the LiDAR-based local planner be the one to avoid the obstacles. Then the new solution was benchmarked using the same methods.

A recent publication by Tampuu et al. looked at the effects of speed and delays of end-to-end

self-driving models and concluded that differing velocities in training and testing had adverse effects [9]. Although neither NoMaD nor the new solution presented here are end-to-end models, the effect of velocity was looked at by testing both solutions at two different velocities.

2 Background

This section will first introduce the concept of motion planning for mobile robots and describe ROS and its Navigation package as an example of a classical implementation of robot navigation. Then NoMaD is introduced as a novel approach to navigation based on *diffusion policy* [7]. Therefore, this section will also discuss diffusion: what diffusion models are, what diffusion policy is, and go into depth about NoMaD itself. Finally, a standardized benchmarking method is introduced.

2.1 Motion Planning

Motion planning is the method by which robots compute paths from their initial placement through an environment to a desired goal placement while avoiding collisions with obstacles [5]. Using a 2-dimensional example, the world is $\mathcal{W} = \mathbb{R}^2$, $\mathcal{O} \subset \mathcal{W}$ is the obstacle region, and the robot $\mathcal{A}$ is modelled as a rigid polygon, which can move through $\mathcal{W}$, but must avoid $\mathcal{O}$. The computation of paths takes place in configuration space (C-space) instead of $\mathcal{W}$, which is the set of all possible rigid-body transformations (position and orientation combinations) of the robot in $\mathcal{W}$. The robot's pose or configuration $q = (x_t, y_t, \theta)$ in 2D worlds is defined by x, y coordinates and θ orientation. Therefore, the C-space is three-dimensional. But if the robot in question has more degrees of freedom (DOF) then the dimensionality of C-space also increases. The C-space is divided into $\mathcal{C}_{free}$, which is the set of all non-colliding configurations, and $\mathcal{C}_{obs}$, which is all the colliding configurations.

Planning can be done in two ways [5]: combinatorial planning and sampling-based planning. Combinatorial planning works by constructing structures in the C-space that capture all the information needed for planning. Essentially, the whole C-space is vertically divided into trapezoid-shaped sections, and each section is then connected by a straight-line path. The downside of this approach is that it is slower as it analyzes the whole C-space. Sampling-based planning, on the other hand, works by iteratively exploring $\mathcal{C}_{free}$ for a path. An example of this is rapidly exploring random trees (RRT). This approach is much faster but may fail even if a solution exists as it does not look at the whole C-space.

2.2 Robot Operating System

ROS is a robotics-oriented open-source framework that provides a communications layer to node programs [10]. NoMaD was built for ROS1 [7], which this section will focus on, but it is coming to its end of life in 2025 [11] and is already in the process of being replaced by ROS2. ROS1 is implemented in a peer-to-peer topology, allowing very modular robot software design and it supports multiple programming languages, such as C++ and Python. The network is controlled by *ROS master*, which provides a lookup mechanism to allow ROS nodes to find

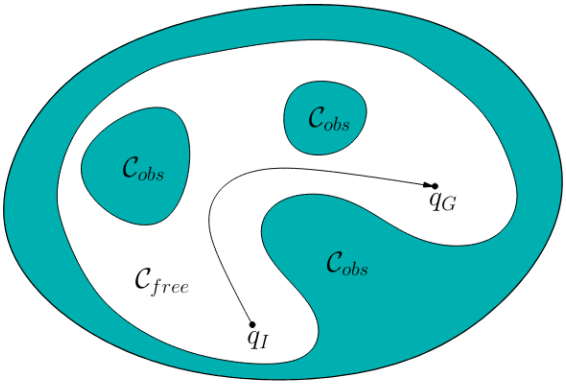

Figure 2.1: Example of C-space [5].

each other. Communication between nodes is done through either *topics* or *services*. *Topics* are message pipelines that can have many subscribers and many publishers. *Services* are, in contrast, unicast synchronous transactions, meaning that there is one response for each request. The modularity of ROS also lets users use previously developed solutions without having to tailor them to their robots such as ROS Navigation.

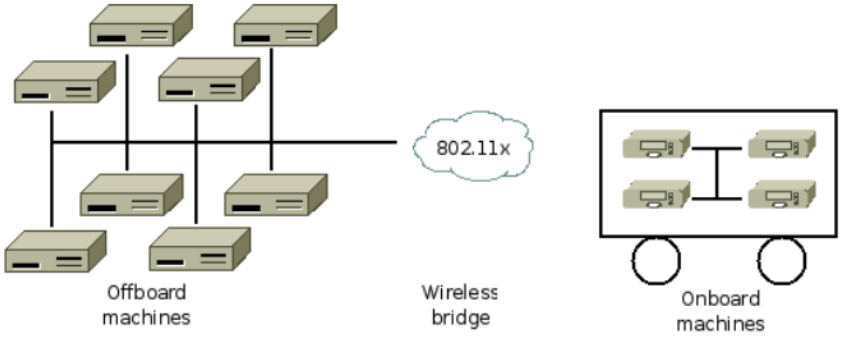

Figure 2.2: Example of ROS1 running on multiple machines [10].

2.2.1 ROS Navigation

ROS Navigation is a ROS package that takes in data from odometry and sensors and outputs velocity commands to a mobile robot [12]. It is meant for differential drive (turning by rotating wheels at different rates, not steering) and holonomic drive (driving in any direction without turning) robots. It requires a distance measurement sensor like LiDAR, which is used for map building and localization. Works best on robots which have a nearly square or circular shape. The components of navigation and the data flow are shown in Figure 2.3.

Navigation begins with knowing the coordinate frames and the relationships between them [13]. These answer the questions of where exactly the sensor is located on the robot and if the obstacle is 1 meter away from the sensor, how far it is from the centre of the robot. This is done using

the ROS Transform (tf) library as it provides a standard way to track the coordinate frames and the transforms of an entire system [14, 10]. The transforms are stored in a tree data structure, allowing them to have child transforms, which provides easy relative positioning, and fast searching.

The shape of a robot is described using Unified Robot Description Format (URDF), which is an XML format. It is comprised of links, which are the structural elements of a robot, and joints, which describe how links are connected and how they can move [15]. A tf tree is then built from the URDF file to describe all the coordinate reference frames for the whole system [10].

Odometry is the estimation of changes in position using motion sensors, like rotary encoders attached to the wheels of a robot. This allows the robot to measure how far and how fast it drives and provide that information to ROS. Odometry also provides its own coordinate frame, which is always centred on the robot and can be used for navigation and issuing goal pose commands. E.g. ”drive x meters forward and y meters left from the current location”, not ”move to coordinates x, y on the map”.

As the Navigation package outputs velocity commands, there must be a controller which handles them. This is done by a Base Controller, which reads the velocity commands and controls the robot’s motors to fulfil the commands.

Global and local planners

ROS Navigation uses two separate motion planning components - a global planner and a local planner, each with its own cost map [12]. The global planner creates paths over the entire environment using algorithms such as A* and Dijkstra’s algorithm [16]. The local planner tries to follow the path while avoiding obstacles using algorithms like Dynamic Window Approach (DWA) [17]. The costmaps are 2D data structures that are used to indicate a cost function, meaning how costly it would be for the robot to move to that location. E.g. the cost of free space would be 0; an obstacle would be 255; and areas near an obstacle would be somewhere in between so the robot would not go close to one unless needed to.

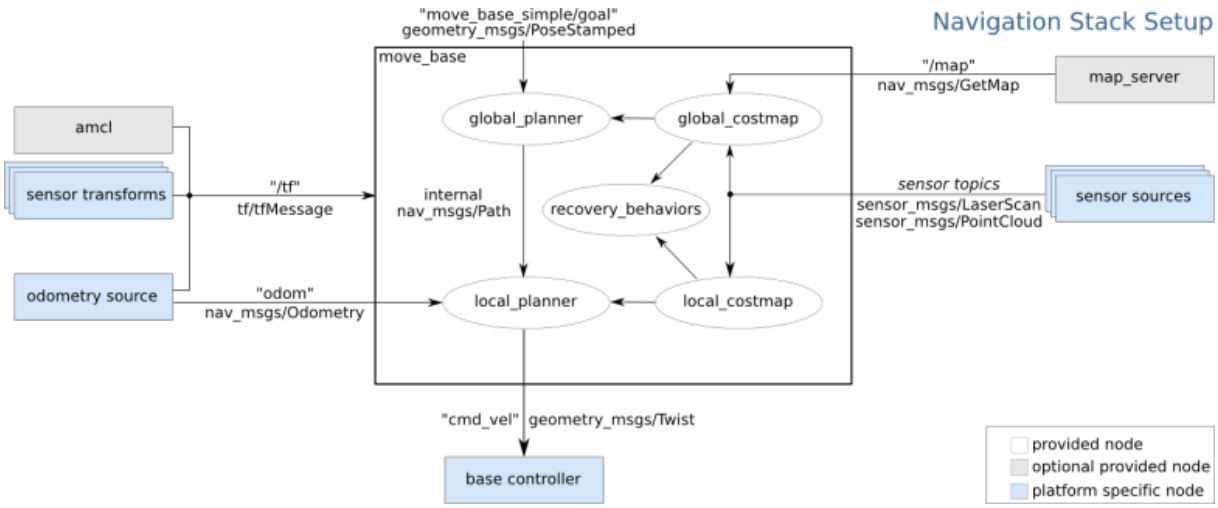

Figure 2.3: ROS Navigation diagram [13].

2.3 Diffusion Probabilistic Models

In 2015, Dickstein et al. introduced a novel way to model highly complex data sets using probabilistic models inspired by non-equilibrium statistical physics [18], where the structure of a data distribution is slowly destroyed in a forward diffusion process and then restored by a learned reverse process. A kind of Monte Carlo simulation called Langevin dynamics explains how to define a Gaussian diffusion process, which can have any distribution as its equilibrium. This is implemented using a Markov chain, which then iteratively converts one distribution of data into another (e.g. a standard Gaussian into a bimodal distribution).

The main idea is for the forward process to, over T time steps, destroy a more complex data distribution by adding some Gaussian noise to it and training the model (the Markov chain) to predict what noise was added. Then, the model performs the reverse process by iteratively subtracting some learned noise from a pure noise data distribution (see Figures 2.4 and 2.5). The end product is a new complex data distribution that is similar to the distribution of the training data.

Further progress was presented by Ho et al. in 2020 in their paper "Denoising Diffusion Probabilistic Models" (DDMP) [19] and by Nichol et al. in 2021 in their paper "Improved Denoising Diffusion Probabilistic Models" (iDDPM) [20].

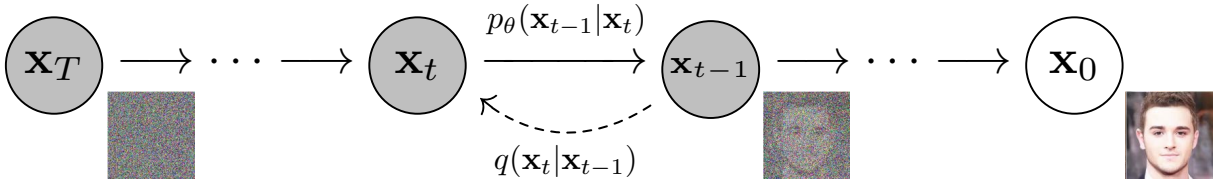

Figure 2.4: Directed graphical diffusion model [19].

2.3.1 Mathematical Preliminaries

This part will explain the diffusion processes using images as an example, where $\mathbf{x}_t$ is an image at timestep t. E.g. $\mathbf{x}_0$ is the original image, $\mathbf{x}_5$ is the image with 5 iterations of noise added, and $\mathbf{x}_T$ is the final iteration and is pure noise. The forward process is noted as $q(\mathbf{x}_t|\mathbf{x}_{t-1})$, which means $\mathbf{x}_{t-1}$ (image with less noise) is the input and $\mathbf{x}_t$ (image with more noise) is the output. As Equation 2.1 shows, calculating $\mathbf{x}_t$ is done by sampling a normal (Gaussian) distribution with $\sqrt{1-\beta_t}\mathbf{x}_{t-1}$ as the mean and $\beta_t I$ as the variance, where β_t is the noise scheduled for time step t by the scheduler β. Ho et al. chose a linear noise schedule: $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$.

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t, \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \tag{2.1}$$

Where $\mathcal{N}$ is the normal distribution; $\mathbf{x}_t$ is the output of $\mathcal{N}$; then the mean and variance; β is the noise scheduler ranging from 0 to 1; I is an identity matrix.

Equation 2.1 was further simplified in [19] to be able to be computed in one step. Note that as before, in Equation 2.1, the computation was done for one timestep; now, in Equation 2.2, the computation is done for any timestep in one calculation. This allows the model training process to be more efficient.

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}) \tag{2.2}$$

Where $\alpha_t = 1 - \beta_t$; $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$.

The reverse process (Equation 2.3) takes in a noisy image $\mathbf{x}_t$ and produces a less noisy image $\mathbf{x}_{t-1}$. Here the variance was fixed $\Sigma_\theta(\mathbf{x}_t, t) = \sigma_t^2 I$ giving the equation $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \sigma_t^2\mathbf{I})$ and could fixed to either $\sigma_t^2 = \beta_t$ or $\sigma_t^2 = \tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$, which have similar results but are optimal for different objectives. μ_θ is the reverse process mean function approximator, which is the part we want the model to learn (predicting the mean of the noise).Through some reparameterization, Ho et al. reach Equation 2.4, where $\boldsymbol{\epsilon}_\theta$ is a function approximator predicting $\boldsymbol{\epsilon}$ from $\mathbf{x}_t$ and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ (predicting the noise itself).

$$p(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}, \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t)) \tag{2.3}$$

$$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t))\right) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) \tag{2.4}$$

Ho et al. train the model by optimizing the usual variational bound on the negative log likelihood as seen in Equation 2.5. Meaning that the model wants to minimize the value of the negative log likelihood, which is approximated by the variational bound (L). Likelihood is the probability density (like the bell curve of a normal distribution) of the data given the parameter value. E.g. instead of calculating the probability of getting two heads in a row with flips of a fair coin, likelihood says how much support there is on the coin being fair given the observation of two heads in a row. Then log likelihood is the logarithm of likelihood.

Eventually, the following form is reached through some derivations and simplifications: Equation 2.6 [19]. In simpler terms, the loss function is the mean squared error between the noise added in the forward process and the noise predicted by the model (Equation 2.7) [6].

$$\mathbb{E}\left[-\log p_\theta(\mathbf{x}_0)\right] \leq \mathbb{E}_q\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] = \mathbb{E}_q\left[-\log p(\mathbf{x}_T) - \sum_{t\geq 1}\log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})}\right] =: L \tag{2.5}$$

$$L_{\text{simple}}(\theta) := \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t)\right\|^2\right] \tag{2.6}$$

$$\mathcal{L} = MSE(\boldsymbol{\epsilon}, \boldsymbol{\epsilon}_\theta(x_0 + \boldsymbol{\epsilon}, t)) \tag{2.7}$$

Finally, all of this is put together into Algorithms 1 and 2. Algorithm 1 starts by sampling an image from a dataset, sampling an array of integers between 1 and T for the time steps, and sampling noise $\boldsymbol{\epsilon}$ from a normal distribution. Then the model is optimized using Equation 2.6. Sampling or generating a new image is done using Algorithm 2: first sampling $\mathbf{x}_T$ as pure Gaussian noise, then using Equation 2.4 to get a less noisy image x_{t-1} in a loop until a noiseless image is reached. An example of a diffusion model trained on 2D data is in Figure 2.5.

2.3.2 U-net

Ho et al. use a U-net as the neural network architecture [19]. The U-net is a convolutional neural network (CNN) architecture designed for use cases like biomedical imaging, where in-

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

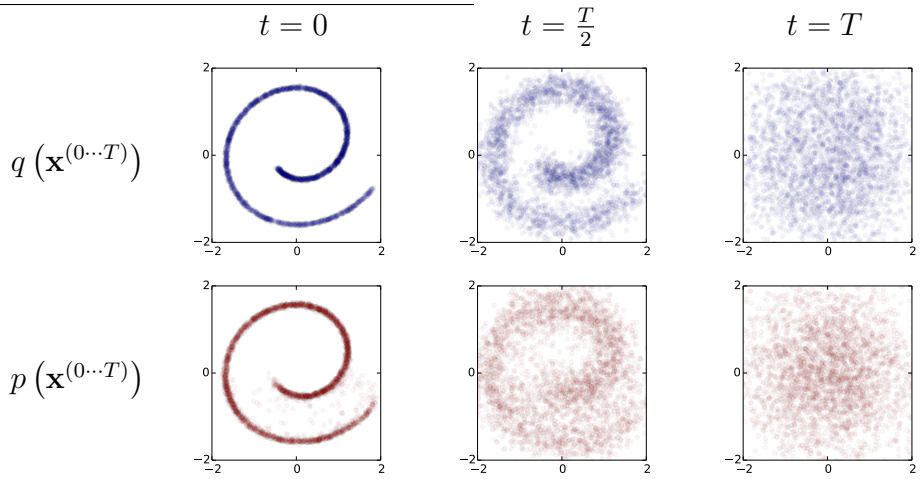

Figure 2.5: A diffusion model trained on 2D Swiss roll data The first row shows the forward process at the first timestep, middle timestep and last timestep. The second row shows the reverse process (read right to left), where a set of data with identity-covariance Gaussian distribution is turned into a distribution that resembles the original Swiss roll distribution [18].

stead of just the classification of the whole image, the output should include localization, i.e. a classification label for each pixel [21] (see Figure 2.6).

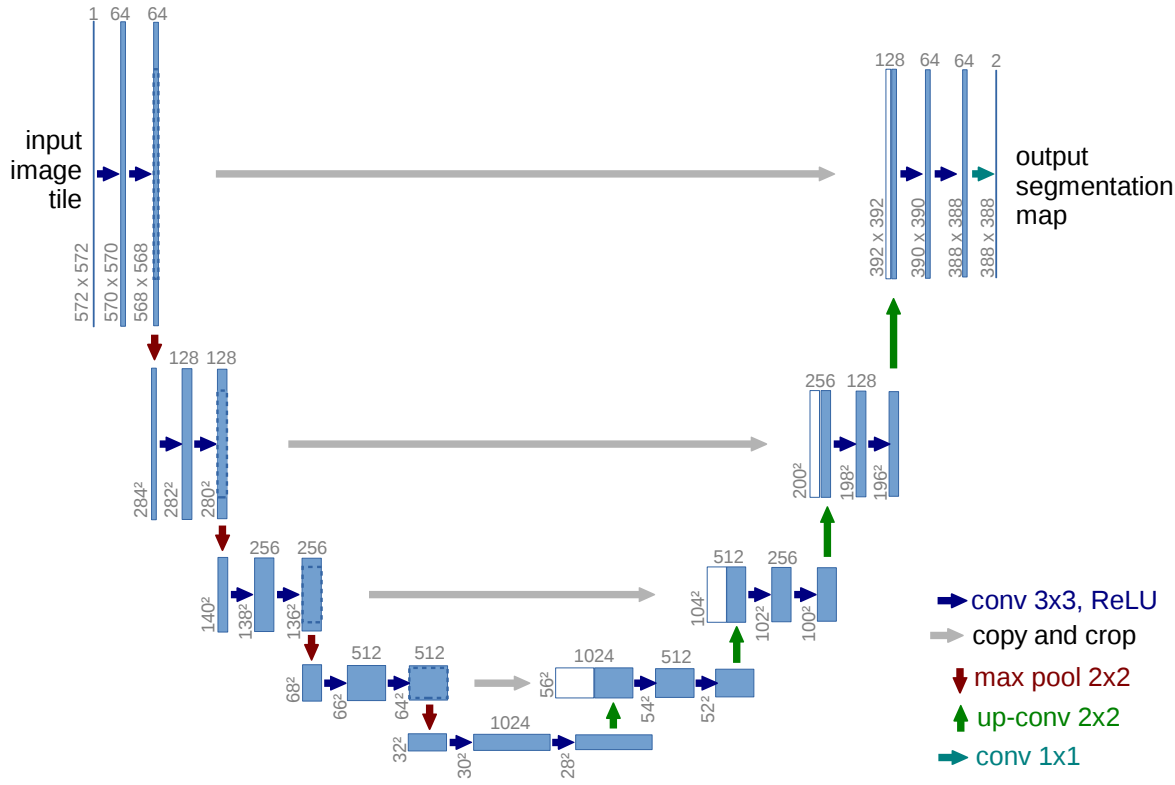

Figure 2.6: U-net architecture. Example by Ronneberger et al. [21] (note the U-shape). An image is sampled down to and then back up again, with intermediate downsampling results being concatenated during upsampling (the grey horizontal arrows), producing a segmentation map.

2.4 Diffusion Policy

Diffusion Policy by Chi et al. is a method of generating robot behaviour with a conditional denoising diffusion process [6]. This approach outperforms other state-of-the-art robot learning methods on average by 46.9% and allows models to learn multimodal behaviour (see Figure 2.8), generate a sequence of actions instead of a single action, and have stable training. While diffusion models are usually used in image generation, Chi et al. use them to learn *visuomotor policies* (learned behaviours with visual input and movement-based output).

They do this by first changing the formulation to output robot actions. This means that at time step t, the model takes in the latest T_o time steps of observational information $\mathbf{O}_t$ as input, predicts T_p steps of actions, and executes T_a steps. To clarify, T_o is the observation horizon, T_p is the action prediction horizon, and T_a is the action execution horizon, which can be set to less than T_p. Chi et al. state that this makes the actions more consistent without hurting responsiveness.

Secondly they use DDPM to approximate the conditional distribution $p(\mathbf{A}_t|\mathbf{O}_t)$ (distribution of actions $\mathbf{A}_t$ given observations $\mathbf{O}_t$) instead of the joint distribution $p(\mathbf{A}_t, \mathbf{O}_t)$ (joint distribution of $\mathbf{A}_t$ and $\mathbf{O}_t$) used in an earlier related work by Janner et al. [22]. The first section of Figure 2.7 shows a robotic arm getting a series of observations $\mathbf{O}_t$ as an input and then generating a series of waypoints (actions) $\mathbf{A}_t$ for the end effector to follow.

Diffusion Policy was examined with two model architecture options: CNN-based and transformer-

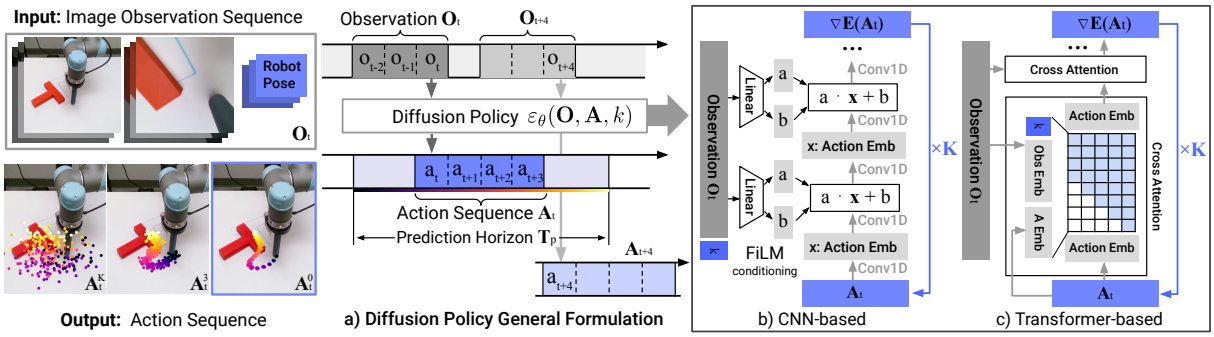

Figure 2.7: Diffusion Policy Overview a) shows the general formulation of Diffusion Policy. b) shows CNN-based Diffusion Policy. c) shows transformer-based Diffusion Policy [6].

based. The CNN-based architecture was based on the U-Net architecture mentioned earlier, but the two-dimensional spatial convolutions were replaced by one-dimensional spatial convolutions by Janner et al. [22] allowing dynamically changing input dimensionality (longer or shorter horizons) and further modifications by Chi et al. implementing the changes mentioned previously [6]. The transformer model is not relevant to this thesis as NoMaD uses the CNN model, but it proved to be better at more complex tasks at the cost of additional tuning. Figure 2.7 shows the general formulation of Diffusion Policy and the structures of CNN and transformer models.

Diffusion Policy makes use of a ResNet-18 based visual encoders to map images from the robot's cameras into the latent embedding O_t, which are trained together with the diffusion model. The noise schedule used in this work is the Square Cosine Schedule proposed by Nichol et al. in iDDPM [20], which adds noise more slowly and works better on smaller resolution images. Improvements in inferencing speed were made by employing the Denoising Diffusion Implicit Models (DDIM) approach [23]. This allowed Chi et al. to train their model with 100 denoising diffusion iterations, but only use 16 iterations in real-world inference, reducing latency.

Evaluation in simulated and real world environments [6] proved Diffusion Policy to have many major advantages over previous ML-based behaviour learning models like Long Short Term Memory Gaussian Mixture Model (LSTM-GMM, robomimic) [24], Implicit behavioral cloning (IBC) [25], and Behavior transformers (BET) [26]. Diffusion Policy learns behaviours that are multimodal, meaning it can accomplish tasks in a number of different ways and commits to one (see Figure 2.8). It can complete sub-goals in an inconsistent order, is better at position control compared to velocity control, and is good at idle actions (e.g. waiting while pouring a cup). Diffusion Policy was demonstrated by Chi et al. in real-world tasks like spreading sauce on a pizza and flipping a cup.

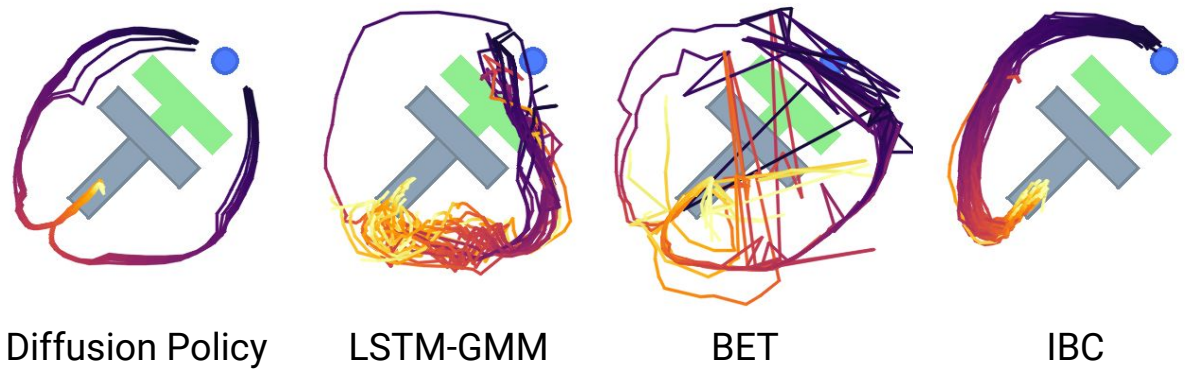

Figure 2.8: Comparison of how different models learned a task with Diffusion Policy showing good multimodal behaviour [6].

2.5 Navigation with Goal Masked Diffusion

Navigati**o**n with Goal **Ma**sked **D**iffusion (NoMaD) by Sridhar et al. is a Diffusion Policy based architecture meant for robotic navigation and exploration [7]. The authors consider this the first successful goal-conditioned diffusion model and the first optionally task-oriented model deployed on a physical robot. NoMaD is closely related to the Visual Navigation Transformer (ViNT) [27] navigation policy, even sharing many authors, and is essentially an extension of it. While ViNT uses a separate image diffusion model to generate candidate subgoals conditioned on the current observation, NoMaD, in contrast, generates a series of waypoints for the robot to follow. This approach requires 15 times fewer parameters in the model, making it much more efficient. The architecture of NoMaD is shown in Figure 2.9.

NoMaD uses the ViNT policy as the framework for processing input images. It uses an encoder ($\psi(O_i)$) to process observational images O_t and a goal fusion encoder ($\phi(O_t, O_g)$) to tokenize inputs. The usage of goal images is optional and can be turned off by a goal masking mechanism. The tokens are then processed by multi-headed attention layers (transformer), and the output is a context vector c_t, which is then used to predict future actions a_t using a Diffusion Policy model ϵ_θ and temporal distance between O_g and O_t. Here $i \in t - P, ..., t$; O_i is the last P observed images; O_t is the latest image; and O_g is the goal image.

NoMaD also makes use of episodic memory taken from ViKiNG [28] and is paired with a high-level planner. The memory is in the form of a topological graph with nodes being the robot's visual observations and edges being navigable paths between two nodes. The edges are determined by the temporal distance prediction. The addition of the memory and planner provides NoMaD with goal-seeking behaviour and allows it to navigate very large environments using sub-goals.

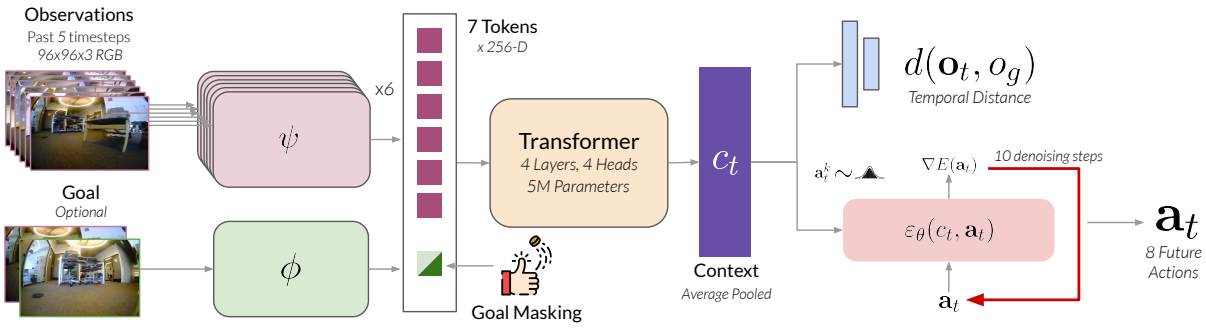

Figure 2.9: NoMaD Architecture Two encoders ψ, ϕ are used to tokenize observation and goal images. The tokens are then processed by a transformer into a context vector c_t. The context vector is then used to predict future actions using Diffusion Policy and predict how far the robot currently is from its goal [7].

PD Controller

NoMaD uses a proportional-derivative (PD) controller [29], which is a variant of the proportional-integral-derivative (PID) controller. A PID controller is a method of feedback control containing elements, which perform those mathematical functions [30]. It works by using the error between a goal value and the current value to scale the adjustment of the current value so that it would not overshoot or oscillate (e.g. the cruise control of a car). The Proportional element accounts for the error at instant t, the Integral element for the cumulative error up to t, and the Derivative element the derivative of the error at t.

NoMaDs PD controller is used to control its position. As the actions NoMaD generates are a sequence of waypoints for the robot to move to, the first one (closest to the robot) is sent to the PD controller, which then produces velocity commands for the base controller.

2.6 Benchmark for Autonomous Robot Navigation

The Benchmark for Autonomous Robot Navigation (BARN) is a dataset of 300 environments meant to be used as standardized tests for robot navigation [31]. The environments are generated using cellular automation, then the A* algorithm is used to plan a path in the $\mathcal{C}_{free}$ space. That path is then used to calculate the following metrics to characterize the generated environment:

1. Distance to Closest Obstacle - distance to the nearest obstacle averaged over all points in the path.
2. Average Visibility - average of the distances to obstacles in eight rays averaged over all points in the path.
3. Dispersion - average number of potential paths out of every point in the path.
4. Characteristic Dimension - the average tightness of every point in the path.
5. Tortuosity - how bendy a path is.

These metrics are used to predict a Combined Difficulty Level by a learned ML model. Some example environments are in Figure 2.10.

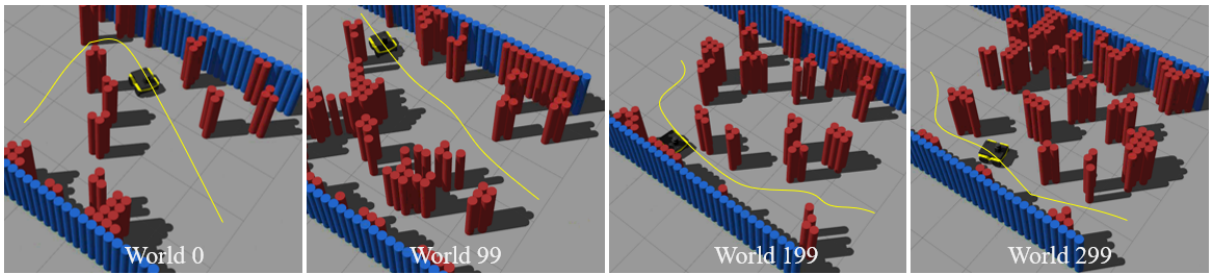

Figure 2.10: Example BARN environments. Four environments simulated with Gazebo in order of increasing difficulty [31].

3 Requirements

In order to achieve the goals of benchmarking NoMaD and benchmarking NoMaD as a global planner the following requirements had to be met:

1. Generation of testing environments.
2. Acquisition of materials to build the environments.
3. Deployment of NoMaD on a suitable mobile robot.
4. Baseline testing of NoMaD.
5. Benchmarking of NoMaD at two different maximum velocities.
6. Integration of NoMaD into ROS Navigation as a global planner.
7. Benchmarking of NoMaD as a global planner at two different maximum velocities.

Baseline testing aimed to prove that the whole setup worked and that the NoMaD model was good enough to be used in further benchmarking if it could reach a similar performance that was achieved by Sridhar et al. [7] in very easy testing environments. The benchmarking intended to measure the performance of both NoMaD and the new solution in more complex environments and if the performance was dependent on the velocity of the robot.

4 Methodology

This chapter explains in detail how NoMaD was integrated as a global planner in ROS Navigation, what hardware was this deployed on and tested, and then the procedure for baseline testing and benchmarking.

4.1 Software implementation

The integration of NoMaD encompasses modifications to the NoMaD source code, the creation of a global planner plug-in, a ROS node to facilitate communication between NoMaD and the navigation nodes, and various ROS launch files and other scripts. The code and scripts used in this work are provided in a GitHub repository: link.

The result of the implementation is shown as a diagram in Figure 4.1. First, the NoMaD navigation code is modified by adding a ROS topic publisher to access the whole path it generated, and its PD controller was removed.

Second, a ROS node was written to listen to the NoMaD path messages. The author chose to use the last waypoint of the path as the goal for the local planner because preliminary testing showed it to work reasonably well. The option to use the full sequence of waypoints was considered but was rejected because the length of the path was relatively short and the local planner resolved a similar or even better path with just the last waypoint. In contrast, the original NoMaD implementation uses the first waypoint as the goal. A goal pose was put together from the chosen waypoint and published to ROS Navigation.

Third, a global planner plugin was developed to replace the default global planner. This plugin would act as a dummy and do no processing or path generation on its own. Its only job was to accept goal pose messages and set them as the goal for the local planner.

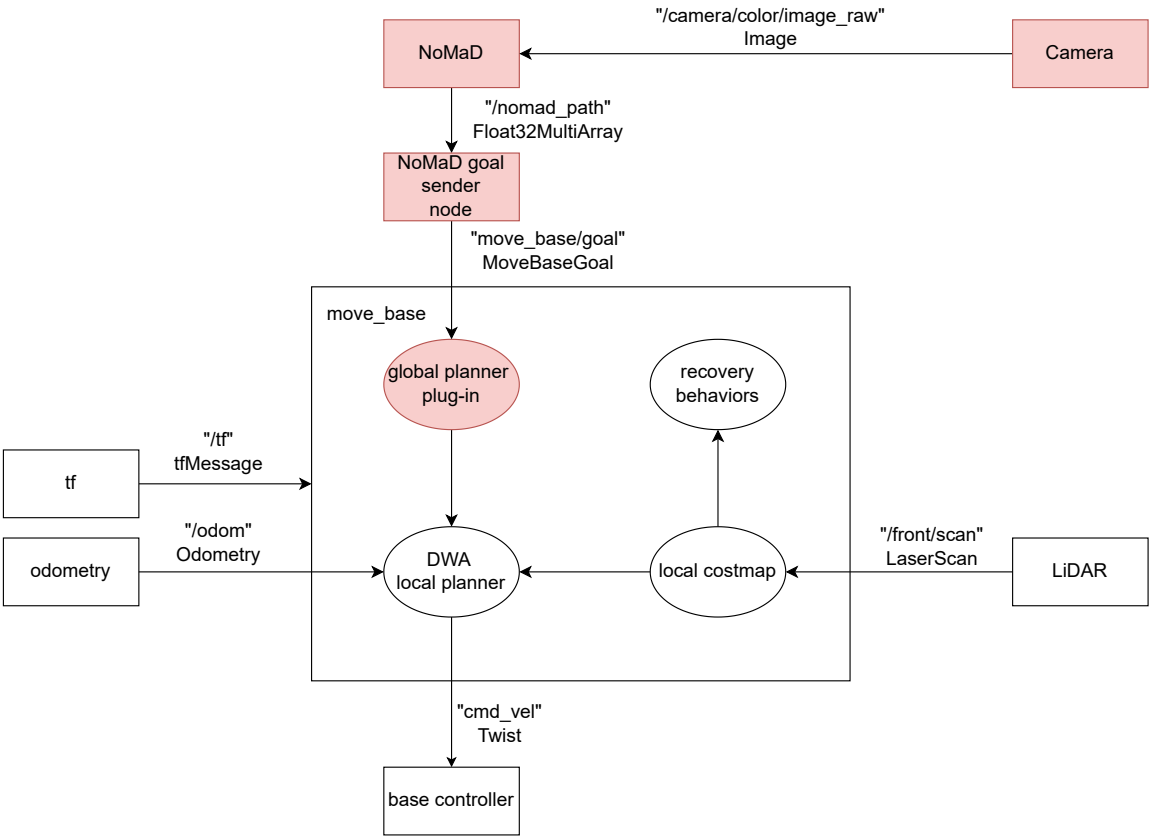

Figure 4.1: Implementation diagram. The red parts indicate the software components that were added to ROS Navigation as part of this thesis.

4.1.1 About training the NoMaD model

A pre-trained model, which was available on the NoMaD GitHub page [29], was used in this work. It was trained on RECON, TartanDrive, SCAND, GoStanford2 (Modified), and SACSoN/HuRoN datasets and some datasets that were unreleased [7].

The author made the decision not to train a new model based on the following: some datasets used in training the pre-trained model were unreleased, so there would have been fewer datasets to train with overall; making a new dataset based on the testing environment would have been prohibitively time-consuming; and NoMaD is meant to be able to handle unseen environments [7]. Baseline testing was used to prove that the pre-trained model was good enough to be used in further benchmarking.

4.2 Hardware

Both NoMaD and the integrated solution were deployed on the same hardware, which consisted of a small unmanned ground vehicle Jackal, an Intel D435i camera, a Sick TIM551 LiDAR, and a more powerful personal computer (PC). All of the software was run on the onboard computer of the Jackal except for the NoMaD model itself, which was run on the PC.

4.2.1 Clearpath Jackal

The Jackal by Clearpath Robotics is a relatively small entry-level rectangular robot meant for robotics research [32]. It has an onboard computer and is fully integrated with ROS. The Jackal was chosen because it was readily available at the University of Tartu and was already outfitted with the required sensors, which are described in the following sections.

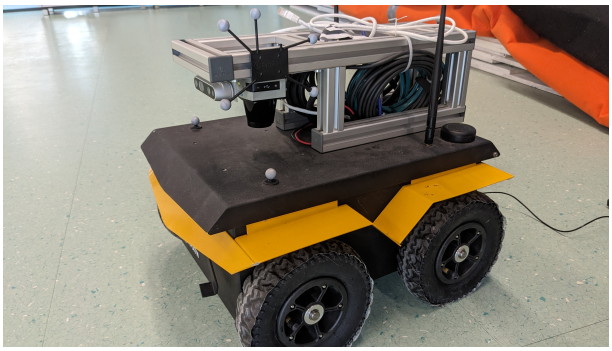

Figure 4.2: Clearpath Jackal outfitted with a camera and a LiDAR.

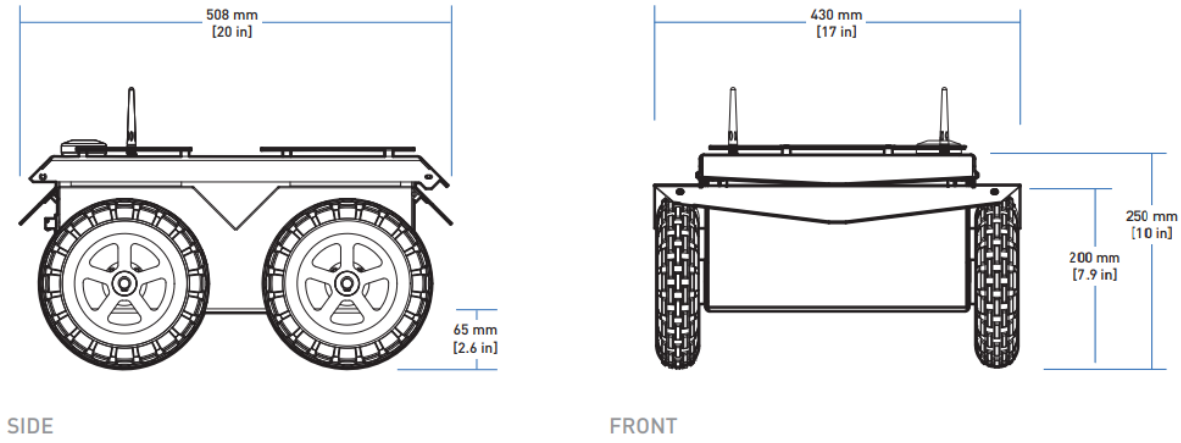

Figure 4.3: Clearpath Jackal dimensions [32].

Table 4.1: Clearpath Jackal specifications [32, 33].

External dimensions	508 x 430 x 250 mm
Drive power	500 W
Maximum speed	2.0 m/s
Weight	17 kg
Processor	Intel i3 4330TE
Memory	4 GB
Operating system	Ubuntu 18.04 LTS
ROS version	Melodic

4.2.2 Intel RealSense D435i Camera

The D435i is a multi-camera unit produced by Intel containing a regular RGB camera, a stereo depth camera, and an inertial measurement unit [34]. In this work, only the RGB camera part was used. It was mounted on the superstructure of the Jackal robot facing forward. This camera was not optimal for this task because its field-of-view is too narrow, 69 degrees [29], but issues coming from this could be mitigated to a certain degree and are discussed in the "Limitations" section. Relevant specifications are brought out in Table 4.2.

Table 4.2: D435i RGB camera specifications [34].

Resolution	1920 × 1080
FOV (H × V)	69° × 42°
Frame rate	30 fps
Interface	USB3.1

Figure 4.4: Intel RealSense D435i Camera [34].

4.2.3 SICK TIM551 LiDAR Sensor

The TIM551 is a light detection and ranging (LiDAR) sensor produced by SICK AG [35]. It was mounted on the robot's superstructure behind the camera. The data from this sensor was used to avoid obstacles near the robot. It was already mounted on the robot and deemed suitable for this case. Relevant specifications are brought out in Table 4.3.

Table 4.3: TIM551 specifications [35].

Measurement principle	HDDM+ (time of flight)
Light source	Infrared (850 nm)
Aperture angle	270 deg
Scanning frequency	15 Hz
Working range	0.05 m ... 10 m
Interface	Ethernet, TCP/IP

Figure 4.5: Sick TIM551 LiDAR [35].

4.2.4 Personal Computer

A PC was used to run the NoMaD model because it required a CUDA 10+ compatible graphics processing unit (GPU) [29] and the Jackal robot lacked a GPU. The robot and the PC were connected over a WiFi network. Specifically, a Lenovo Legion 5 17ACH6H laptop was used in this work and the relevant specifications are described in Table 4.4.

Processor	AMD Ryzen 7 5800H
Memory	32 GB
Operating system	Ubuntu 20.04 LTS
GPU	Nvidia RTX 3070 Mobile
GPU driver	nvidia-driver-535 (proprietary)
Python	3.8.5
CUDA	11
ROS	Noetic

Table 4.4: PC specifications

4.3 Configuration

This section highlights configuration options that are relevant or changed from their default values. They are already configured in the GitHub repository.

NoMaD related configuration:

- RealSense camera was launched with default options except ”initial_reset:=true enable_depth:=false”.
- NoMaD observation frame rate 4 Hz. This sets how often NoMaD accepts camera inputs and scales the length of the output path.
- Max angular velocity 0.4 rad/s.
- Max velocity 0.2 m/s or 0.4 m/s.

Local planner related configuration:

- Max angular velocity 0.4 rad/s.
- Max velocity 0.2 m/s or 0.5 m/s.
- Escape velocity -0.5 m/s.
- Costmap inflation radius: 0.2 m.
- DWA enabled.

4.4 Baseline Testing and Benchmarking

Two stages of testing were required to first prove that the whole setup is working correctly with baseline testing and then actually measure how well both architectures could navigate the environments with benchmarking. As this work also aimed to see if velocity had an effect on the performance of both NoMaD and the new integrated solution, the main tests were conducted at two different maximum velocities - for NoMaD 0.2 m/s (default speed) and 0.4 m/s; for the new solution 0.2 m/s and 0.5 m/s (default speed of the local planner). The NoMaD baseline tests were done using the faster speed. 30 runs were conducted for each test and for each speed, so 300 runs in total, and the only metric that was measured was the success rate - whether the robot could reach the goal without collisions. Cases where the robot had not collided yet, but was seemingly stuck, e.g. driving in a circle multiple times, were counted as a failure. The data was recorded in Google Sheets, and after all the runs were done, the average of the 30 results for each environment and speed combination was considered to be their success rate.

4.5 Testing Environments

Ideally, this thesis would have used actual BARN environments. However, they were too large to fit into the available room at the University of Tartu Delta Centre, and setting up the required number of obstacles would have been quite expensive. Therefore smaller environments had to be designed. Since the floor of the room was made from 60×60 cm tiles, they were used as a grid to place the obstacles, and since the room contained obstacles which could not be removed, they were incorporated into the designs of the environments. The obstacles were made from $40 \times 40 \times 60$ cm cardboard boxes, 20 of them were sourced from PAKENDIKESKUS AS for around 60 EUR, and nine 11.5×74.5 cm cardboard tubes, which were available at the university. The layout designs are in Appendix A.

In total four environments were designed - two for baseline testing and two for the actual benchmarking. Baseline layout 1 (see Figure 7.1) was meant to be as simple as possible and form the general structure for all the other environments. It can be described as a wide corridor or a small room, where there are no obstacles besides bounding walls and the goal lies directly ahead from the start point. This layout was designed to simulate an obstacle-free room to see if the whole setup was working correctly and that NoMaD could actually direct the robot to the goal. All other test layouts were based on the general design of this one but with increased difficulty.

Baseline layout 2 (see Figure 7.2) is similar to baseline test 1, but it is much narrower - about 3 widths of the robot, and the goal is again directly ahead from the starting position. This layout was meant to imitate one of NoMaD's demonstration videos, where it drives in narrow corridors.

The benchmarking layout 1 took baseline layout 1 and a single cluster of obstacles obstructing the path to the goal (see Figure 7.3). This layout aimed to be relatively simple to navigate, with multiple paths to the goal. The goal was still straight ahead from the starting position but obscured by the obstacles. All the spaces between the obstacles were wide enough for the robot to move through.

The benchmarking layout 2 was designed to pose a greater challenge, thus having a larger amount and wider distribution of obstacles (see Figure 7.4). The start and goal positions remained the same but with even more obstacles in the way.

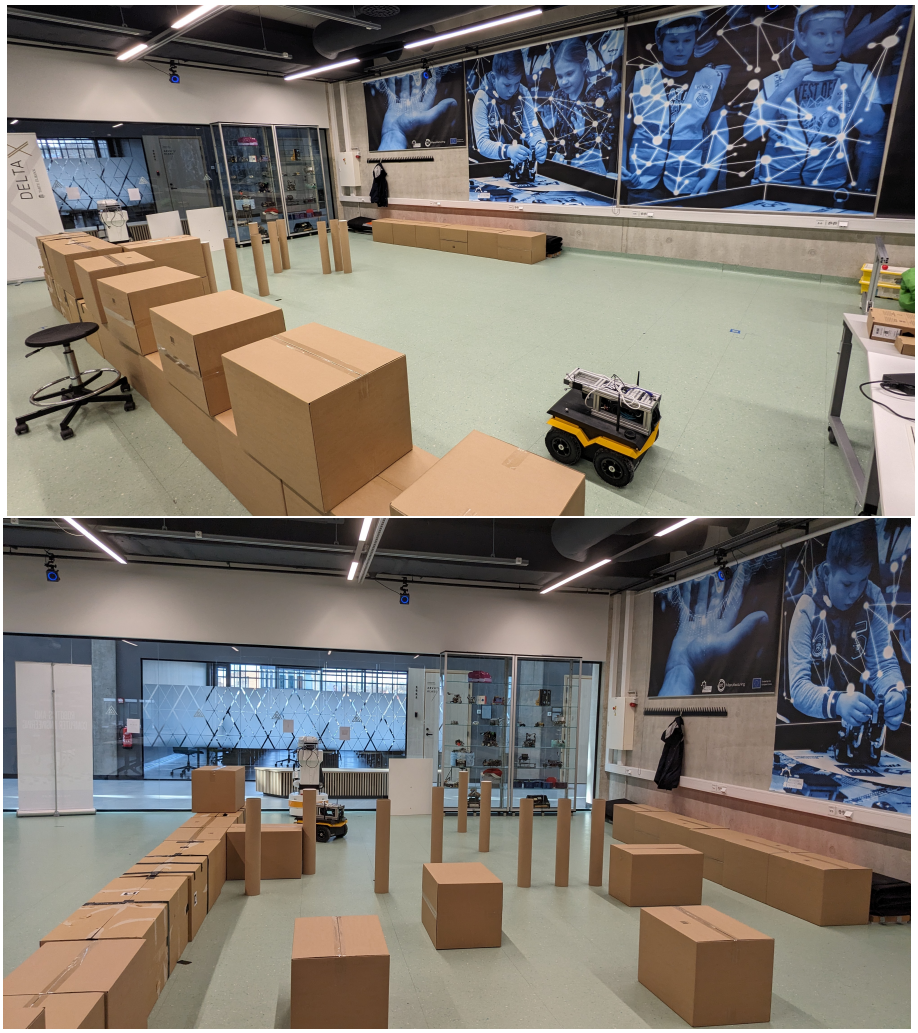

Figure 4.6: Pictures of benchmarking layout 1 (top) and 2 (bottom).

4.5.1 Validation Procedure

This procedure is the same for both baseline tests and benchmark tests.

1. Start camera and LiDAR nodes.
2. Make a topological map with the scripts from NoMaD.
3. Place the robot in the starting square (indicated as the arrow in the environment layouts) in the correct orientation.
4. Start ROS Navigation.
5. Start NoMaD.
6. Robot starts driving.
7. Robot reaches goal, collides with an obstacle, or fails to reach the goal.
8. Stop NoMaD and Navigation.
9. Record result.
10. Go to 3 unless finished.

5 Results

All the data gathered in testing is available in Appendix B.

5.1 Results of baseline tests

Table 5.1 shows the success rate of NoMaD in the baseline tests. With the pre-trained model and a maximum velocity of 0.4 m/s, NoMaD achieved a success rate of 96.7%, meaning it only failed 1 run out of 30 in baseline test 1. In baseline test 2 with the same model and velocity, NoMaD failed 7 times out of 30 leading to an average success rate of 76.7%.

Table 5.1: Averaged results of the baseline tests.

Success rate	Baseline 1	Baseline 2
NoMaD @ 0.4m/s	96.7%	76.7%

5.2 Results of benchmarking tests

Table 5.2 shows the average success rate of NoMaD and the NoMaD as a global planner solution in benchmarking tests 1 and 2. Both NoMaD and NoMaD as a global planner used the same pre-trained model. First, the NoMaD model reached a 53.3% success rate by succeeding in 16 runs out of 30 in test 1 and 26.7% in the more complex test 2 with 0.2 m/s maximum velocity. With the faster velocity, the results were worse at 23.3% and 3.3% (only 1 successful run out of 30).

NoMaD as a planner was successful 86.7% of the time (26 out of 30) at the lower maximum velocity in both environments. With higher maximum velocities, the success rates were 83.3% and 80.0% respectively.

Table 5.2: Benchmarking results.

Success rate	Benchmark 1	Benchmark 2
NoMaD @ 0.2m/s	53.3%	26.7%
NoMaD @ 0.4m/s	23.3%	3.3%
Planner @ 0.2m/s	86.7%	86.7%
Planner @ 0.5m/s	83.3%	80.0%

6 Discussion

This chapter will first discuss the limitations of the work done in this thesis, what was done to compensate for them, and then analyze the results.

6.1 Limitations

- The camera FOV of 69 degrees was too narrow according to issue #19 in the NoMaD GitHub [29], which suggested a 180-degree FOV camera.
- The camera could see over the boxes, meaning that the obstacles were not full-height walls. This could influence the path NoMaD generates.
- The camera was unstabilized. This could cause blurry images when the robot is accelerating.

Due to the camera limitations, the scoring of NoMaD by itself was made more lenient. For example, if an obstacle was out of view of the camera and the robot turned into it, it was not considered a collision (see Figure 6.1 for an example).

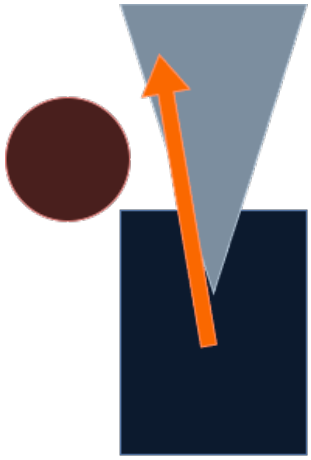

Figure 6.1: Example sketch of a situation where a collision would not be considered a failure. The black rectangle is the robot, the circle is an obstacle, the orange arrow designates the movement direction, and the grey triangle is the field of view of the camera (not to scale).

6.2 Analysis

The baseline tests set out to prove that NoMaD was deployed on the robot successfully, that the setup could achieve similar results to what the authors of NoMaD reported, and that the pre-trained model worked in the testing environment.

Both Baseline tests had high success rates (96.7% and 76.7%), which were comparable to what was presented by Sridhar et al. (90%) [7]. With these results, it can be said that the whole setup was working correctly and that the pre-trained model was indeed suitable for use in the benchmarking tests.

In the benchmarking tests, NoMaD failed to achieve such a high success rate. Visual observations seemed to indicate that the poor performance was because of the PD controller. For example, if the robot was too close to an obstacle it sometimes could not turn fast enough and had no recovery behaviours like reversing or turning in place. Increasing the maximum velocity significantly reduced the performance resulting in only one successful run in the more complex layout compared to eight successful runs at the lower speed.

Using NoMaD as a global planner produced consistent results with around 80% success rate in every benchmarking test. This was an increase of 30% to 60% compared to NoMaD alone and close to the 90% presented by Sridhar et al. Additionally, there was no performance degradation with higher velocity. There were zero collisions recorded, which was to be expected because the local planner actively avoided running into obstacles by overriding any goal pose generated by NoMaD. Still, the success rate was not 100%. The main failure mode was getting stuck in a situation where NoMaD says to go forward, but the local planner considers the path too narrow. This problem could possibly be alleviated by optimizing the parameters of the costmap and the local planner. Another common failure mode was the inability to commit to a path, so the robot drove around in a circle. Fixing this could possibly require higher-level logic to recognize this issue and then initialise some kind of a special behaviour, like circling the other way or positioning itself more head-on when entering narrow paths.

When comparing the two solutions, the benefit of integrating NoMaD with the classical robot navigation approach over just pure NoMaD is that it pretty much guarantees that the robot will not collide with obstacles. LiDARs also have a much wider FOV (270 degrees) than any normal camera, meaning that the robot is much more aware of its surroundings, compared to just relying on a camera. The requirement of a LiDAR is a disadvantage as well because they can be expensive (e.g. the Sick TIM551 costs around 3000 EUR [36]) and they require additional power, processing, and parameter tuning. A drawback of relying only on NoMaD would be that it cannot

7 Conclusion & Future Work

As reported by its authors, NoMaD, a recently presented state-of-the-art mobile robot navigation architecture, outperformed the previous state-of-the-art (Subgoal Diffusion) by 25%. Inspired by the success of NoMaD, this thesis set out to achieve three goals: benchmarking NoMaD in a standardized setting, developing a new solution by integrating NoMaD into the classical robot navigation approach as a global planner, and then benchmarking that new solution. The detrimental effect of velocity was also investigated by running the benchmarking tests at a lower and a higher velocity.

While it was not possible to use actual standardized testing environments like BARN, it was used as a guide to design similar environments that could be set up in the room that the author had access to. Two environments were designed for baseline testing and two for benchmarking.

Baseline testing of NoMaD showed that the robot was set up correctly and that the pre-trained model was usable in the testing environment by attaining a similar success rate to Sridhar et al. - 90%. Further testing in more complex obstacle layouts had a lower success rate, which deteriorated greatly when maximum velocity was increased. Although there was a definite limitation on the hardware side due to the narrow FOV of the camera, it was mitigated by more lenient grading, but head-on collisions were still head-on collisions. Visual observations by the author suggested that the main cause of the failures was the simplicity of its PD controller. The controller could sometimes not react fast enough and lacked recovery behaviours like reversing or turning in place.

The new solution using NoMaD as a planner was implemented successfully. In testing it reliably achieved around 80% success rate and recorded no collisions. Compared to the NoMaD benchmarks this is a 30% to 60% increase but still lower than the 90% presented by Sridhar et al. It also suffered no degradation when speed was increased, because the local planner was better at slowing down near obstacles. The main drawback of this solution is the need for a LiDAR, which comes with its own extra cost and processing requirements.

There are many ways the planner solution could be improved further. Removing the camera limitation by using a wider FOV camera should be the first to be addressed. On the software side, the path-processing ROS node could probably be removed by moving the functionality to either NoMaD or the global planner plugin. While the current design is not detrimental to performance, this would simplify it. Adjusting the parameters of the local planner and costmap would most likely provide the largest benefit. Implementing *receding-horizon control* from Diffusion Policy could also provide some benefit. This would mean using not the last waypoint of the sequence generated by NoMaD but one from the middle of the sequence.

Acknowledgements

I would like to thank:

my supervisors, Arun Kumar Singh and Karl Kruusamäe, for their guidance,

Fatemeh Rastgar, Houman Masnavi, Vishal Mandadi, and Kallol Saha for answering the questions I had and for their help,

my colleagues at Evikontroll Systems for their support and understanding attitude,

Grammarly for pointing out missing commas,

the country of Brazil for making good coffee,

and Larissa for her unwavering love and encouragement.

Bibliography

[1] Ardi Tampuu et al. “A Survey of End-to-End Driving: Architectures and Training Methods”. In: *IEEE Transactions on Neural Networks and Learning Systems* 33.4 (2022), pp. 1364–1384. DOI: 10.1109/TNNLS.2020.3043505.

[2] Nantawat Pinkam, François Bonnet, and Nak Young Chong. “Robot collaboration in warehouse”. In: *2016 16th International Conference on Control, Automation and Systems (ICCAS)*. 2016, pp. 269–272. DOI: 10.1109/ICCAS.2016.7832331.

[3] David Hambling. “Ukraine Prepares To Roll Out An Army Of Ground Robots”. In: *Forbes* (Mar. 14, 2024). Accessed: 2024-04-09. URL: https://www.forbes.com/sites/davidhambling/2024/03/14/ukraine-prepares-to-roll-out-an-army-of-ground-robots/.

[4] Muhammad Diginsa, Noraimi Shafie, and Nazir Yusuf. “Review: Issues and Challenges of Simultaneous Localization and Mapping (SLAM) Technology in Autonomous Robot”. In: *International Journal of Innovative Computing* 13 (Nov. 2023), pp. 59–63. DOI: 10.11113/ijic.v13n2.408.

[5] S.M. LaValle. “Motion Planning: The Essentials”. In: *Robotics & Automation Magazine, IEEE* 18 (Mar. 2011), pp. 79–89. DOI: 10.1109/MRA.2011.940276.

[6] Cheng Chi et al. “Diffusion Policy: Visuomotor Policy Learning via Action Diffusion”. In: *Proceedings of Robotics: Science and Systems (RSS)*. 2023.

[7] Ajay Sridhar et al. “NoMaD: Goal Masked Diffusion Policies for Navigation and Exploration”. In: *arXiv pre-print* (2023). URL: https://arxiv.org/abs/2310.07896.

[8] Ajay Sridhar et al. *NoMaD: Goal Masked Diffusion Policies for Navigation and Exploration*. https://general-navigation-models.github.io/nomad/index.html. Accessed: 2024-05-20. 2023.

[9] Ardi Tampuu, Kristjan Roosild, and Ilmar Uduste. “The Effects of Speed and Delays on Test-Time Performance of End-to-End Self-Driving”. In: *Sensors* 24.6 (2024). ISSN: 1424-8220. DOI: 10.3390/s24061963. URL: https://www.mdpi.com/1424-8220/24/6/1963.

[10] Morgan Quigley. “ROS: an open-source Robot Operating System”. In: *IEEE International Conference on Robotics and Automation*. 2009. URL: https://api.semanticscholar.org/CorpusID:6324125.

[11] *endoflife.date ROS*. https://endoflife.date/ros. Accessed: 2024-05-15.

[12] Eitan Marder-Eppstein. *ROS Navigation*. http://wiki.ros.org/navigation. Accessed: 2024-04-04.

[13] *ROS Navigation Tutorials*. http://wiki.ros.org/navigation/Tutorials. Accessed: 2024-04-22.

[14] Tully Foote. “tf: The transform library”. In: *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*. Open-Source Software workshop. Apr. 2013, pp. 1–6. DOI: 10.1109/TePRA.2013.6556373.

[15] *ROS URDF*. http://wiki.ros.org/urdf. Accessed: 2024-04-22.

[16] David Lu!! *Global Planner*. http://wiki.ros.org/global_planner. Accessed: 2024-05-15.

[17] Eitan Marder-Eppstein. *DWA Local Planner*. https://wiki.ros.org/dwa_local_planner. Accessed: 2024-05-15.

[18] Jascha Sohl-Dickstein et al. *Deep Unsupervised Learning using Nonequilibrium Thermodynamics*. 2015. arXiv: 1503.03585 [cs.LG].

[19] Jonathan Ho, Ajay Jain, and Pieter Abbeel. “Denoising Diffusion Probabilistic Models”. In: *arXiv preprint arxiv:2006.11239* (2020).

[20] Alex Nichol and Prafulla Dhariwal. *Improved Denoising Diffusion Probabilistic Models*. 2021. arXiv: 2102.09672 [cs.LG].

[21] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[22] Michael Janner et al. *Planning with Diffusion for Flexible Behavior Synthesis*. 2022. arXiv: 2205.09991 [cs.LG].

[23] Jiaming Song, Chenlin Meng, and Stefano Ermon. *Denoising Diffusion Implicit Models*. 2022. arXiv: 2010.02502 [cs.LG].

[24] Ajay Mandlekar et al. *What Matters in Learning from Offline Human Demonstrations for Robot Manipulation*. 2021. arXiv: 2108.03298 [cs.RO].

[25] Pete Florence et al. *Implicit Behavioral Cloning*. 2021. arXiv: 2109.00137 [cs.RO].

[26] Nur Muhammad Mahi Shafiullah et al. *Behavior Transformers: Cloning k modes with one stone*. 2022. arXiv: 2206.11251 [cs.LG].

[27] Dhruv Shah et al. *ViNT: A Foundation Model for Visual Navigation*. 2023. arXiv: 2306.14846 [cs.RO].

[28] Dhruv Shah and Sergey Levine. “ViKiNG: Vision-Based Kilometer-Scale Navigation with Geographic Hints”. In: *Robotics: Science and Systems XVIII*. RSS2022. Robotics: Science and Systems Foundation, June 2022. DOI: 10.15607/rss.2022.xviii.019. URL: http://dx.doi.org/10.15607/RSS.2022.XVIII.019.

[29] Ajay Sridhar et al. *NoMaD: Goal Masked Diffusion Policies for Navigation and Exploration*. https://github.com/robodhruv/visualnav-transformer. 2023.

[30] H. Unbehauen. *CONTROL SYSTEMS, ROBOTICS AND AUTOMATION - Volume II: System Analysis and Control: Classical Approaches-II*. EOLSS Publications, 2009. ISBN: 9781848261419. URL: https://books.google.ee/books?id=RF1xDAAAQBAJ.

[31] Daniel Perille et al. *Benchmarking Metric Ground Navigation*. 2020. arXiv: 2008.13315 [cs.RO].

[32] Clearpath Robotics. *Clearpath Robotics Documentation ROS 1 Noetic*. https://docs.clearpathrobotics.com/docs/ros1noetic. Accessed: 2024-05-16.

[33] *JACKAL UNMANNED GROUND VEHICLE*. Clearpath. 2020.

[34] *Intel® RealSenseTM Product Family D400 Series*. D435i. Revision 018. Intel. Mar. 2024.

[35] *LiDAR sensor*. TIM551-2050001. SICK AG. Apr. 2024.

[36] *TIM551 - Sick indoor and outdoor laser scanner for short-range measurement*. https://www.generationrobots.com/en/401699-tim551-sick-indoor-and-outdoor-laser-scanner-for-short-range-measurement.html. Accessed: 2024-05-19. Generation Robots.

Appendixes

7.1 Appendix A

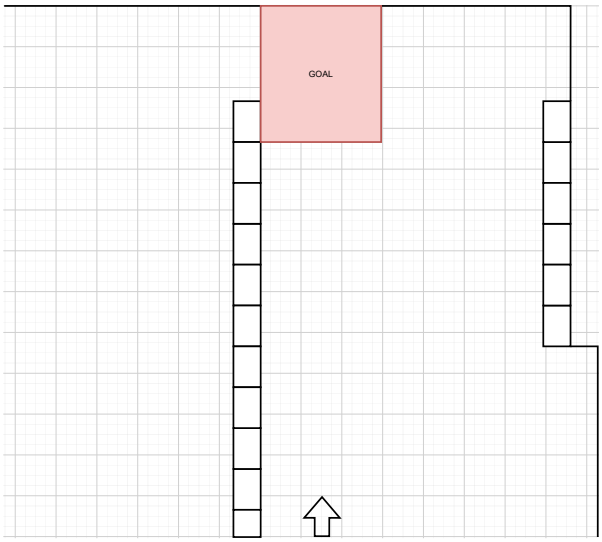

Figure 7.1: Screenshot of baseline environment 1. The grid is 60×60 cm and the obstacles are 60×40 cm.

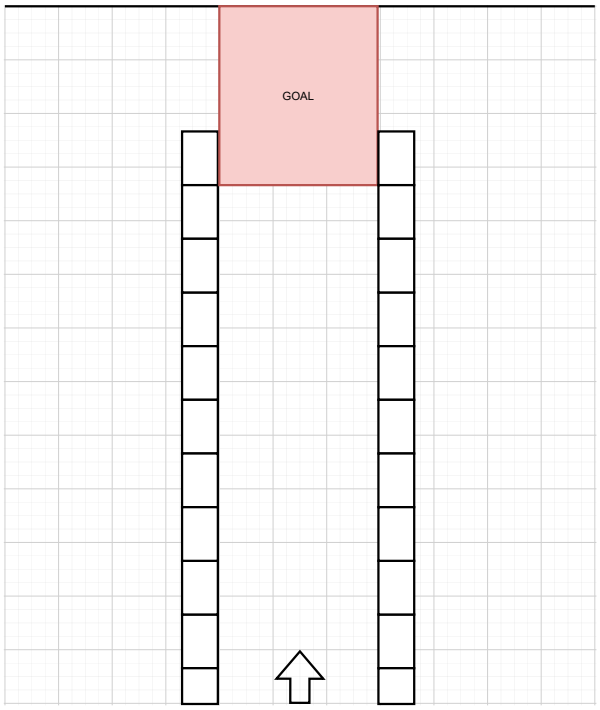

Figure 7.2: Screenshot of baseline environment 2.

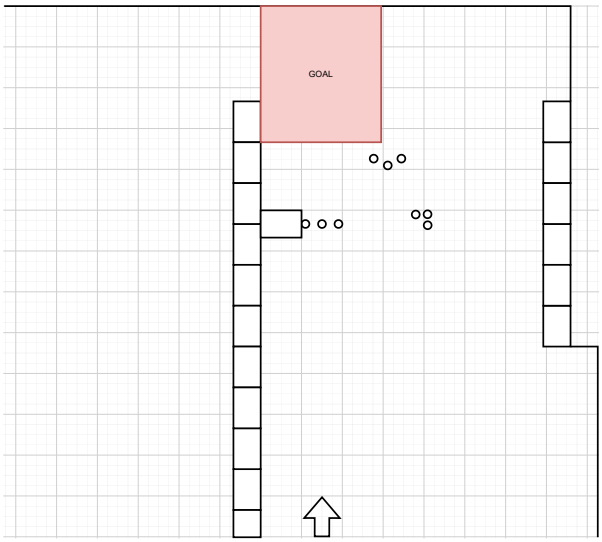

Figure 7.3: Screenshot of benchmarking environment 1.

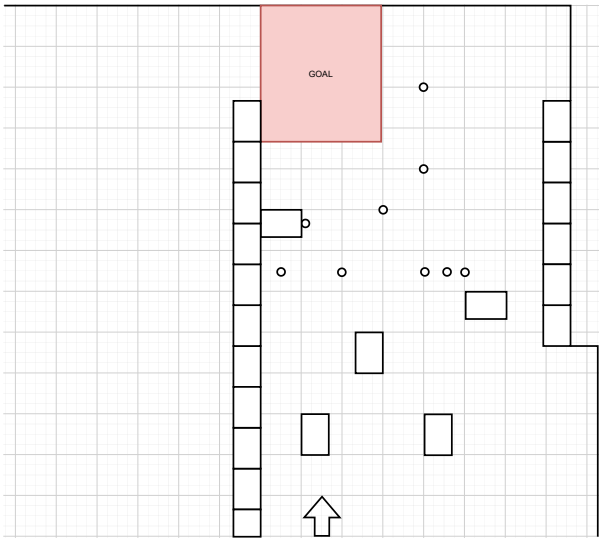

Figure 7.4: Screenshot of benchmarking environment 2.

7.2 Appendix B

Run	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Baseline 1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Baseline 2	1	1	1	0	0	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0
NoMaD 1 slow	0	0	0	1	1	1	0	1	1	1	1	0	0	1	0	1	0	1	0	1	1	0	0	1	1	0	1	1	0	0
NoMaD 1 fast	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	0	1	0	0	0
NoMaD 2 slow	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	1	1	0	0	0	1	0	0	0
NoMaD 2 fast	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
NoMaD as global planner 1 slow	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	0	1	1
NoMaD as global planner 1 fast	1	1	1	1	1	1	0	1	1	1	1	1	0	1	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	0
NoMaD as global planner 2 slow	1	1	1	1	1	1	1	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1
NoMaD as global planner 2 fast	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	0	1	0	1	0	0	1	1	1

Non-exclusive licence to reproduce thesis and make thesis public

I, Robert Allik

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Validation of NoMaD as a Global Planner for Mobile Robots

   supervised by Arun Kumar Singh, Karl Kruusamäe

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Robert Allik
20.05.2024