CHALMERS
UNIVERSITY OF TECHNOLOGY

# A World Model Reinforcement Learning Approach for Vehicle Control

## A vehicle controller based on DreamerV3 and a research platform for training and evaluating machine learning based controllers

Master's thesis in Complex Adaptive Systems

Andreas Munck
Oscar Wir

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se

Master's Thesis in Complex Adaptive Systems

# A World Model Reinforcement Learning Approach for Vehicle Control

A vehicle controller based on DreamerV3 and a research platform for training and evaluating machine learning based controllers

ANDREAS MUNCK
OSCAR WIR


CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
Gothenburg, Sweden 2026

A World Model Reinforcement Learning Approach for Vehicle Control
A vehicle controller based on DreamerV3 and a research platform for training and evaluating machine learning based controllers
ANDREAS MUNCK
OSCAR WIR



Supervisor: Professor Dag Bergsjö, Eira Systems AB
Examiner: Associate Professor Krister Wolff, Department of Mechanics and Maritime Sciences

Master's Thesis 2026
Department of Mechanics and Maritime Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

Cover: A small autonomous off-road vehicle is shown navigating. The upper image uses a motion-trail effect to visualize the full path traveled by the vehicle around the perimeter and highlights its mobility, path-following behaviour, and ability to operate in real-world outdoor environments. The lower images show close-up views of the vehicle on gravel and uneven outdoor terrain.

Typeset in LaTeX
Gothenburg, Sweden 2026

A World Model Reinforcement Learning Approach for Vehicle Control
A vehicle controller based on DreamerV3 and a research platform for training and evaluating machine learning based controllers
ANDREAS MUNCK
OSCAR WIR
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology

# Abstract

The purpose of this thesis was to investigate whether a world-model-based reinforcement learning approach could be used for path following on a small off-road unmanned ground vehicle. The method involved building a 1/10 scale RC car platform equipped with a binocular camera and developing a ROS 2 graph to manage interprocess communication, including pose estimation and vehicle control. A world-model reinforcement learning controller was developed by adapting the DreamerV3 implementation for the path-following task. This controller was evaluated across various trajectories, including sine waves and clothoid turns on uneven grass, as well as backtracking on sand. A Pure Pursuit controller was used as a baseline. However, it was not tuned to individual paths and did not represent state-of-the-art controllers. The results indicate that the controller tracked all paths effectively and mostly smoothly, outperforming Pure Pursuit on all tasks in terms of cross-track error, successfully bridging the sim-to-real gap using a limited amount of training data. While it exhibited minor over-adjustments on certain paths and underutilisation of the steering range, the controller demonstrated an emergent behaviour of reversing to adjust its alignment at sharp corners. Consequently, this validated the successful development of the underlying research platform.

Keywords: World model, Reinforcement learning, DreamerV3, Vehicle controller, Path following, Off-road, ROS 2.

# Preface

This report presents the outcome of our master's thesis project carried out at the Department of Mechanics and Maritime Sciences at Chalmers University of Technology during the spring of 2026. The project was conducted in collaboration with Eira Systems AB and focused on developing and evaluating a world-model reinforcement learning approach for path following with off-road unmanned ground vehicles. The work also included the development of a ROS 2-based research platform for training and evaluating machine-learning-based vehicle controllers.

# Acknowledgements

We would like to thank our examiner, Krister Wolff, for his support throughout the project and for the helpful feedback he provided during the review of this work. We also thank our supervisor, Dag Bergsjö, and Eira Systems AB, who made this project possible and supported us throughout.

Andreas Munck, Oscar Wir, Gothenburg, June 2026

# List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

| | |
|---|---|
| BEV | Bird's-eye view |
| CTE | Cross-track error |
| DC | Direct current |
| ESC | Electronic speed controller |
| GRU | Gated recurrent unit |
| ICC | Instantaneous center of rotation |
| IMU | Inertial measurement unit |
| KL | Kullback–Leibler |
| MCU | Microcontroller unit |
| ML | Machine learning |
| MLP | Multilayer perceptron |
| MPC | Model predictive control |
| PID | Proportional–integral–derivative |
| PP | Pure pursuit |
| PWM | Pulse-width modulation |
| RC | Radio-controlled |
| RGB | Red–green–blue |
| RL | Reinforcement learning |
| RMSE | Root mean squared error |
| RMS | Root mean square |
| RNN | Recurrent neural network |
| ROS | Robot Operating System |
| RSSM | Recurrent state space model |
| SL | Supervised learning |
| UGV | Unmanned ground vehicle |
| VIO | Visual–inertial odometry |
| VSLAM | Visual simultaneous localization and mapping |
| WM | World model |

# Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

## Indices

| | |
|---|---|
| $t$ | Discrete time-step index |
| $i$ | Rollout starting-point index |

## Parameters

| | |
|---|---|
| $k$ | Lookahead gain used by the Pure Pursuit controller |
| $l_{\min}$ | Minimum lookahead distance used by the Pure Pursuit controller |
| $N$ | Number of rollout starting points used in the validation metric |
| $T$ | Finite sequence length used as the rollout horizon |
| $v$ | Reference forward velocity used by the Pure Pursuit baseline |
| $\beta_{\text{rec}}$ | Weight applied to the reconstruction loss |
| $\beta_{\text{dyn}}$ | Weight applied to the dynamics loss |
| $\beta_{\text{rep}}$ | Weight applied to the representation loss |
| $\beta_{\text{move}}$ | Weight applied to the motion-prediction loss |
| $\gamma$ | Discount factor used for return estimation |
| $\lambda$ | Bootstrapping parameter used for return estimation |
| $\lambda_{\text{act}}$ | Weight applied to the action-change penalty |
| $\lambda_{\text{ent}}$ | Entropy regularization coefficient |

# Variables

| | |
|---|---|
| $a_t$ | Control action at time step $t$ |
| $A_t$ | Advantage estimate used for policy optimization |
| $c_t$ | Continuation flag used in return estimation |
| $d_t$ | Planar distance between the vehicle and the target waypoint |
| $\Delta a_t$ | Change in control action between consecutive time steps |
| $\Delta p_t$ | Translational motion increment in the vehicle frame |
| $\Delta q_t$ | Rotational motion increment represented as a quaternion |
| $E_{\text{val}}$ | Open-loop rollout validation error |
| $h_t$ | Deterministic latent state of the world model |
| $\mu_t$ | Mean of the actor action distribution |
| $p_t$ | Vehicle position at time step $t$ |
| $p^{\text{wp}}$ | Target waypoint position |
| $q_t$ | Vehicle orientation represented as a quaternion |
| $R_t$ | Bootstrapped return target |
| $r_t$ | Total reward at time step $t$ |
| $r_t^{\text{goal}}$ | Reward assigned when the target waypoint is reached |
| $r_t^{\text{prog}}$ | Reward based on progress toward the target waypoint |
| $s_t$ | Environment state at time step $t$ |
| $v_x$ | Longitudinal vehicle velocity |
| $x_t$ | Observation at time step $t$ |
| $z_t$ | Stochastic latent state of the world model |
| $\theta_t$ | Relative heading angle to the target waypoint |
| $\theta_t^{\text{norm}}$ | Normalized relative heading angle to the target waypoint |
| $\psi_t$ | Vehicle yaw angle |
| $\sigma_t$ | Standard deviation of the actor action distribution |

# Contents

# List of Figures

# List of Tables

# 1
# Introduction

## 1.1 Background

A human can sit in a car they have never driven before and within minutes adapt to its weight, its steering response, and the grip of the road beneath it. Conventional controllers, by contrast, require either careful manual tuning or an accurate vehicle model to achieve the same result.

Unmanned Ground Vehicles (UGVs) are an enabling technology for operation in unstructured off-road environments, such as search and rescue, agriculture, and exploration. Unlike on-road autonomous driving, off-road driving has no clear structure or predictable surface conditions, requiring the vehicle to constantly adapt to the environment. However, the much lower interaction with other people and vehicles reduces the need for high precision and strict safety requirements.

A vehicle controller for longitudinal and lateral control is an essential part of the autonomous driving stack. It takes desired longitudinal and lateral velocities as inputs and outputs throttle, brake, and steering signals to lower layers of the stack. Conventional methods often require manual tuning. Supervised learning (SL) also struggles with this problem because small errors compound over time, pushing the system into states underrepresented in the training data. Once the system enters such unseen states, the model has no reliable way to recover since SL learns by imitation [1]. Methods exist to mitigate this, but SL is fundamentally limited by the quality and coverage of the expert demonstrations. Since real driving involves an enormous state space, achieving sufficient coverage in practice is very difficult.

The autonomous driving stack can be divided into two parts: path generation and path following. To follow a path, the vehicle must know its position relative to the path. This is solved by the simultaneous localization and mapping (SLAM) module. Once the relative position is known, the controller is responsible for issuing throttle, brake, and steering commands to keep the vehicle on the path.

The conventional controllers such as PID and Pure Pursuit share a fundamental limitation: they are reactive, correcting errors that have already occurred with no prediction of what lies ahead. Model Predictive Control (MPC) solves this by optimizing control over a horizon of future steps, and works for both longitudinal and lateral control. However, it relies on an accurate vehicle model that must be tuned

for each vehicle, which is especially difficult under the changing dynamics of off-road conditions, and solving an optimization problem at every timestep is computationally demanding.

As mentioned, supervised learning is fundamentally limited here since a controller has no single correct answer to imitate. Reinforcement learning (RL) instead learns by interacting with an environment and maximizing the total discounted return. Unlike supervised learning, an RL agent can explore and improve online through its own experience, making it naturally suited to the sequential decision-making structure of lateral control.

One fundamental problem with RL is that training directly on a real vehicle is inefficient. Training in simulation is a natural alternative, but the reality gap, the mismatch between simulated and real vehicle dynamics, means policies that perform well in simulation can fail when deployed on a real vehicle. One solution can be world models, which learn a compact internal model of the environment directly from real data, allowing the agent to train inside its own learned simulation rather than a created one.

## 1.2 Purpose

The purpose of this thesis is to investigate whether a world-model-based reinforcement learning approach can be used for path following on a small off-road unmanned ground vehicle. This is motivated by the difficulty of using conventional vehicle controllers in off-road environments, where vehicle dynamics and terrain interaction can vary and where accurate vehicle models are difficult to derive. In such environments, useful control decisions may depend not only on the vehicle state and pose relative to the path, but also on sensor information about the current and upcoming terrain.

This work therefore studies a controller that can make use of onboard sensor data, including camera images, IMU measurements and odometry estimates. Camera observations can provide information about the ground surface and the terrain ahead, while inertial and odometry measurements describe the vehicle's motion response. The aim is to study whether this sensor-based information can be captured in a learned world model and used to train a reinforcement learning controller through imagined rollouts.

Furthermore, the projects aim is also to develop a research platform using ROS 2 and an RC crawler, integrating necessary components such as visual-inertial odometry, to facilitate the training and testing of the controller in the real world. An application this platform should be able to enable is manual remote control and autonomous backtracking if loosing connection.

In particular the following research questions are being investigated:

1. How accurately can a world model trained on real sensor and control data from a small off-road UGV predict short-horizon vehicle motion across different terrain conditions, and how does prediction accuracy vary between the tested terrains and observation modalities?
2. Can a controller trained through imagined rollouts in the learned world model perform real-world path following when provided with a reference path and a pose estimate?
3. How does the world-model reinforcement learning controller compare with a pure pursuit baseline in terms of cross-track error and ability to handle different path geometries, including curves, oscillating paths, and paths requiring reversing?

## 1.3 Goals

The primary goal is to develop and evaluate a world-model reinforcement learning-based path-following controller that, given a globally planned path and pose estimates from classical estimation methods, performs control to enable robust navigation on off-road terrain. The secondary goal needed for the primary goal is to develope a research platform which enables path following for machine learning based controllers, such as the world model based one.

## 1.4 Scope and Limitations

This thesis focuses on path following for a small-scale off-road UGV using a world-model reinforcement-learning controller and a modular ROS 2-based research platform. The work does not aim to demonstrate a complete autonomous driving system, but rather to evaluate whether a learned controller can follow predefined paths using onboard sensor data and an available pose estimate. The scope is limited as follows:

- **Autonomy stack:** The controller is only intended for pure path following. A reference path is assumed to be available, and the system does not perform global planning, such as selecting a route based on terrain or mission objectives. It also does not perform local planning, meaning that it does not detect obstacles or modify the reference path online to avoid obstacles or terrain hazards. The platform is intended to be modular so that such functions can be added in future work, but they are not implemented or evaluated in this thesis.

- **Localization and mapping:** The thesis assumes that a pose estimate is available from the localization system. SLAM performance, visual–inertial odometry accuracy, map quality, loop closure, drift, and localization robustness are not systematically evaluated. Since the path-following metrics are computed from the estimated vehicle pose rather than from an independent ground-truth measurement, localization errors may affect the absolute reported performance. These unknown errors may limit direct comparison with external results obtained using different localization systems or ground-truth measurement setups.

- **Experimental and evaluation scope:** The experiments were conducted under manageable test conditions, including indoor carpet and off-road surfaces such as grass, large-grained sand, and a forest backtracking course, with sufficient lighting for visual-inertial odometry. More demanding conditions, such as night operation, rain, mud, water, dense vegetation, and steep slopes, were not included in the evaluation. Since the world model was trained from data collected in a limited set of environments, generalization to substantially different terrain and operating conditions was only evaluated to a limited extent. Pure Pursuit is used as a reference baseline, rather than as a comprehensive benchmark against state-of-the-art classical controllers.

- **Platform and transferability:** The controller is evaluated only on one small RC crawler platform with direct command of the steering servo and electronic speed controller through a microcontroller pass-through. The results should therefore not be interpreted as directly transferable to full-scale vehicles, high-speed driving, other vehicles, or systems with production-level actuator abstractions such as ABS, traction control, braking controllers, or other safety layers.

## 1.5 Related work

World models have recently gained attention as a framework for learning control policies in autonomous driving tasks. Several studies have explored this approach in domains such as autonomous racing and simulated urban driving [2, 3, 4, 5, 6]. In these works, latent world models are trained from sensory inputs such as LiDAR or camera observations and used to generate predicted future states for policy learning or planning. Experimental results across both simulated environments and small-scale robotic platforms demonstrate that world-model-based approaches can learn driving behaviors while requiring substantially fewer interactions with the environment. In this work, we investigate the use of an RSSM-based world model framework for path following using visual observations from stereo cameras together with vehicle odometry estimated through classical SLAM, where the world model is utilized for vehicle dynamics prediction in varying terrain.

RC car-based research platforms enabling machine learning-based autonomous driving have been made before [7]. However, these are mainly based on RC cars for driving fast on flat ground, not crawlers, which this work will use. These crawlers enable higher ground clearance and better traversability through rough terrain, though at the cost of a lower speed on flat ground. This research platform also aims to be modular, enabling integration of the path generation and -planning parts of the autonomous stack. The aim is to make the ROS 2 workspace as controller-agnostic as possible.

# 2
# Theory

## 2.1 Conventional controller methods

A PID controller computes a steering command from the cross-track error. This is the perpendicular distance from the vehicle to the desired path. The proportional term reacts to the current error. The integral term corrects small persistent offsets. The derivative term dampens rapid changes to prevent overcorrection:

$$u(t) = K_p e(t) + K_i \int e(\tau) d\tau + K_d \dot{e}(t)$$

where $u(t)$ is the steering command, $e(t)$ is the cross-track error, and $K_p$, $K_i$, $K_d$ are tuned parameters.

A major limitation of the PID controller is its inability to anticipate path curvature. In a curve, the correct steering angle is non-zero even when the vehicle is perfectly on the path ($e = 0$). Because the controller relies on existing error to generate a proportional response, the vehicle must inherently drift outward before corrective steering is applied.

Pure Pursuit addresses this by computing a steering angle toward a lookahead point on the path at distance $l_d$ ahead. It fits a circular arc from the vehicle's current position. This inherently encodes curvature. The controller steers into the turn rather than reacting to drift. The steering angle is given by:

$$\delta = \arctan\left(\frac{2L \sin \alpha}{l_d}\right) \tag{2.1}$$

where $L$ is the wheelbase, $\alpha$ is the heading angle to the lookahead point, and $l_d$ is the lookahead distance. However, $l_d$ has no universal optimal value. A short lookahead tracks curves well but oscillates on straights, while a long lookahead is smooth on straights but cuts corners. Standard Pure Pursuit also has no velocity term, meaning it behaves identically at 10 km/h and 100 km/h.

The main problem with the methods mentioned above is that they are reactive. To solve this a model of the vehicle is needed to find optimal control. The optimal control problem is to find the control signal $u(t)$ that minimizes a cost function $J$ subject to the system dynamics:

$$\min_{u(t)} J = \int_0^T L(x, u)\, dt + V(x(T)) \quad \text{subject to} \quad \dot{x} = f(x, u)$$

where $L(x, u)$ is the running cost penalizing tracking error and control effort, $V(x(T))$ is a terminal cost at the end of the horizon, and $\dot{x} = f(x, u)$ is the dynamics constraint.

One implementation of optimal control is Model Predictive Control (MPC). It discretizes the problem over a finite prediction horizon $N$ and solves at each timestep:

$$\min_{u} \sum_{k=0}^{N-1} l(x_k, u_k) + V(x_N) \quad \text{subject to} \quad x_{k+1} = f(x_k, u_k), \quad u_{\min} \leq u_k \leq u_{\max}$$

where $l(x_k, u_k)$ is a general stage cost penalizing tracking error and control effort, and $V(x_N)$ is the terminal cost. Only the first control input of the optimized sequence is applied. The horizon then shifts forward and the problem is solved again. This gives it the name receding horizon control. This predictive structure allows MPC to anticipate curves and respect physical constraints. It addresses the core limitations of the reactive geometric methods.

### 2.1.1 Kinematic bicycle model

Pure pursuit assumes a 2D kinematic bicycle model, where the two front wheels and back wheels of are merged. The steering is done by the front wheel only.

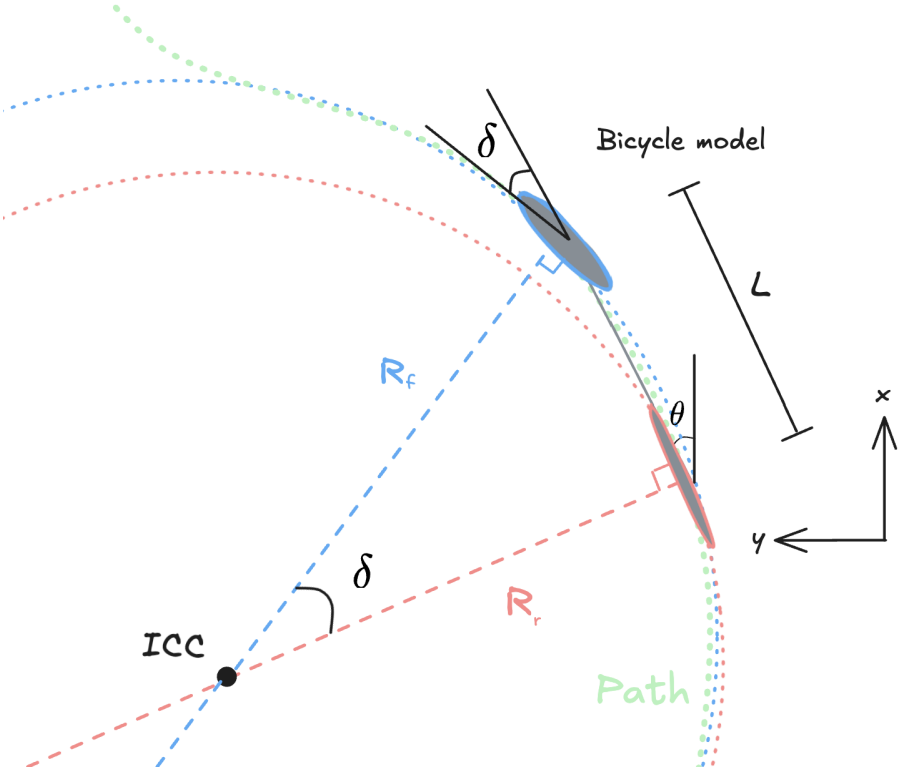



**Figure 2.1:** A kinematic bicycle model following a path in green. Circle of curvature for the front wheel is blue and for the back wheel red. The steering angle is given by $\delta$ and the heading by $\theta$. The global coordinate frame is given in x and y.

From Figure 2.1, the following equations of motion are derived using the Euler approximation:

$$x_{t+1} = x_t + v_t \cdot \cos(\theta_t) \cdot dt$$

$$y_{t+1} = y_t + v_t \cdot \sin(\theta_t) \cdot dt$$

$$\theta_{t+1} = \theta_t + \frac{v_t}{L} \cdot \tan(\delta_t) \cdot dt$$

$$v_{t+1} = v_t + a_t \cdot dt$$

$$\delta_{t+1} = \delta_t + \omega_t \cdot dt$$

where $\delta_t$ is the steering angle, $\omega_t$ the steering rate, $v_t$ the speed, $L$ the wheel base and $a_t$ the linear forward acceleration.

The equation for the vehicle's yaw rate is obtained by observing that the front wheel steering angle $\delta$ forms a triangle with the wheelbase L and $R_r$, the distance from the ICC to the rear axle.

$$\tan(\delta) = \frac{L}{R_r} \implies R_r = \frac{L}{\tan(\delta)}$$

The relationship between the vehicle's linear velocity and yaw rate $\dot{\theta}$ is:

$$v = \dot{\theta} \cdot R_r \implies \dot{\theta} = \frac{v}{R_r}$$

Substituting $R_r = \frac{L}{\tan(\delta)}$ into the equation yields the yaw rate:

$$\dot{\theta} = \frac{v}{\frac{L}{\tan(\delta)}} = \frac{v \cdot \tan(\delta)}{L}.$$

### 2.1.2 Pure Pursuit

#### 2.1.2.1 History

Pure pursuit was developed by Coulter [8] by modeling human driving behavior. A human driver looks at a point further up the road and steers towards it. This allows the pure pursuit controller to anticipate the path rather than being purely reactive. The underlying mathematical concept of "pure pursuit"—following a target by continually pointing the velocity vector toward it, is much older. Missile algorithms utilized this principle to strike moving targets in the 1900s. It was first formalized in the 1700s by observing how pirate ships intercept targets [9, 10].

#### 2.1.2.2 Algorithm

The core concept of the algorithm is to steer towards a point further up the path. A goal point $(g_x, g_y)$ is selected at an Euclidean lookahead distance $l$ from the origin, which is placed on the rear axle. A circular arc connects the rear axle to the goal point. To reach this goal point, the vehicle must travel along a circular arc. According to the kinematic bicycle model, the radius $R_g$ from the goal point to the ICC must equal the rear axle turning radius $R_r$. The steering angle $\delta$ is therefore chosen to satisfy $R_g = R_r$. See Figure 2.2 for illustration.

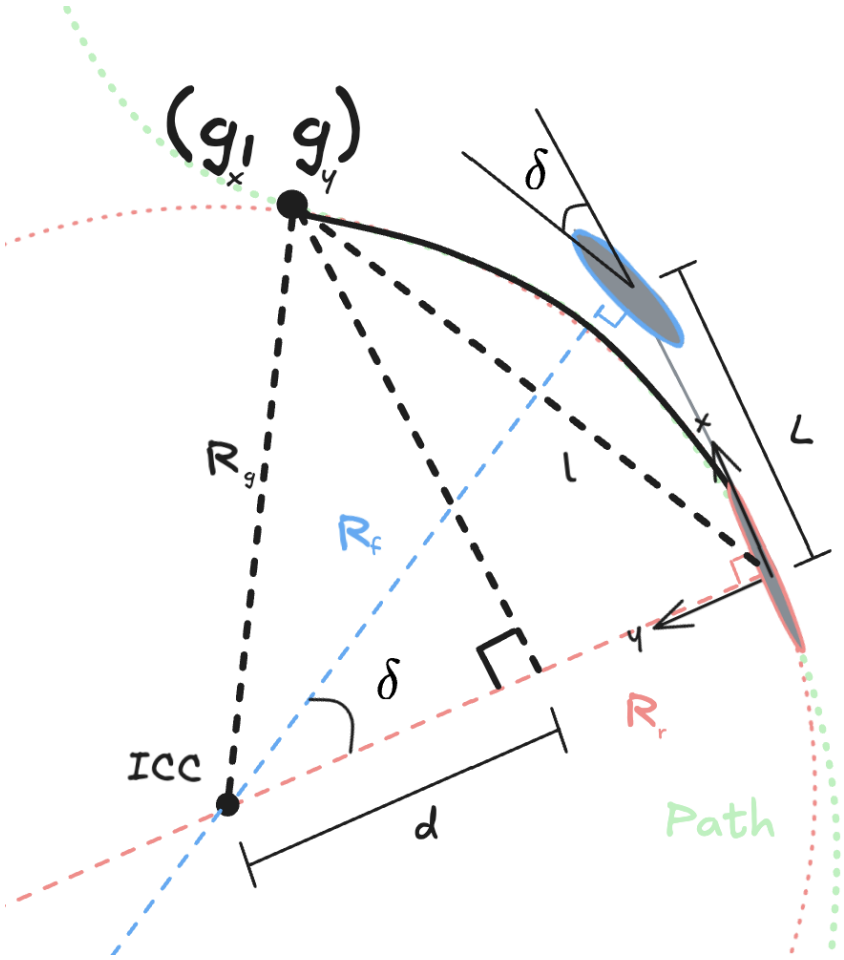



**Figure 2.2:** Illustration for explaining the Pure Pursuit algorithm. Here the origin of the coordinate frame is placed on the rear axis.

The origin is placed at the center of the rear axle of the kinematic bicycle model. The x-axis is aligned parallel to the vehicle's longitudinal direction of travel.
Let R denote the matching radius where $R_r = R_g$. The radius R can be expressed using the goal point and lookahead distance l. Following the geometry in Figure 2.2, the derivation for Eq 2.1 is as follows.

$$R = g_y + d$$

From the Pythagorean theorem, the lookahead distance and radius relate as:

$$l^2 = g_x^2 + g_y^2$$

and

$$R^2 = d^2 + g_x^2$$

Substituting $d = R - g_y$ into the radius equation yields:

$$R^2 = (R - g_y)^2 + g_x^2 = R^2 - 2Rg_y + (g_y^2 + g_x^2)$$
$$\Rightarrow 2Rg_y = (g_x^2 + g_y^2) = l^2 \Rightarrow$$

$$\Rightarrow R = \frac{l^2}{2g_y}$$

The relationship between the turning radius, wheelbase L, and steering angle $\delta$ is:

$$\tan(\delta) = \frac{L}{R}$$

Inserting the derived expression for R gives the final steering angle equation for the Pure Pursuit algorithm:

$$\delta = \arctan(\frac{L2g_y}{l^2}) \tag{2.2}$$

#### 2.1.2.3 Determining lookahead distance

From the Pure Pursuit steering Eq 2.2, the relationship between the steering angle $\delta$ and the lookahead distance $l$ is:

$$\tan(\delta) = \frac{2Lg_y}{l^2} \tag{2.3}$$

For small steering angles where $\tan(\delta) \approx \delta$, the response is proportional to the inverse square of the lookahead distance ($\delta \propto l^{-2}$). A short lookahead distance therefore acts as a high controller gain and produces aggressive steering. It forces the vehicle to correct tracking errors quickly. However, physical steering rate limits often prevent the wheels from straightening fast enough as the vehicle reaches the path. This delay causes overshooting and oscillations. Conversely, a long lookahead distance provides smoother steering but results in much slower path convergence.

#### 2.1.2.4 Adaptive pure pursuit

To prevent the vehicle from making overly sharp turns at high speeds, the lookahead distance can be made proportional to the vehicle speed [11].

$$l = kv$$

where $k$ is a gain parameter with the dimension of time. It represents the time required to reach the goal point if the vehicle were to drive straight toward it.

The fundamental tuning problem remains. A low $k$ results in oscillating behavior. A large $k$ results in cutting corners. The gain can be tuned empirically by choosing a small initial value and increasing it until oscillations disappear [12]. To prevent the lookahead distance from becoming too short at low speeds, the adaptive distance is often restricted by a lower bound.

## 2.2 Reinforcement learning

Reinforcement learning (RL) is a learning framework where an agent learns by trial-and-error, through interaction with an environment, to select actions that maximize long-term cumulative reward [13].

### 2.2.1 The RL problem

#### 2.2.1.1 Problem formulation

The RL problem can be mathematically formulated as a Markov decision process (MDP). Here the agent takes an action $A_t$ and receives a new state $S_{t+1}$ and reward $R_{t+1}$ from the environment. Figure 2.3 illustrates this process [13].

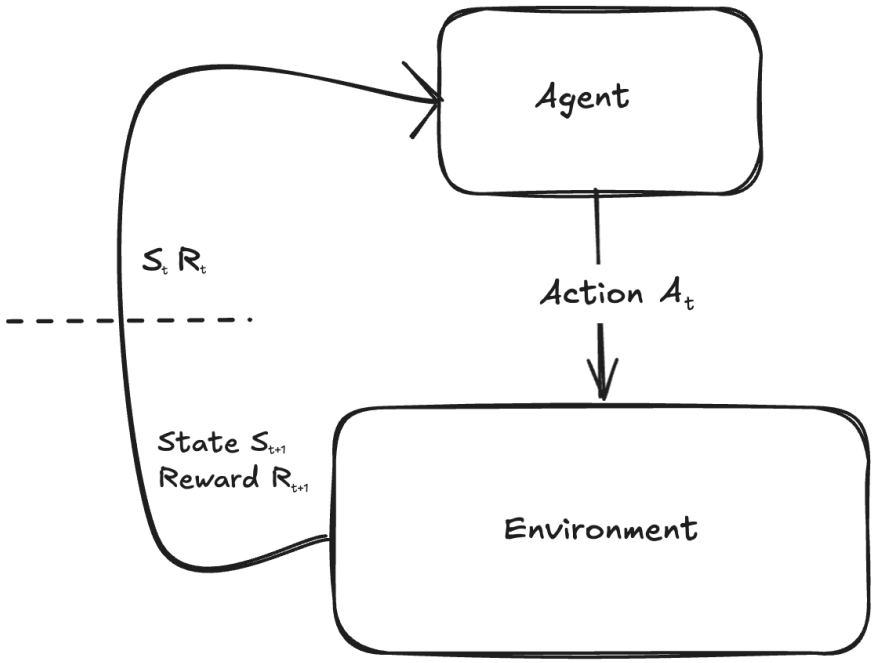



**Figure 2.3:** A Markov decision process.

The MDP formulation assumes the Markov property. The next state $S_{t+1}$ and reward $R_{t+1}$ depend only on the current state $S_t$ and action $A_t$. They do not depend on the full history of prior states and actions, as the current state accounts for them [13].

A short note on syntax: capital letters $S, A, R$ denote stochastic variables, while the corresponding lower-case letters $s, a, r$ denote specific values they can take. For value functions, $v_\pi$ and $q_\pi$ denote the true value functions, while capital $V$ and $Q$ denote their learned estimates.

The goal is to maximize the discounted total accumulated return $G_t$ defined as:

$$G_t \coloneqq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{2.4}$$

where $\gamma$ is the discounting factor [13, p. 57]. If the process, also called an episode, has an end the definition is:

$$G_t \coloneqq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{T} \gamma^k R_{t+k+1}, \tag{2.5}$$

where $T$ is the episode length [13, p. 55]. The discounted return is chosen over just the cumulative return to

1. Make the sum in Eq 2.4 converge for $\gamma < 1$.
2. Model that future rewards are more uncertain than immediate ones.
3. Control the agents planning time horizon. A small $\gamma$ results in a shortsighted greedy agent, while a large $\gamma$ results in farsighted planning agent.

To maximize $G_t$ the agent needs to act in a the best possible way. Which action the agent choses given a certain state is determined by the agents policy $\pi$.

$$\begin{cases} a \sim \pi(s), & \text{if stochastic,} \\ a = \pi(s), & \text{if deterministic.} \end{cases} \tag{2.6}$$

The goal can now be expressed as

$$\text{GOAL:} \quad \max_{\pi} \mathbb{E}[G_t]. \tag{2.7}$$

$G_t$ is a goal and not a tool for achieving it. It is known first when the episode is finished. Therefore $G_t$ needs to be estimated. The value functions do this.

#### 2.2.1.2 Value functions

The state value function $v_\pi(s)$ answers the question "how good is state $s$?".

$$v_\pi(s) \coloneqq \mathbb{E}_\pi[G_t \mid S_t = s] \tag{2.8}$$

The action value function $q_\pi(a, s)$ answers the question "how good is it to do $a$ in state $s$?" [13, p. 58].

$$q_\pi(s, a) \coloneqq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \tag{2.9}$$

To achieve the goal from Eq 2.7, an optimal policy $\pi_*$ needs to be found. This can be done by finding optimal value functions which are defined as:

$$v_{\pi_*}(s) \geq v_\pi(s) \quad ,\forall\ s \in \mathcal{S} \text{ and } \forall\ \pi, \tag{2.10}$$

$$q_{\pi_*}(s,a) \geq q_\pi(s,a) \quad ,\forall\ s \in \mathcal{S},\ a \in \mathcal{A},\ \text{and } \forall\ \pi. \tag{2.11}$$

where $\mathcal{S}$ and $\mathcal{A}$ is the set of all possible states and actions respectively [13, pp. 62–63].

#### 2.2.1.3 Bellman equations

The Bellman equations find these by using dynamic programming. They express the value of the state as the sum of the reward of that state and the value of the next state.

$$v_\pi = r + \gamma v_\pi(s')$$

where $s'$ is the next state.

To form the Bellman equations the following relations are used:

$$v_\pi(s) = \sum_a \pi(a \mid s) q_\pi(s,a) \tag{2.12}$$

since the state value is just the action value weighted by the probability of taking each action under the policy.

$$q_\pi(s,a) = \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_\pi(s')] \tag{2.13}$$

since the value of taking action $a$ in state $s$ is the expected immediate reward plus the discounted value of the next state, averaged over all transitions $(s', r)$ under the dynamics $p$ .

Now substituting Eq 2.13 into Eq 2.12 forms the Bellman equation for the state value and substituting Eq 2.12 into Eq 2.13 forms the Bellman equation for the action state value [13, pp. 63–64].

#### 2.2.1.4 General policy iteration

The general method for finding the optimal state value function and policy is general policy iteration (GPI). This process evaluates a policy to get a state value function. It then uses that value function to improve the policy. The implementations of evaluation and improvement differ between algorithms [13, p. 86]. In the dynamic programming case using the Bellman equations explicitly, the policy evaluation updates the state value function according to the Bellman equation for the state value function.

$$V(s) \leftarrow \sum_{a \in \mathcal{A}(s)} \pi(a|s) \sum_{\substack{s' \in \mathcal{S} \\ r \in \mathcal{R}}} p(s', r|s, a)\,[r + \gamma V(s')] \tag{2.14}$$

The policy improvement is made by computing the action state value function using Eq 2.13 and taking the arg max over it.

$$\pi(s) = \arg\max_a Q(s, a). \tag{2.15}$$

During training of the agent Eq 2.15 is often not a great choice since it is maximally exploitative, also called greedy. To make the agent not only chose the actions it this far thinks are the best, but to also make it explore more, one solution is improving the policy by choosing a random action with a small probability $\epsilon$ and otherwise improve according to Eq 2.15.

#### 2.2.1.5 Model-free or model-based

A model in the context of model-based or model-free RL is a tool the agent uses to predict the response from the environment based on its actions. This is an estimation of $p(s', r|s, a)$ [13, p. 7].

Therefore, while model-based algorithms learn to plan actions, model-free algorithms learn reward-action associations directly. A world model is an example of a model-based RL [2].

### 2.2.2 Approximations of expected accumulated return

Since $G_t$ is a very long or infinite sum and the state- and action-spaces can be continuous or very large, the expectation values in Eqs 2.9 and 2.8 are sometimes impossible to calculate. Therefore different methods are used to approximate them. One simple approach is to keep averages of the discounted returns that have followed from being in a certain state and taking a certain action in that state, the average will converge by the law of large numbers to the expectation value. This forms a family of methods called Monte Carlo (MC) methods. One update rule for MC method is

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(g_t - Q(s_t, a_t)\right) \tag{2.16}$$

where $\alpha$ is a constant step size [13].

However MC methods requires that a episode finishes before evaluation. A solution to this is using temporal difference learning (TD), which looks $n$ steps into the future and then estimates the rest with the state value function. $g_t$ is thereby replaced by the $n$-step TD target:

$$g_{t:t+n} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n}). \tag{2.17}$$

Using a target in an update function based on an approximation is called *bootstrapping* . The $n$ parameter can be used as a bias-variance-tradeoff parameter, since

larger $n$ uses more rewards and less estimations and therefore lowers bias, but increase the variance because more noisy transitions which accumulate [13].

However when not in the tabular case, that is $\mathcal{S}$ and $\mathcal{A}$ are not discrete and too large to be listed as state-action pairs. The form of the update functions needs to change.

### 2.2.3 Linear function approximation

When not in the tabular case approximation of $v_\pi$ and/or $q_\pi$ are made with parametrized functions, with fewer parameters than possible states [13].

The parameters $\mathbf{w}$ are chosen to minimize the value error $\bar{VE}$.

$$\bar{VE}(\mathbf{w}) = \sum_{s \in \mathcal{S}} \mu(s) \left(v_\pi(s) - \hat{v}_\pi(s, \mathbf{w})\right)^2 \tag{2.18}$$

where $\mu$ is a weight distribution often chosen to be how often a state is visited.

The $\bar{VE}$ cannot be computed since $v_\pi(s)$ is not known. Therefore a gradient descent is made with a relevant target $U_t$. Two options for $U_t$ are the MC and TD targets mentioned above.

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[U_t - \hat{v}(S_t, \mathbf{w})\right] \nabla \hat{v}(S_t, \mathbf{w}) \tag{2.19}$$

Some targets, such as the TD target Eq 2.17 $U_t$ depends on the weights, which results in the target changing when updating, making it not a true gradient descent step. It is known as semi-gradient descent [13, p. 202].

#### 2.2.3.1 $\lambda$-return

One target that combines MC and TD and enables continuous bias-variance trade-off tuning with a parameter $\lambda$ is the the $\lambda$-return [13, pp. 288–292].

$$G_t^\lambda := (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_{t:T} \tag{2.20}$$

where $G_{t:t+n}$ is the TD target from Eq 2.17.

### 2.2.4 Deep RL

Replacing the linear functions with deep neural networks forms deep RL. Further, instead of predicting weights of the value function, a parametrized policy could be used and skip the value functions. Methods utilizing this are called policy gradient methods (PGM) [14].

#### 2.2.4.1 REINFORCE

The goal is to perform some sort of gradient ascent with a update function on the form of

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{2.21}$$

where $\boldsymbol{\theta}$ are the parameters and $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is the policy gradient. A naive choice is to take the gradient of the policy to find the step direction. The gradient of the policy indicates which direction increases or decreases the probability of choosing an action. This gradient is weighted with $g_t$ to determine the step size in that direction [13].

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha g_t \nabla_{\boldsymbol{\theta}} \pi(a_t \mid s_t; \boldsymbol{\theta}) \tag{2.22}$$

A problem with Eq 2.22 is that an action's update grows with how often that action is already selected. Frequently chosen actions therefore accumulate larger updates and can win out even when their return $g_t$ is low. Dividing by $\pi(a_t \mid s_t; \boldsymbol{\theta})$ corrects for this sampling frequency. The function is updated to

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha g_t \frac{\nabla_{\boldsymbol{\theta}} \pi(a_t \mid s_t; \boldsymbol{\theta})}{\pi(a_t \mid s_t; \boldsymbol{\theta})} \tag{2.23}$$

which is equivalent to

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha g_t \nabla_{\boldsymbol{\theta}} \ln[\pi(a_t \mid s_t; \boldsymbol{\theta})] \tag{2.24}$$

A problem with Eq 2.24 is that if all rewards are positive and large, no negative updates are made to bad actions. To combat this the concept *baseline* is introduced.

Baselines are introduced to compare how good an action is compared to some function, called the baseline, which explains the average reward from the actions in a certain state. The baseline function $b(s_t)$ can be almost any function which does not depend on an action [14].

A natural choice of $b(s_t)$ is $v_\pi(s_t)$, because this results in the comparison between how good an action is and how good it is to be in that state. $v_\pi(s_t)$ can be estimated by a neural network with parameters $\phi$, $V_\phi(s_t)$, and with the objective function

$$\phi = \arg\min_{\phi} \mathop{\mathbb{E}}_{s_t, \hat{R}_t \sim \pi} \left[ \left( V_\phi(s_t) - \hat{R}_t \right)^2 \right], \tag{2.25}$$

where $\hat{R}_t$ is a reward function, often chosen as rewards-to-go [14].

$$\hat{R}_t \coloneqq \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}), \tag{2.26}$$

This results in the update equation for the fundamental deep RL algorithm, called REINFORCE with baseline or vanilla policy gradient (VPG) [14].

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha (g_t - V_\phi(s_t)) \nabla_{\boldsymbol{\theta}} \ln[\pi(a_t \mid s_t; \boldsymbol{\theta})] \tag{2.27}$$

Another name for the baseline is the critic when it estimates the value function [13, p. 321]. It critiques the actions of the policy neural network, which is called the actor.

The actor networks often predicts a categorical policy for discrete tasks and a diagonal Gaussian for continuous action spaces [14].

#### 2.2.4.2 Entropy

When an actor learns, it moves its action distribution from a randomly initialized, most often uniform, distribution towards one with more concentrated probability mass. The problem is that an overly concentrated policy may converge prematurely to a suboptimal local minimum before the action space has been sufficiently explored. Moving away from a uniform distribution reduces the entropy. Therefore, a term that penalizes low entropy is added to the actor loss, encouraging exploration [15]. The entropy $H$ has the form

$$H[\pi(a \mid s)] = -\sum_{a} \pi(a \mid s) \ln \pi(a \mid s). \tag{2.28}$$

## 2.3 World models

World models, as introduced by Ha and Schmidhuber [2], are neural networks designed to learn compact and predictive representations of an environment. These models are motivated by theories in neuroscience suggesting that the brain constructs internal models of the world that enable prediction and shape perception. In particular, this idea is closely related to predictive coding, a theory proposing that biological systems continuously anticipate future sensory inputs and reduce prediction error by comparing expected sensory feedback with actual observations, conditioned on past observations and executed actions [2][16]. In this context, a world model can be understood as a computational model that predicts future sensory observations from previous observations and actions.

World models give an RL agent a memory through the hidden state of an RNN. This can be used to predict the future and therefore reduce the need for trial and error by the RL agent. A major problem in RL is credit assignment, which is assigning actions to their long-term consequences. Training larger neural networks to learn a good policy is therefore harder. Therefore, it is suitable to have a separate larger model, such as the RNN, to encode the information of the world, leaving the agent's network smaller. Aside from the memory part and the RL controller part, a vision model consisting of a CNN encoder and decoder is used to learn and compress visual information [2].

### 2.3.1 Recurrent state space model

The Recurrent State Space Model (RSSM) is a latent dynamics model used in world model approaches such as DreamerV3. The latent state consists of a deterministic

hidden state $h_t$ and a stochastic variable $z_t$. The deterministic component captures temporal dependencies, while the stochastic component represents uncertainty and allows modeling multiple possible futures. This hybrid structure overcomes the limitations of purely deterministic or purely stochastic models [17, 18].

The transition dynamics predict a prior distribution over the stochastic latent state:

$$h_t = f(h_{t-1}, z_{t-1}, a_{t-1}), \tag{2.29}$$
$$z_t \sim p(z_t \mid h_t), \tag{2.30}$$

where $a_{t-1}$ is the previous action and $f$ is typically implemented as a recurrent neural network such as a GRU.

An encoder infers the posterior distribution over the stochastic latent state using the current observation $x_t$:

$$z_t \sim q(z_t \mid h_t, x_t). \tag{2.31}$$

Observations are predicted from the full latent state via an observation model (decoder) that grounds the latent space:

$$x_t \sim p(x_t \mid h_t, z_t). \tag{2.32}$$

# 3
# Methods

## 3.1 Research platform

To evaluate the controllers, a small UGV platform was developed by modifying a 1/10-scale RC crawler, the Redcat Gen 8 V2 International Scout II [19]. A binocular camera with an integrated IMU was used for localization, and the software was developed within a ROS 2 [20] graph to accommodate modularity.

### 3.1.1 Hardware

The hardware of the research platform was developed to be able to handle rollovers at the maximum speed of the Redcat crawler, estimated to be around 4 m/s. It also needed to be enclosed around the computing unit to keep objects and dirt off the electronics.

#### 3.1.1.1 Bill of materials

The main components used to build the hardware are listed in Table 3.1.

**Table 3.1:** Bill of material for research platform.

| Purpose | Component |
|---|---|
| Chassi | Redcat crawler original |
| DC Motor | Redcat crawler original |
| ESC | Redcat crawler original |
| Steering servo | Redcat crawler original |
| Computer | Nvidia Jetson Orin Nano |
| MCU | Arduino Nano |
| Battery | 3s 7000mAh - 60C - Gens Ace Bashing G-Tech EC5 |
| Binocular camera | Intel Realsense D455 |
| Shell | PLA |

#### 3.1.1.2 Design

The electronics were kept simple, with one battery powering the computer, the steering servo, and the motor. An MCU was used to feed signals through to the ESC because it natively handles PWM signals and the OS scheduling of the main computer was thought to create a delay. A wiring schematic can be found in Figure 3.1.

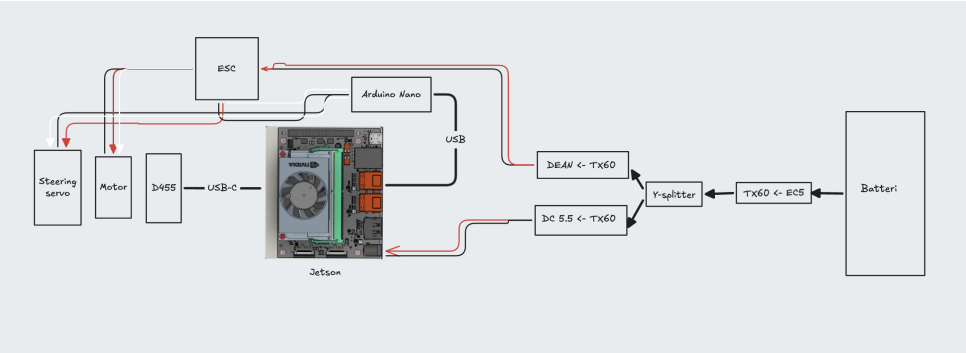



**Figure 3.1:** Wiring schematic

**Figure 3.2:** The UGV platform.

The existing frame holding the ESC, DC motor, steering servo, and the radio receiver (not used) was kept. The two pillars were fastened to this frame, and on these pillars a baseplate was placed, accommodating the computer. Another plate was placed on this baseplate, pointing forward, to hold the binocular camera and the MCU. The shell was then built in three pieces: one covering the front plate, one the middle, and one simply made to enclose the shell and create a more even weight distribution.

It was also designed to include a fan creating positive pressure inside the shell to keep dust out and the temperature down. But it was found not to be needed, as next

to no dust gathered inside the shell and the computer's temperature was kept low, possibly because of the large air volume inside the shell. A picture of the platform can be observed in Figure 3.2.

### 3.1.2 Software

The software was built in three parts: the firmware on the MCU, the ROS 2 workspace on the Jetson computer and a data pipeline to convert the ROS 2 bags into a suitable format. The Isaac ROS wrapper for ROS 2 is used, together with ROS 2 Humble, the latest distribution supported for the Nvidia Jetson Orin.

#### 3.1.2.1 Firmware

The Arduino received serialized packets over USB from the Jetson. Each packet contained an ID, specifying whether it targeted the steering servo or the DC motor, and a PWM value, which was sent to the digital pins using the Servo library. Hard-coded PWM clamps were applied during early testing. A watchdog timer tracked the time since the last received packet and sent a stop signal to the motor if none arrived, guarding against a computer shutdown or lost connection while driving.

#### 3.1.2.2 Localization and mapping

For localization and mapping, the Isaac ROS VSLAM package was used, which implements the CUDA-accelerated cuVSLAM algorithm [21], making it well suited to the Jetson. The wrapper also supported the Intel Realsense camera. cuVSLAM used its two infrared binocular cameras together with its IMU.

The setup was based on the example launch file provided by Nvidia [22]. However, their default parameters gave poor performance and dropped frames, which in turn made the odometry jump. Through trial and error, the camera frame rate was lowered from 90 to 30 Hz to mitigate the dropped frames. This was motivated by more IMU readings occurring between frames, making the sensor fusion less reliant on the visual data, and by the bandwidth this freed up. Auto exposure was also turned off, as the visual odometry could not handle it reliably and it correlated with frame dropping. To match the lower frame rate, the `image_jitter_threshold_ms` parameter was increased to 35.0 ms, allowing for the maximum 2 ms jitter recommended by Nvidia [23]. Aside from this, and the RGB camera running at 30 Hz at a resolution of 424x240, the parameters from [22] were used unchanged.
However, the visual odometry remained sensitive to moving objects and occasionally jumped, as seen in Figure C.1, and dropped frames still occurred. Potential further fixes include calibrating the IMU and adding a rear-facing camera.

#### 3.1.2.3 ROS 2 workspace

The ROS 2 workspace is divided into four main functions: Observations, Paths, Control, and Driver. Each function comprises one or more packages, which in turn

contain one or more nodes. Figure 3.3 illustrates a simplified version of the ROS 2 graph.

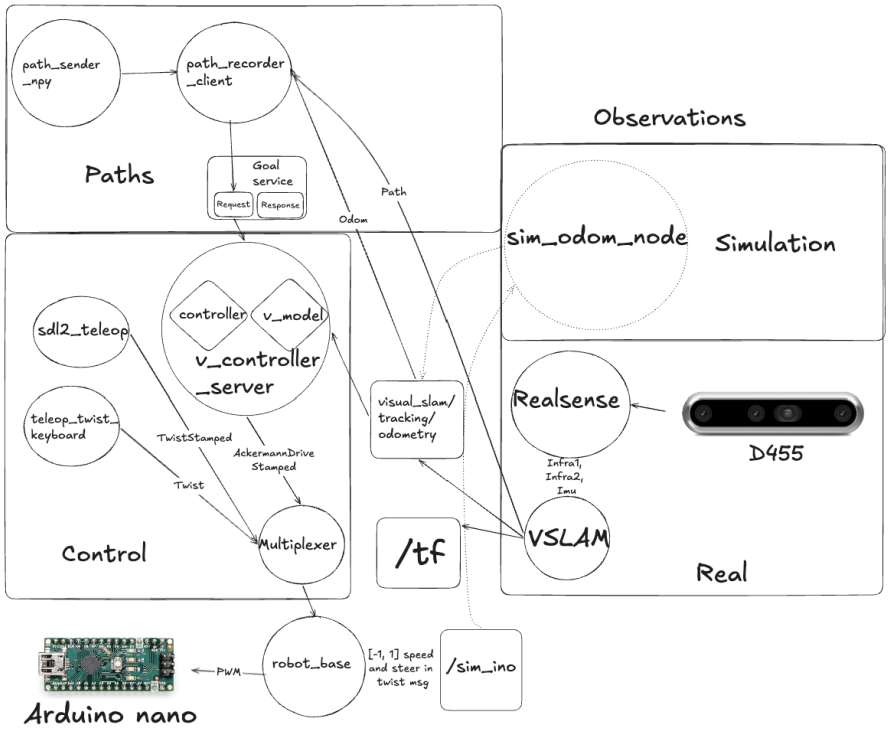



**Figure 3.3:** Simplified ROS 2 graph used in the project. Dotted nodes and connections are used only in simulation. Observations come either from a bicycle-model simulation node or, in reality, from the VSLAM and camera nodes. Paths are loaded from NumPy files, recorded from odometry, or taken directly from the VSLAM node, then sent to the vehicle controller server, which outputs a driving command. The vehicle can also be driven manually from an SDL2-compatible controller or keyboard. Through the multiplexer node, these override the controller commands. The selected command is sent to the robot base node, which packages it for the MCU and, in simulation mode, feeds the steering back to the simulation node.

**3.1.2.3.1 Observations** The cuVSLAM node subscribes to the inertial and binocular topics from the Intel Realsense wrapper node [24] and publishes an odometry message and the dynamic tf transforms `map->odom` and `odom -> <suitable static frame>`. In this project `base_link`, placed at the center of the rear axle. The node can also collect a history of recent poses, transform them into the latest map frame, and expose them through the service `/visual_slam/get_all_poses`.

To test features without access to the camera, Jetson, or vehicle, a simple test node was implemented, see Section C.1 for more.

**3.1.2.3.2 Paths** To follow paths, a ROS 2 action is implemented, with the path recorder node as the action client and the controller server node as the action server.

The action message is a custom version of `nav2_msgs/action/FollowPath` with an added boolean for reverse, see Appendix C.1.2.

The path recorder can record a path from the tf topic, triggered by a service, which enables backtracking. To obtain the full path in the latest map frame, `/visual_slam/get_all_poses` can be called instead. The node can also receive an external path from the path sender node, which reads a NumPy file and sends a Path message through a custom service [25] (Listing C.2). On a trigger service signal, the path recorder client sends the path to the controller server within an action.

**3.1.2.3.3 Control** The controller server is implemented as a single-threaded executor node, separated from the multi-threaded executor container holding the path recorder, multiplexer, and robot base nodes. Separating it simplifies thread management. The controller node allocates a dedicated worker thread to run the control loop, preventing interruptions from incoming service requests and topic callbacks. The control loop runs at the image data frequency, 30 Hz, and the color image and odometry topics are synchronized using the ROS 2 message filter library.

Before entering the control loop, the worker thread transforms the path into the latest map frame. Each iteration, it fetches the latest synced data from the message filter and updates the path window sent to the controller. A window is used because the controller works in either the base or odom frame, and transforming a long path to that frame every iteration would be costly. When the controller uses the base link, as in this project, the odometry only provides the timestamp used to sync the map-to-base transformation with the images.

The path window is updated by calling a controller-plugin method that returns the new start index for the window, allowing different path handling per controller. For instance, the PP plugin searches for the closest point in the path window: it needs this point for its algorithm, and going back to a missed point is undesirable, since it would make the vehicle take a large-radius turn and struggle to recover to the path. The WM-RL plugin, by contrast, checks whether it is within a distance parameter of the first point in the window and, if so, pops that point by incrementing the index. This more waypoint-style handling is possible because WM-RL recovers easily from a missed waypoint.

Keeping the full path in the map frame rather than the odometry frame made it necessary to handle map jumps. Such jumps should only arise from SLAM loop closures, but since the odometry tended to jump on dropped frames or moving objects, they occurred more frequently and were more severe. This, together with the drift that is hard to manage over longer paths, is why the path was not simply stored in the odometry frame. To handle jumps in the map frame the path was re-transformed into the latest map frame if a too large jump or twist had been made, see Section C.1.1.

The controller server was designed to be controller-agnostic, allowing seamless switching between models. This was achieved with the ROS 2 plugin architecture: a base class is compiled into the controller server node at build time, while the specific plugin inheriting from it is loaded dynamically at runtime, so controllers can be switched without rebuilding the package. The idea to use this for the controller comes from Nav2 [26], and the base class implementation is similar to theirs.

A multiplexer node enabled seamless switching of the control input between a keyboard, an SDL2-compatible [27] controller, and the controller server. Manual steering did not cancel the goal sent to the controller server. A separate cancel service had to be called for that, enabling intervention or guiding during a trajectory. This was not used in any of the runs in this report.

Building on the existing Nav2 framework was considered, but integrating and evaluating the WM-RL controller within it was expected to be harder, especially given limited prior ROS 2 experience. Building from scratch gave more control and deeper familiarity with ROS 2, enabling solutions such as the map-jump handling above.

#### 3.1.2.4 Data pipeline

During runs, data were recorded to train the world model and later evaluate the performance of both controllers. The data were recorded using ROS 2 bags. These needed to be converted into a suitable format for ML training and data visualization. ROS 2 bags are not suitable since they are sequential. The bags were therefore converted into NumPy arrays. The pipeline can be divided into three subparts: extraction, synchronization, and processing, each with its own challenges.

The extraction was done utilizing the Rosbags Python library [28]. The image data meant that the data would be too large to keep in RAM. Therefore NumPy's memory-mapped files were used, reading and writing directly to the disk. The images were downscaled to 64x64 pixels, so the size were small enough to use NumPy arrays over compressed image files for faster loading during training.

The synchronizing was important since the messages are not guaranteed to be recorded by the ROS 2 bag at the same time. The python library Pandas method `merge_asof` was used to synchronize on the timestamps of the messages, not the timestamps when they were recorded into the ROS 2 bag. The color images were selected as anchor so the other topics were synchronized to it.

The processing included calculating the movement deltas required by the world model. These were then used to filter out parts of the rollouts which included unnaturally large jumps in movement deltas, thereby splitting the rollout also.

## 3.2 WM-RL controller

This section describes the implemented control approach for path following. The task is to generate control commands that allow the vehicle to follow a path composed by a sequence of waypoints while accounting for varying vehicle dynamics. To address this, a model-based reinforcement learning approach is used. A learned world model is first trained in a self-supervised manner to capture the vehicle dynamics from real-world data. This model is then used as a simulator in which a policy is trained using reinforcement learning. By training the policy within the learned model, large amounts of synthetic interaction data can be generated efficiently, reducing reliance on policy interactions with the real world. The resulting policy acts as a controller that maps the current vehicle state and waypoint information to control actions.

### 3.2.1 World model implementation and training

A World model (WM) is utilized to predict the vehicle dynamics resulting from control actions and environmental influences, see Figure 3.4. The WM, implemented by an RSSM that is outlined in section 2.3.1, encodes observations $x_t$ consisting of RGB-images, visual-odometry and IMU readings. The images contain information about the ground surface, obstacles, and spatial context which are relevant for predicting the vehicle's current and future movement behavior. The observations are encoded into a stochastic discrete latent representation. Following DreamerV3 [18], the encoder maps each observation to a posterior distribution over discrete latent variables, from which a set of one-hot vectors is sampled using the straight-through gradient estimator. The resulting stochastic latent representation captures uncertainty in the observations and promotes richer representation learning. The encoded observation $z_{t-1}$, together with an action $a_{t-1} \in [-1, 1]^2$ representing throttle and steering, is provided as input to a GRU sequence model that produces a deterministic latent state $h_t$ summarizing the history of past observations and actions.

The latent state $[h_t, z_t]$ is used by the following model components:

- A dynamics model predicts the prior distribution over the stochastic latent state:

$$z_t \sim p(z_t \mid h_t). \tag{3.1}$$

- A decoder reconstructs the observation:

$$x_t \sim p(x_t \mid h_t, z_t). \tag{3.2}$$

- An auxiliary movement predictor estimates the relative vehicle motion:

$$(\delta d_t, \delta q_t) \sim p(\delta d_t, \delta q_t \mid h_t, z_t), \tag{3.3}$$

where $\delta d_t = [\delta x_t, \delta y_t, \delta z_t]$ denotes translational displacement and $\delta q_t$ denotes rotational change represented as a quaternion.

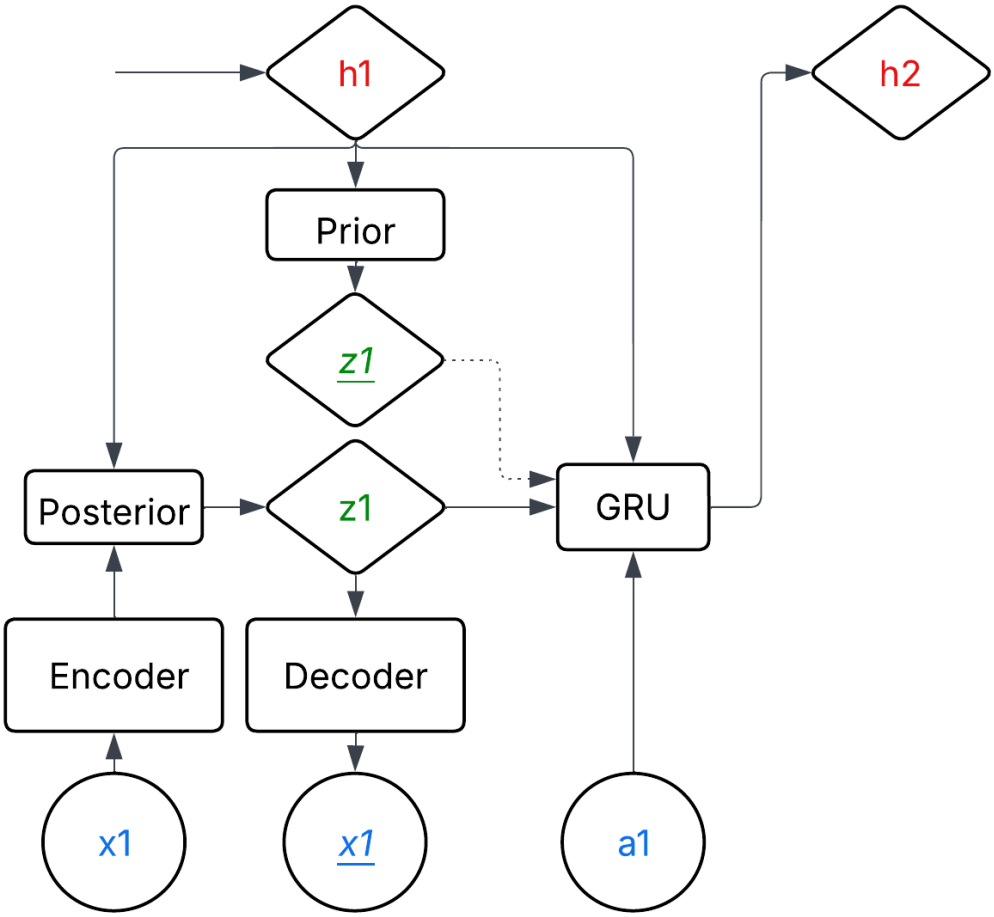



**Figure 3.4:** World model diagram. Rectangles denote neural network modules, diamonds denote latent states, and circles denote world-model variables: observations $x_t$, reconstructed observations $\hat{x}_t$, and actions $a_t$. The decoder input includes both the stochastic latent state $z_t$ and the deterministic latent state $h_t$; the latter connection is omitted from the diagram for readability.

#### 3.2.1.1 Loss function and validation metric

The world model is trained using the following loss function, computed over subsequences of length $T$:

$$\mathcal{L} = \mathbb{E}\left[\sum_{t=1}^{T} \left(\beta_{\text{rec}}\mathcal{L}_{\text{rec},t} + \beta_{\text{dyn}}\mathcal{L}_{\text{dyn},t} + \beta_{\text{rep}}\mathcal{L}_{\text{rep},t} + \beta_{\text{move}}\mathcal{L}_{\text{move},t}\right)\right]. \tag{3.4}$$

With loss weights $\beta_{\text{rec}} = 0.5$, $\beta_{\text{dyn}} = 1$, $\beta_{\text{rep}} = 0.1$ and $\beta_{\text{move}} = 1$.

The dynamics and representation losses follow the KL balancing formulation used in [18], including $\beta_{\text{dyn}}$, $\beta_{\text{rep}}$ :

$$\mathcal{L}_{\text{dyn},t} = \max\left(1, D_{\text{KL}}\left(\text{sg}\left(q_\phi(z_t \mid h_t, x_t)\right) \,\|\, p_\phi(z_t \mid h_t)\right)\right), \tag{3.5}$$

$$\mathcal{L}_{\text{rep},t} = \max\left(1, D_{\text{KL}}\left(q_\phi(z_t \mid h_t, x_t) \,\|\, \text{sg}\left(p_\phi(z_t \mid h_t)\right)\right)\right), \tag{3.6}$$

where $\text{sg}(\cdot)$ denotes the stop-gradient operator.

The reconstruction loss is computed using mean squared error:

$$\mathcal{L}_{\text{rec},t} = \left\|\hat{x}_t^{\text{img}} - x_t^{\text{img}}\right\|_2^2 + \|\hat{x}_t^{\text{vec}} - x_t^{\text{vec}}\|_2^2 . \tag{3.7}$$

Here, $x_t^{\text{img}}$ is the flattened image vector and $x_t^{\text{vec}}$ includes the other observations. The hat indicates it is a prediction.

The auxiliary movement prediction loss involves rotational and translational components. Here, $q$ denotes the rotation (as a quaternion) and $d$ denotes Cartesian translation.

$$\mathcal{L}_{\text{move},t} = 1 - \left|\frac{\hat{q}_t}{\|\hat{q}_t\|_2} \cdot \frac{q_t}{\|q_t\|_2}\right| + \left\|\hat{d}_t - d_t\right\|_2^2 . \tag{3.8}$$

A validation metric is used to evaluate how accurately the learned world model predicts future vehicle motion over extended open-loop rollouts on a held-out validation dataset. Unlike the movement prediction loss, which evaluates one-step predictions during training, this metric measures how prediction errors accumulate when the model feeds the observation latent $z$ predicted from the prior distribution forward without access to observations.

For each trajectory in the validation set, the observations are first encoded sequentially to construct a buffer of deterministic latent states $h_t$. A set of rollout starting points is then sampled uniformly throughout the trajectory. From each starting point, the model performs an open-loop rollout over a fixed horizon of 20 steps using only the stored latent state and the sequence of future actions. At every rollout step, the model predicts the relative motion increment, which is accumulated into a predicted vehicle pose. In parallel, the corresponding ground-truth motion increments from odometry are accumulated into a reference pose trajectory.

The validation metric is defined as the mean Euclidean distance between the predicted and ground-truth translational positions over all rollout steps and all rollout starting points:

$$\mathcal{E}_{\text{val}} = \frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} \|\hat{p}_{i,t} - p_{i,t}\|_2 , \tag{3.9}$$

where $N$ denotes the number of rollout starting points, $T$ is the rollout horizon, $\hat{p}_{i,t}$ is the predicted position at rollout step $t$, and $p_{i,t}$ is the corresponding ground-truth position.

This metric provides an estimate of the long-term consistency of the learned dynamics model and is used for validation and model selection during training. Lower values indicate that the world model maintains more accurate trajectory predictions over extended horizons.

#### 3.2.1.2 Neural networks

The RSSM architecture is implemented using multiple neural network modules for encoding, latent-state prediction, recurrent dynamics, and observation reconstruction. In the selected configuration, the model hidden size is set to 384 and following the general size scaling used in [18].

The image encoder consists of four convolutional layers with kernel size 4, stride 2, and padding 1. For the chosen model size, the channel widths are 24, 48, 96, and 192. Each convolution is followed by group normalization and a SiLU activation. For $64 \times 64$ image observations, this produces a $192 \times 4 \times 4$ feature map, which is flattened into a 3072-dimensional image embedding. Vector observations are processed separately using a smaller MLP, referred to as the vector encoder, which maps the input vector to a 384-dimensional embedding using two linear layers with RMS normalization and SiLU activations.

The posterior encoder combines the deterministic latent state, the image embedding, and the vector embedding. This gives an input dimension of $3072 + 3072 + 384 = 6528$. The posterior network maps this input through a 384-unit hidden layer and then outputs $32 \cdot 24 = 768$ logits, which are reshaped into 32 categorical latent variables with 24 classes each. The stochastic latent state is sampled using straight-through one-hot categorical sampling.

Temporal dynamics are modeled using the recurrent sequence model. The recurrent core follows the DreamerV3-style GRU formulation with block-linear layers [18]. The deterministic state has size 3072 and is split into 8 blocks of 384 units. The previous deterministic state, stochastic latent state, and action are first embedded into 384-dimensional vectors and then processed by block-linear layers. The recurrent update predicts reset, candidate, and update components, each of size 3072.

The dynamics predictor uses an MLP to predict the prior distribution over the stochastic latent variables from the deterministic latent state. It maps the 3072-dimensional deterministic state through two 384-unit hidden layers and outputs 768 logits, again corresponding to 32 categorical variables with 24 classes each. Observation reconstruction is performed using separate image and vector decoders. The image decoder projects both the deterministic and stochastic latent states to a $192 \times 4 \times 4$ spatial representation and upsamples it through four transposed convolution layers with channel widths 96, 48, 24, and the final image output channels. The vector decoder uses an MLP that first embeds the stochastic latent into 384 units, concatenates it with the deterministic state, and maps the combined representation to the reconstructed vector observation. Additional auxiliary prediction heads are implemented for vehicle movement prediction in latent space, producing

a normalized quaternion and a 3D position delta.

#### 3.2.1.3 Observation data and latent buffer

During vehicle operation, RGB images, visual odometry, and IMU data streams are recorded with independent timestamps, then synchronized offline after collection to produce temporally aligned observations with a common frequency of 30 Hz, anchored to the image data. Linear and angular velocities are obtained from visual–inertial odometry, where visual odometry is fused with IMU measurements. Raw IMU linear acceleration measurements are additionally retained as part of the observation space, indicating the direction of gravity. The relative motion targets are derived from the odometry poses, where relative rotation is obtained through quaternion composition, while relative translation is computed from the positional difference between consecutive positions and expressed in the vehicle base coordinate frame of the previous pose. For this work, the data were collected by manually driving the vehicle in a random and diverse manner to capture the vehicle-environment dynamics. The data are preprocessed for use in the WM by normalizing the images and applying the symlog transformation to the vector observations.

To retain temporal context during training, a deterministic latent buffer is maintained for each trajectory, where a trajectory denotes a continuous sequence of observations and actions collected from the vehicle. These trajectories are significantly longer than the subsequences used for optimization, which are drawn from random positions within each trajectory. For a subsequence starting at time $t$, the recurrent state is initialized from the stored latent state $\tilde{h}_t$ rather than from a fixed zero initialization. After processing the training subsequence, the resulting latent states are written back to the buffer. This allows subsequent samples from the same trajectory to be initialized from a state that better reflects the updated dynamics encoded by the neural network [18].

### 3.2.2 Actor–critic reinforcement learning and world model utilization

The actor–critic framework is trained entirely within the learned world model described in the previous section. The world model replaces the real environment during policy optimization and provides an efficient simulator for generating trajectories. The overall method consists of four main components: the environment state built with the world model, the environment reward formulation, the critic (value function), and the actor (policy).

#### 3.2.2.1 Environment state

The environment state $s_t = [h_t, z_t, \theta_t^{\text{norm}}, d_t]$ is composed of the world model (WM) latent representation $[h_t, z_t]$ together with additional path-related features, specifically the relative heading and distance $[\theta_t^{\text{norm}}, d_t]$ to a given target waypoint. Consequently, the WM latent representation itself encodes only information about the vehicle dynamics and terrain, while goal-directed information is provided separately. This allows flexibility for changing the provided path information and reward structure without retraining or regathering training data for the WM.

The initial deterministic recurrent latent $h_t$ and stochastic observation latent $z_t$ are sampled from a precomputed buffer of real-world trajectories, ensuring realistic initial states. Waypoints are then sampled from randomly selected future positions within the same trajectory segment. An alternative approach would be to use waypoints collected during real-world policy interaction, however, this is not considered in the present work.

Given actions from a policy, the environment is rolled out by performing latent transitions using the recurrent dynamics of the world model, where $h_t$ is updated using $h_{t-1}$, $z_{t-1}$, and an action $a_{t-1}$. The next observation latent $z_t$ is sampled from the prior distribution predicted by the dynamics model conditioned on the updated deterministic latent. This results in an open-loop rollout that produces imagined trajectories without incorporating real observations.

Path-related features and rewards are computed from the vehicle pose, which is obtained by integrating predicted motion deltas over time. At each rollout step, the motion deltas are predicted by the auxiliary movement model that takes the world model latent state $[h_t, z_t]$ as input, and the pose is updated by applying these incremental motions. Specifically, the world model outputs a relative motion estimate consisting of a translational displacement $\Delta\mathbf{p}_t = [\delta x, \delta y, \delta z]$ and a rotational increment represented as a quaternion $\Delta\mathbf{q}_t$, both expressed in the vehicle's local (base) frame. Let the pose at time $t$ be given by position $\mathbf{p}_t \in \mathbb{R}^3$ and orientation $\mathbf{q}_t$. The updated pose is obtained via:

$$\mathbf{p}_{t+1} = \mathbf{p}_t + \mathbf{R}(\mathbf{q}_t)\Delta\mathbf{p}_t, \tag{3.10}$$

$$\mathbf{q}_{t+1} = \mathbf{q}_t \otimes \Delta\mathbf{q}_t, \tag{3.11}$$

where $\mathbf{R}(\mathbf{q}_t)$ is the rotation matrix corresponding to $\mathbf{q}_t$, and $\otimes$ denotes quaternion multiplication. This formulation rotates the local displacement into the global frame before accumulating it, while orientation is updated through successive quaternion composition that combining the rotation deltas. Repeated application over time yields an integrated pose trajectory based purely on predicted motion increments.

Given the updated pose, the path-related features, $[\theta_t^{\text{norm}}, d_t]$ are computed relative to the target waypoint $\mathbf{p}^{\text{wp}}$. First the relative displacement in the map frame is calculated as

$$\mathbf{d}_t^m = \mathbf{p}^{\text{wp}} - \mathbf{p}_t = \begin{bmatrix} d_x^m & d_y^m \end{bmatrix}. \tag{3.12}$$

The vehicle yaw angle $\psi_t$ is extracted from the orientation quaternion $\mathbf{q}_t = [q_x, q_y, q_z, q_w]$ as

$$\psi_t = \text{atan2}\left(2(q_w q_z + q_x q_y), 1 - 2(q_y^2 + q_z^2)\right). \tag{3.13}$$

The displacement vector is then rotated into the vehicle base frame using the inverse yaw rotation $R(\psi_t)^{-1}$:

$$\begin{bmatrix} d_x^b \\ d_y^b \end{bmatrix} = R(\psi_t)^{-1} \begin{bmatrix} d_x^m \\ d_y^m \end{bmatrix} = \begin{bmatrix} \cos\psi_t & \sin\psi_t \\ -\sin\psi_t & \cos\psi_t \end{bmatrix} \begin{bmatrix} d_x^m \\ d_y^m \end{bmatrix}. \tag{3.14}$$

The relative heading is computed as

$$\theta_t = \text{atan2}(d_y^b, d_x^b), \tag{3.15}$$

and is normalized to the interval $[-1, 1]$ as

$$\theta_t^{\text{norm}} = \frac{\theta_t}{\pi}. \tag{3.16}$$

The distance feature is the planar Euclidean distance to the waypoint:

$$d_t = \sqrt{(d_x^m)^2 + (d_y^m)^2}. \tag{3.17}$$

Together, $[\theta_t^{\text{norm}}, d_t]$ form an ego-centric representation of the waypoint relative to the vehicle.

#### 3.2.2.2 Environment rewards

Using the imagined pose trajectory and the target waypoint, rewards are computed to encourage progress toward the goal while discouraging inefficient motion.

At each time step, the planar distance between the vehicle and the waypoint is computed as $d_t$. The primary reward signal is the reduction in goal distance between consecutive steps:

$$r_t^{\text{prog}} = d_t - d_{t+1}. \tag{3.18}$$

To discourage unnecessary motion, a penalty is applied based on the step size. Let $s_t$ denote the distance traveled between two consecutive positions. The step penalty consists of two parts, a proportional cost on the step size and an additional penalty if the step exceeds a predefined maximum step length $s_{\max}$:

$$c_t^{movement} = 0.2\, s_t + 0.5\, \max(s_t - s_{\max}, 0). \tag{3.19}$$

To encourage stable control behavior, an additional penalty is applied to abrupt changes in the action commands. Let $a_t$ denote the action at time step $t$. The action change is computed as the difference between consecutive actions:

$$\Delta a_t = a_t - a_{t-1}. \tag{3.20}$$

A smoothness penalty is then defined as the norm of this action difference scaled by a weighting factor:

$$c_t^{\text{act}} = \lambda_{\text{act}}\, \|\Delta a_t\|_2, \tag{3.21}$$

where $\lambda_{\text{act}} = 0.01$.

A sparse reward $r_t^{\text{goal}}$ is assigned when the vehicle reaches the waypoint. The waypoint is considered reached once the distance to it falls below a threshold of 0.05m. The reward is granted only once, at the time step when the agent first enters this region, which simultaneously terminates the episode.

The total reward at each step is expressed as:

$$r_t = r_t^{\text{prog}} + r_t^{\text{goal}} - c_t^{movement} - c_t^{act}. \tag{3.22}$$

#### 3.2.2.3 Critic

The critic is a learned value function $V(s)$ trained to estimate the expected future cumulative reward $R$ conditioned on environment state under the current action policy. The WM latent component of the state is beneficial for the critic due to its approximately Markovian representation capturing predictive information about future state distributions [18].

The accumulated rewards target is calculated as:

$$R_T = V(s_T) \tag{3.23}$$

$$R_t = r_t + \gamma c_t \left[(1-\lambda)V(s_{t+1}) + \lambda R_{t+1}\right] \tag{3.24}$$

where $r_t$ is the step reward associated with the state transition $s_t \to s_{t+1}$, $\gamma$ is the discount factor, $c_t$ is the continuation flag, $V(s_{t+1})$ is the estimated value of the next state, and $\lambda$ controls the degree of bootstrapping [18][13]. The continuation flag masks return contributions after episode termination, preventing value propagation beyond terminal states.

The used bootstrap formulation enables consideration of rewards beyond the finite rollout horizon $T = 20$, through the estimated value. This allows the policy to train toward waypoint targets that are farther away than can be reached within a single rollout, while still encouraging trajectories that position the vehicle favorably for future progress.

The critic implementation uses a categorical value distribution with exponentially and symmetrically spaced bins and two-hot encoded targets, following [18]. This representation decouples gradient magnitude from value scale, improving stability when the return distribution spans a varying numerical range. In this context, such variation can arise from differences in the distance between the sampled waypoint and the initial vehicle state. To prevent the waypoint-related features from being dimensionally dominated by the significantly larger WM latent representation, separate encoder networks are used for the WM latent state $[h_t, z_t]$ and the waypoint features $[\theta_t^{\text{norm}}, d_t]$ before their feature embeddings are concatenated and processed by the critic and actor networks.

The critic is trained by minimizing the negative log-likelihood between the predicted categorical value distribution and the two-hot encoded target distribution.

$$\mathcal{L} = -\mathbb{E}\left[\sum_{t=1}^{T} \log p(R_t \mid s_t)\right] \tag{3.25}$$

#### 3.2.2.4 Actor

The actor represents the control policy $\pi(a_t \mid s_t)$ and maps the environment state to continuous steering and throttle actions. The input state consists of the WM latent representation together with the waypoint-related features:

$$s_t = [h_t, z_t, \theta_t^{\text{norm}}, d_t]. \tag{3.26}$$

The actor predicts both the mean $\mu_t$ and log-standard deviation $\log \sigma_t$ for each action dimension. The log-standard deviation is bounded to a predefined interval to avoid numerical instability and excessively uncertain actions:

$$\log \sigma_t \in [\log \sigma_{\min}, \log \sigma_{\max}]. \tag{3.27}$$

A Gaussian action distribution is then defined as

$$\tilde{a}_t \sim \mathcal{N}(\mu_t, \sigma_t), \tag{3.28}$$

followed by a hyperbolic tangent transformation:

$$a_t = \tanh(\tilde{a}_t), \tag{3.29}$$

which constrains the actions to the interval $[-1, 1]$. This matches the control representation used by the world model and environment, where the two action dimensions correspond to steering and throttle commands.

During training rollouts, actions are sampled from the transformed distribution to encourage exploration, while during inference the policy acts deterministically by directly using the mean action of the predicted distribution.

The actor is optimized using a policy-gradient objective based on the estimated advantage:

$$A_t = \frac{R_t - V(s_t)}{S}, \tag{3.30}$$

where $R_t$ is the bootstrapped return, $V(s_t)$ is the critic estimate, and $S$ is an exponential moving average normalization factor computed from the range between the $95^{\text{th}}$ and $5^{\text{th}}$ percentiles of the bootstrapped return targets within the current training batch for reward scaling [18].

The actor loss is defined as

$$\mathcal{L}_{\text{actor}} = -\mathbb{E}\left[\log \pi(a_t \mid s_t) A_t + \lambda_{\text{ent}} \mathcal{H}(\pi(\cdot \mid s_t))\right], \tag{3.31}$$

where $\mathcal{H}(\pi(\cdot \mid s_t))$ denotes the entropy of the action distribution and $\lambda_{\text{ent}}$ is a small entropy regularization coefficient.

## 3.3 Classical implementation

An Adaptive Pure Pursuit algorithm was developed which used a sliding window over the path to find the closest point to the back axis and the closest point further away than the lookahead distance. Due to lack of time, no longitudinal control was developed and therefore the controller was in reality reduced to the vanilla Pure Pursuit version.

Parameters were chosen by tuning on diverse set of backtracking of manually driven paths. The parameters were then not tuned for each individual path in Table 3.2. The motivation was to partly save time but also that no tuning was made on the WM-RL controller. The parameters choosen were

$$\begin{cases} k & = 0.95 \text{ s}, \\ v & = 0.55 \text{ m/s}, \\ l_{min} & = 1.1 \text{ m}, \end{cases} \tag{3.32}$$

resulting in the actual lookahead distance being $l = l_{min} = 1.1$ m. The speed was set to this value to be comparable to WM-RL controllers max speed of 0.65 m/s. However the WM-RL controller could slow down further helping it in some scenarios.

## 3.4 Evaluation

The WM-RL controller was evaluated on a range of paths testing for specific manuevers, described in Section 3.4.2. It was also tested on its backtracking ability. That is, following an arbitrary manually recorded path backwards or forwards. Forwards was also tested because there was only a camera facing forwards on the vehicle and it was believed that this would make it hard for the WM-RL to go backwards. The WM-RL controller was tested on the following terrains:

- uneven grass,
- large grained sand,
- uneven surface with sticks and dirt in forest.

This was done to test how well the controller maintains accurate path tracking under varying vehicle dynamics.

The WM-RL controller was also benchmarked against the Pure Pursuit controller on several paths and surface conditions. Both the WM-RL controller and the Pure Pursuit controller are evaluated on identical reference paths and similar surface conditions. However, this was just to give one reference point and is far from the state of the art conventional controllers.

During all tests, the distance to the goal pose considered reaching the goal was set to 0.1 m. The path-following metrics are computed from the estimated vehicle pose rather than from an independent ground-truth measurement.

### 3.4.1 Performance metrics

The controllers are evaluated based on four categories:

1. **Time** to complete path
2. **Cross track error:** How closely it tracks the path, viewed from a BEV, so the CTE is 2D.
3. **Control effort**, i.e. how much it steers and throttles and thereby tear the servo and motor.
4. **Smoothness**: How smooth it drives, to not wear the hardware.

At each timestep, the closest point on the linearly interpolated reference path is identified, and the perpendicular distance between the vehicle position and the path is computed, in 2D. This distance defines the cross-track error (CTE).

The comparison is performed across multiple terrain types to evaluate how well each controller adapts to the terrain. The E90L turn on sand was performed four times, with the first two turns being on a slightly slanted downhill, while the two others were on a slight uphill slant.

The control effort is measured as the RMS of the normed, in the interval [-1, 1],

steering and throttle output.

The smoothness was measured as the max and RMS 2D acceleration $a$ and jerk $\frac{da}{dt}$.

### 3.4.2 Paths

To evaluate the controllers' performance across a range of path types, the test paths listed in Table 3.2 were selected. They cover constant curvature paths, clothoids, oscillations, and maneuvers requiring reversing.

**Table 3.2:** Path configurations used for controller evaluation. Here $r$ is the radius, $\lambda$ the wave length, $A$ the amplitude and $l$ the arm length of the cross pattern.

| Abbrev. | Description | Path type | Parameters |
|---|---|---|---|
| C | Circle | Constant curvature (closed) | $r = 2.0$m |
| E90L | Clothoid, 90° left turn | Linearly varying curvature | $r_{\min} = 2.0$ m |
| S0.3 | Low-amplitude sine wave | Oscillations | $\lambda = 3.0$ m, $A = 0.3$ m |
| S0.8 | High-amplitude sine wave | Oscillations | $\lambda = 3.0$ m, $A = 0.8$ m |
| K | K-turn | Reversal, constant curvature, discrete jumps in curvature. | $r = 3.5$ m |
| X | Cross pattern | Reversal | $l = 2.0$ m |

The K-turn is a maneuver for Ackermann vehicles to turn 180°. The K-turn used here is oversized for the UGV. However, it still requires the ability to switch from high curvature to a straight line and between going forward and reversing.

A clothoid curve is a curve with linearly changing curvature [29]. For an Ackermann vehicle, the steering angle is approximately proportional to curvature (for small angles). Therefore, if the steering angle changes at a constant rate, the vehicle naturally traces a clothoid. In this work the Python package Clothoids [30] is used to generate these.

All reference paths can be found illustrated in Section B.1.

### 3.4.3 Backtrack

#### 3.4.3.1 Sand

The vehicle was manually driven around a square sandbox, the surface it was driving on was large-grained sand, which made the car slip and wobble significantly. The controllers were then tasked with driving back around, either facing backwards or forwards. See Figure 4.10 as an example. The spacing between points to follow was 0.1 m.

#### 3.4.3.2 Forest

The WM-RL controller was also tasked with driving around very challenging terrain in a forest. The terrain included rocks, sticks, small logs, vegetation, and loose dirt. The terrain was also slanted and sloped, see Figure B.32 in Appendix. Pictures showcasing the terrain are found in Figure 3.5.

**Figure 3.5:** Three pictures illustrating the terrain driven on when backtracking in the forest. The white flags are waypoints building up the course.

The car was driven manually along a course predefined by white flags, as shown in Figure 3.5. This was done to try to repeatedly backtrack approximately the same trajectories. It was also done to keep an external measure of how well the controller followed the path, so as not to rely entirely on the flawed odometry. In total, four runs were made, with two runs successfully backtracking the controller's own backtrack, resulting in a total of six trajectories.

### 3.4.4 Data visualization

As the odometry tended to drift, it sometimes made the map frame twist, resulting in poor BEV projections for visualization. Even though the vehicle had followed the path correctly, it appeared not to have because of this twist. That is, the VSLAM makes the vehicle appear to drive on a twisted plane. This was corrected by fitting a plane through all points of the vehicle's trajectory and rotating it to be level. See Section A.1 for details. However, this assumes near planar surface, so it did not work on the forest backtrack or the sand reference paths. Here, the distance check for map jumps, explained in Section 3.1.2.3.3, had been made 3D, previously 2D, and a rotation check had been included. This resulted in that the fitting a plane and rotation correction no longer was needed.

Motion trails was used to showcase how the vehicle actually drove in reality and, among other things, show the WM-RL controller's emergent behaviors, see Section 4.2.2. The text-to-segmentation version of SAM3 [31] was used to segment out the

research vehicle. These segments were then placed onto the first picture in the video. The images does not exactly represent the true trajectory due to not keeping the camera completely still, but is able to capture important features.

# 4

# Results and Discussion

## 4.1 Latent imagination rollouts

This section presents results for world-model open-loop rollouts. Starting from a latent state inferred from the preceding observation–action history, the model latent state is propagated forward using the continuation of the recorded action sequence, while the stochastic observation latent state is sampled from the model's predicted prior distribution without conditioning on future observations.

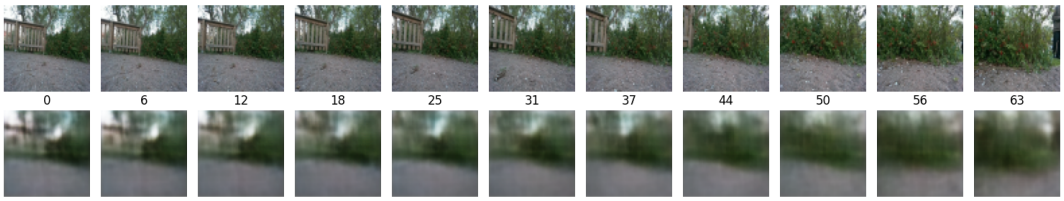
To quantify open-loop pose-prediction accuracy and the relative importance of different observation modalities, the model was rolled out for 20 steps from multiple sampled initial latent states. To assess modality importance, different observation modalities were masked out during latent-state encoding by replacing the selected inputs with zeros. Prediction error is reported as the RMSE of the Euclidean distance between the predicted pose positions and the recorded ground-truth trajectory.

The resulting pose-distance RMSE values for the different surface types are reported in Table 4.1. The results indicate that image observations provide only limited additional benefit compared to vector observations. This may be explained by the relatively monotonous dataset, in which transitions between different environmental conditions are sparse. As a result, visual observations may contain less additional predictive information than vector observations, which directly describe the current vehicle response through velocity and acceleration. In general, perfect motion prediction should not be expected nor necessarily required, since the vehicle–environment interaction contains stochastic effects that cannot be fully inferred from the available observation history. Furthermore, because the world model is generative, its rollouts sample plausible realizations of these stochastic effects based on the learned structure of the environment, rather than producing exact deterministic predictions. Over rollouts, these sampled variations accumulate, increasing the prediction error further.

**Table 4.1:** Open-loop 20-step rollout pose-distance RMSE in metres across surface types under different observation masks. Averaged over 1000 rollouts per surface.

| Surface type | All masked | Vector masked | Image masked | No masking |
|---|---|---|---|---|
| Grass | 0.1034 | 0.0938 | 0.0320 | 0.0262 |
| Gravel | 0.0348 | 0.0351 | 0.0279 | 0.0277 |
| Carpeted floor | 0.0234 | 0.0230 | 0.0188 | 0.0178 |

Figures 4.1a and 4.1b show decoded image observations from open-loop latent rollouts alongside their corresponding ground-truth observations. Since the rollout is propagated without conditioning on future observations, the latent trajectory would be expected to gradually diverge from the ground truth as it moves beyond the part of the future that is well constrained by the observation history. In such a case, the model should still generate visually plausible alternative futures, even if these differ from the recorded ground-truth sequence. However, this behavior is not observed. Instead, the decoded image rollouts suggest limited visual imagination capability. Figure 4.1 suggests overfitting, as the dynamics predictor and image decoder appear to reproduce the training sequence rather than learning a generalizable visual dynamics model. A likely cause of this behavior is the limited size of the training dataset, which consists of roughly $10^5$ environment steps. In contrast, the main reference world model implementation, DreamerV3 [18], uses on the order of $10^8$ environment steps for visually complex tasks. Furthermore, the validation metric described in Section 3.2.1.1, used for model selection, prioritizes the best average pose trajectory prediction relative to ground truth over decoder validation results. See Figure 4.2 for an example pose trajectory.




**(a)** Training trajectory.


**(b)** Validation trajectory.

**Figure 4.1:** World model open-loop rollout (64 steps). Decoded image predictions (bottom row) shown alongside the ground-truth (top row).

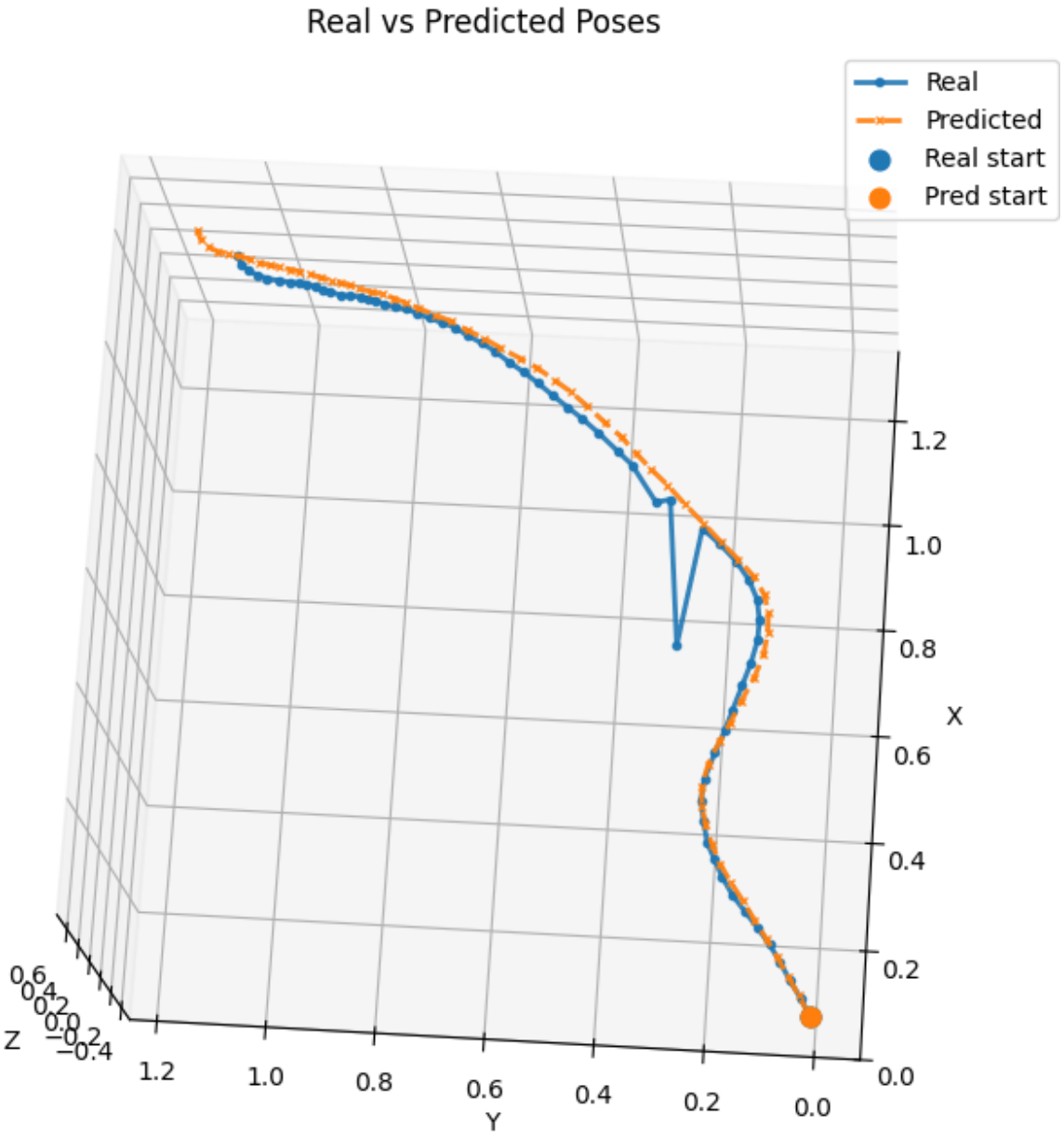



**Figure 4.2:** World model open-loop rollout (64 steps) from a validation trajectory. Accumulated predicted and reference poses derived from motion deltas in metres. The predicted poses closely follows the reference trajectory, but does not reproduce the positional anomaly observed in the visual–inertial odometry estimate.

## 4.2 Discussion and results on the real world performance of WM-RL controller

This section discusses and presents qualitative and quantitative results on the real-world performance of the WM-RL controller, compared with the traditional Pure Pursuit (PP) method.

### 4.2.1 Trajectory tracking results

Results of path-following tests for the evaluated controllers.

#### 4.2.1.1 90 degree turn path on grass

The reference path for E90L is shown in Figure B.1.

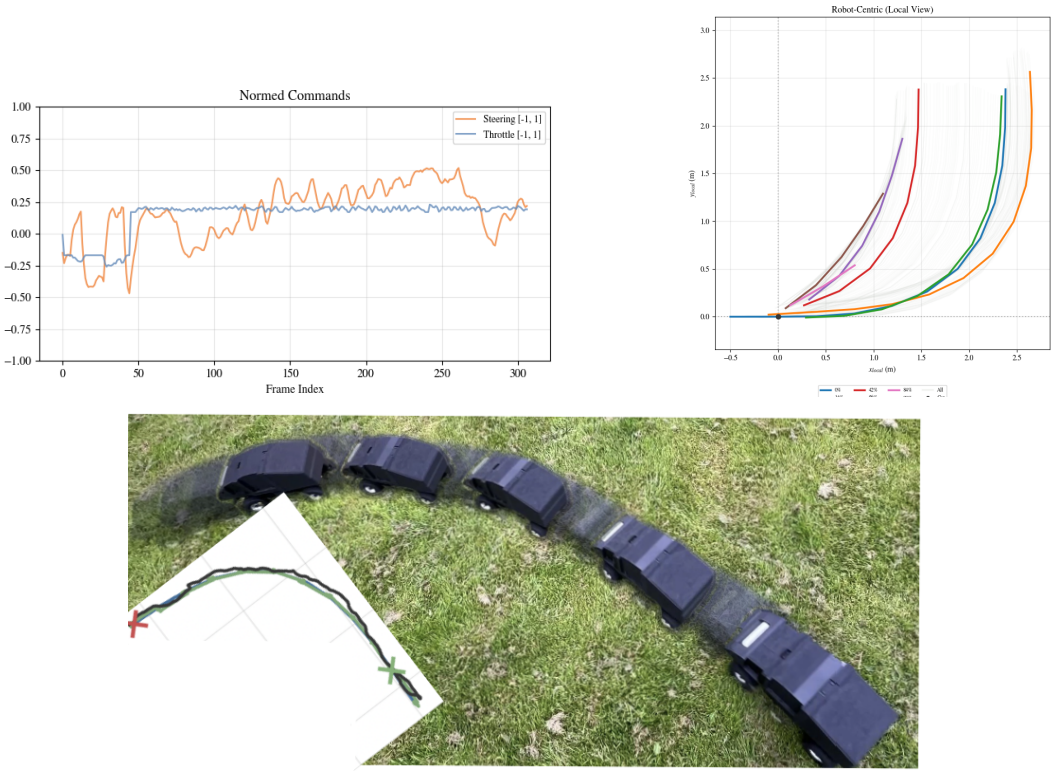



**Figure 4.3:** WM-RL following path E90L on grass. Bottom: motion trail showing the vehicle trajectory (black) from start (green cross) to end (red cross), the odometry-adjusted followed points (green), and the reference path (blue). Top right: robot-centric view, what the controller actually steered after, at different points during the path. Top left: normed steering commands. Note that this path included a short straight it should back up before entering the turn.

**Table 4.2:** Performance metrics for WM-RL following path E90L on grass.

| Metric | Value |
|---|---|
| Mean CTE | 0.042 m |
| Max CTE | 0.106 m |
| Std CTE | 0.028 m |
| RMS Steering | 0.280 |
| RMS Speed | 0.195 |
| Max $a$ | 0.074 m/s$^2$ |
| RMS $a$ | 0.023 m/s$^2$ |
| Max $\frac{da}{dt}$ | 0.027 m/s$^3$ |

Figure 4.3 and Table 4.2 show that the WM-RL controller follows the E90L turn with high precision, despite VSLAM errors and uneven terrain. The cross-track error (CTE) metrics over five runs in Table 4.3 confirm this performance.

Further, the controller handles the linearly changing curvature of the clothoid well. This is evident from the curvature changes in the robot-centric view and the gradual increase in steering during the turn. Moreover, the controller has learned to make adjustments using low steering control signals, resulting in a lower RMS steering value, as shown in Table 4.2. The controller also maintains a steady forward speed throughout the turn, reducing jerk and acceleration and thereby enabling smoother driving. In addition, the controller demonstrates its ability to switch seamlessly between forward and reverse throttle.

**Table 4.3:** Average performance metrics across five path-following runs on the E90L path (grass).

| Metric | WM-RL | PP |
|---|---|---|
| Mean CTE | 0.038 m | 0.329 m |
| Max CTE | 0.118 m | 0.586 m |
| Std CTE | 0.029 m | 0.180 m |

Compared to PP, WM-RL shows much better tracking ability, as seen in Table 4.3. This is partly due to its ability to continuously readjust to the path without starting to oscillate, and partly due to it not explicitly calculating a circular arc to the next points, and it is therefore more flexible. As mentioned in Section 3.3, the parameters for the PP were not tuned for this path. It therefore cuts the corner to have a large enough lookahead to be able to drive straight without excessively large oscillations. The corner-cutting behavior can be seen in Figure B.7 in the appendix. There is also another run of the WM-RL shown in Figure B.8. The corner-cutting behavior is further shown in the motion trail of the PP in Figure 4.5.

#### 4.2.1.2 90 degree turn path on sand

The same reference path, E90L, was also followed on the sand surface by the WM-RL controller. The surface was also sloped, although the slope was not consistent across all runs. The performance was better on sand than on grass. The grass may have been more uneven, whereas the sand was more sloped and slippery. Comparing the two runs in Figure 4.3 and Figure 4.4, we see that the controller slightly overshot the corner on grass, while it slightly cut the corner on sand. The overshoot is also visible in another grass trajectory in Figure B.8. The corner cutting is visible in most BEV plots of the sand runs, as shown in Figures B.16, B.19, and B.21.

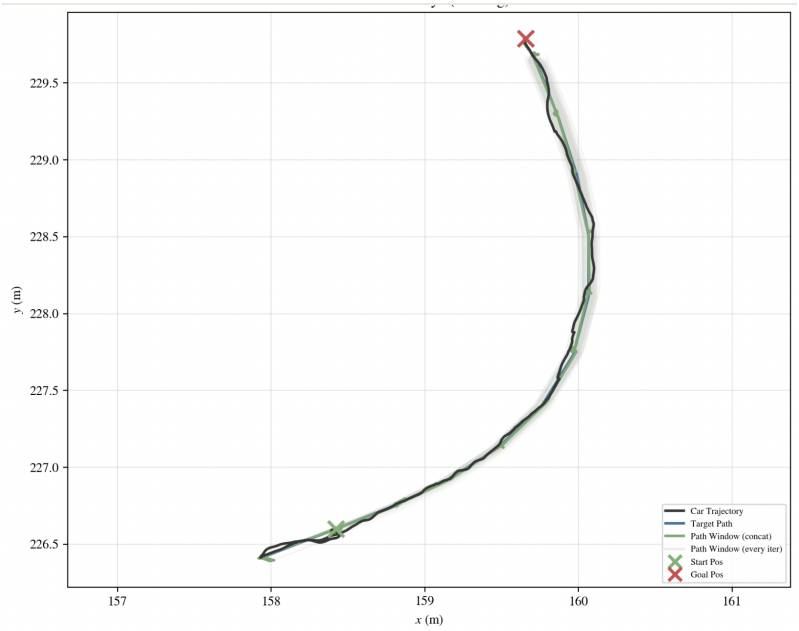



**Figure 4.4:** WM-RL following E90L path on sand. Run 2.

Since the corner-cutting behaviour was observed independently of the surface slope, as discussed in Section 3.4.1, the sand surface, rather than the grass surface, may have caused this effect. Furthermore, by comparing the control signals on grass in the upper-left plot in Figure 4.3 with the control signals on sand in Figures B.17, B.18, and B.20, we observe substantially more steering oscillation on grass. Considering these points, it is possible that the WM-RL controller adjusts its control actions slightly depending on the surface, either indirectly through velocity feedback or directly through image input. However, the dataset is limited, and a major source of error is the visual odometry. One possibility is that the visual odometry simply performed better at the sand location than at the grass location.

The WM-RL controller also tracked the E90L path better on sand than on grass, with similar smoothness, seen in Table 4.4. Individual runs can be found in Tables B.1, B.2, B.3 and B.4. The RMS steering was lower, likely due to cutting the corner slightly on sand. See Tables 4.2 and 4.3 for performance on grass.

**Table 4.4:** Average performance over the four E90L sand runs.

| Metric | Value |
|---|---|
| Time | 13.33 s |
| Mean CTE | 0.024 m |
| Max CTE | 0.086 m |
| Std CTE | 0.021 m |
| RMS Steering | 0.213 |
| RMS Speed | 0.178 |
| Max $a$ | 0.071 m/s$^2$ |
| RMS $a$ | 0.017 m/s$^2$ |
| Max $\frac{da}{dt}$ | 0.033 m/s$^3$ |
| RMS $\frac{da}{dt}$ | 0.008 m/s$^3$ |

#### 4.2.1.3 Circle path on grass

See Figure B.2 for reference path of the circle.


**Figure 4.5:** Pure Pursuit trying to follow path C. Motion trail (left) and birds eye view plot where green is the path and black is the car trajectory (right).

The PP controller shows its inability to follow a constant-curvature path when not tuned, as shown in Figure 4.5. It therefore obtains much higher CTE metrics than WM-RL, as shown in Table 4.5. Both controllers drive smoothly, as shown in Figures B.10 and B.9, and in Table 4.5. However, PP drives more smoothly when considering the maximum acceleration $a$. The WM-RL controller uses a lower control signal because it drives a wider turn.

The WM-RL controller also shows that it can handle a constant-curvature path. It adjusts well to the sudden change in curvature at the start of the path and when the path straightens out at the end. This can be seen both in Table 4.5 and in Figure B.10.

**Table 4.5:** Average performance metrics across five path-following runs on the C path (grass).

| Metric | WM-RL | PP |
|---|---|---|
| Mean CTE | 0.057 m | 0.431 m |
| Max CTE | 0.172 m | 0.598 m |
| Std CTE | 0.048 m | 0.153 m |
| RMS $\delta$ | 0.362 | 0.510 |
| RMS $v_x$ | 0.197 | 0.170 |
| max $a$ | 0.121 m/s$^2$ | 0.067 m/s$^2$ |
| RMS $a$ | 0.025 m/s$^2$ | 0.016 m/s$^2$ |
| max $\frac{da}{dt}$ | 0.042 m/s$^3$ | 0.034 m/s$^3$ |
| RMS $\frac{da}{dt}$ | 0.011 m/s$^3$ | 0.008 m/s$^3$ |

#### 4.2.1.4 K-turn path on grass

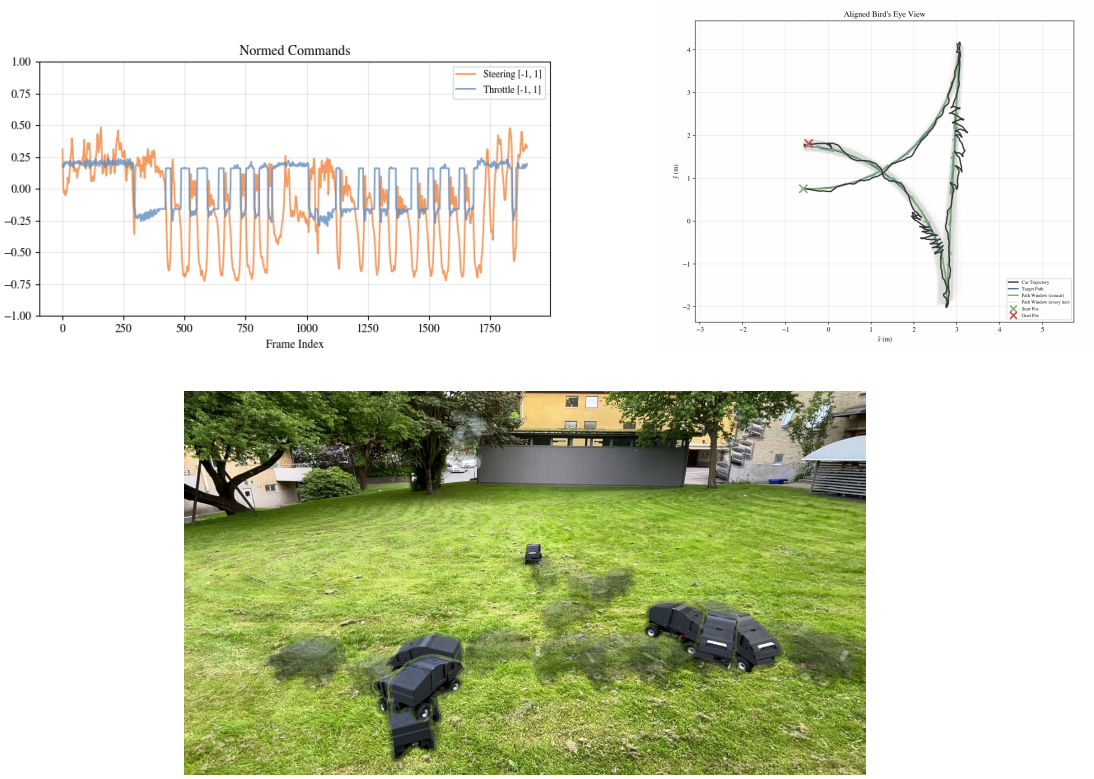



**Figure 4.6:** WM-RL following path K on grass. Top right: BEV plot. Top left: Normed control outputs. Bottom: motion trail from the vehicle.

**Table 4.6:** Performance metrics for one run WM-RL following path K on grass.

| **Metric** | **Value** |
|---|---|
| Mean CTE | 0.064 m |
| Max CTE | 0.266 m |
| Std CTE | 0.055 m |
| RMS $\delta$ | 0.347 |
| RMS $v_x$ | 0.181 |
| Max $a$ | 0.077 m/s$^2$ |
| RMS $a$ | 0.024 m/s$^2$ |
| Max $\frac{da}{dt}$ | 0.039 m/s$^3$ |
| RMS $\frac{da}{dt}$ | 0.009 m/s$^3$ |

The WM-RL controller once again demonstrates its ability to switch between forward and reverse motion. It transitions smoothly between the circular arc and the straight segment. However, it appears less effective when reversing over longer distances. When entering the straight segment, it initially starts reversing but then turns around to drive forward. It then repeats this behaviour when entering the second arc. This results in unnecessarily high steering control effort and a lower overall speed. Although the controller makes several adjustments, as shown in Figure 4.6, it maintains smooth driving behaviour, as shown in Table 4.6. Overall, it tracked the path well over five runs, as shown in Table 4.7. The controller also showed signs of an emergent behaviour of steering before applying throttle, indicating that it has learned that the car turns more easily at lower speeds. Finally, it demonstrated efficient adjustments by driving forward while turning in one direction and then reversing while turning in the opposite direction. This is discussed further in Section 4.2.2.

**Table 4.7:** Average performance metrics across five path-following runs on the K path (grass).

| Metric | WM-RL | PP |
|---|---|---|
| Mean CTE | 0.063 m | – |
| Max CTE | 0.334 m | – |
| Std CTE | 0.067 m | – |

The PP controller could not finish the path. This is due to its inability to switch between forwards and reversing by its own. Even if a planner would have helped it with that, it would have had a hard time dealing with the sudden changes in curvature.

#### 4.2.1.5 Cross pattern path on grass

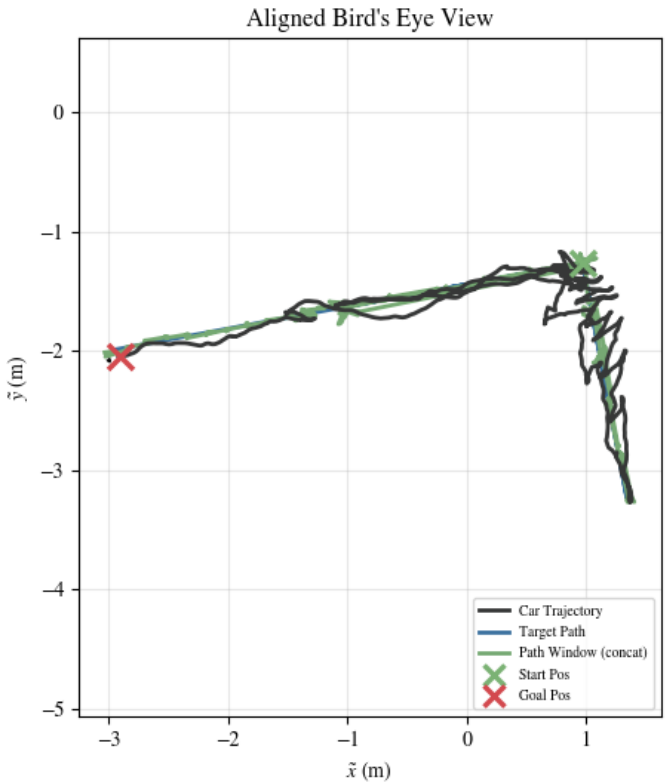



**Figure 4.7:** WM-RL traversing path X on grass. Motion trail showing the vehicle trajectory (black) from start (green cross) to end (red cross), the odometry-adjusted followed points (green), and the reference path (blue).

The cross-pattern path (path X, Figure B.3) tests similar behaviors to the K-turn. Consequently, the PP controller failed to complete it (Figure B.11). The WM-RL controller completed the path, but its turning strategy was suboptimal (Figure 4.7). A human driver would likely move up the first arm, reverse past the starting position, and then drive forward along the left arm. Because the WM-RL controller avoids extended reverse driving, it turns around early on the right arm. The way it turns around, which is very zigzaggy as shown in Figure 4.7 and many other cases such as in Figure 4.6, could be due to the controller's reward shaping, see Eq 3.22. In this case, the reward for being close to the next waypoint can make the controller want to turn around while staying as close to the next waypoint as possible during the maneuver, leading to an overall suboptimal behavior.

That the controller turns several times to turn around is probably also due to the cost term in Eq 3.22 penalizing change in $c_t^{act}$. This punishes to sharp turns which limits the controller performance.

Note also that the VSLAM performance was likely poor here since on the right arm, one way it is suppose to drive straight forward, but it does a zigzagging motion in both directions.

#### 4.2.1.6 Low-amplitude sine wave path on grass

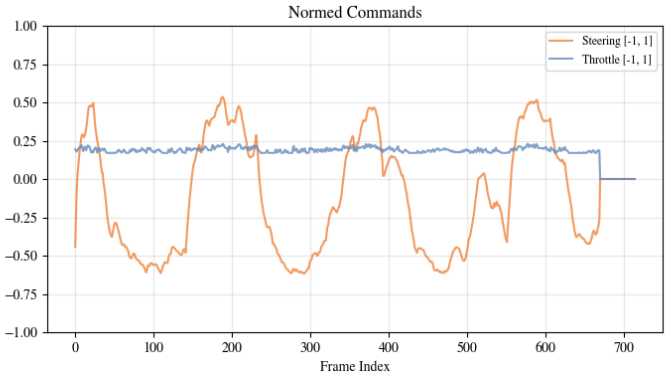



**Figure 4.8:** Normed control outputs for WM-RL controller performing path following on S0.3 (grass).

The WM-RL shows that it can handle repeated gentle turns without slowing down, see Figure B.13 and 4.8. The throttle it is giving is equivalent to near its top speed of 0.65 m/s, showing it tries to drive as fast as possible. The WM-RL also outperformed the PP controller, see Table 4.8. The PP had a lookahead distance that was too long for the given wavelength and therefore cut the corners, as shown in Figure B.12. The PP drove smoother, but it cut the corners and therefore drove a straighter path.

**Table 4.8:** Average performance metrics across five runs on the S0.3 path (grass).

| Metric | WM-RL | PP |
|---|---|---|
| Mean CTE | 0.098 m | 0.182 m |
| Max CTE | 0.333 m | 0.338 m |
| Std CTE | 0.067 m | 0.102 m |
| RMS $\delta$ | 0.380 | 0.209 |
| RMS $v_x$ | 0.184 | 0.157 |
| max $a$ | 0.082 m/s$^2$ | 0.043 m/s$^2$ |
| RMS $a$ | 0.020 m/s$^2$ | 0.013 m/s$^2$ |
| max $\frac{da}{dt}$ | 0.029 m/s$^3$ | 0.016 m/s$^3$ |
| RMS $\frac{da}{dt}$ | 0.008 m/s$^3$ | 0.005 m/s$^3$ |

#### 4.2.1.7 High-amplitude sine wave path on grass

The WM-RL controller was able to track the high-amplitude sinusoidal path without substantially cutting corners, in contrast to the behaviour observed for the Pure Pursuit controller, as shown in Figures B.15 and B.14. This performance appeared to be achieved through frequent use of reverse motion to reorient the vehicle, as indicated by the control commands in Figure 4.9. However, the controller did not fully exploit the available steering range and appeared to rely too heavily on reversing manoeuvres. One possible explanation is that the reward function penalized changes in the control actions too strongly, thereby discouraging large steering inputs.

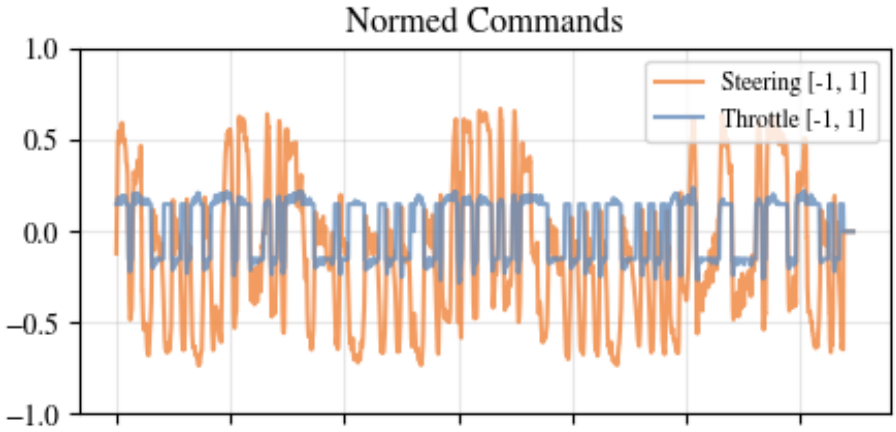



**Figure 4.9:** Normed control outputs for WM-RL controller performing path following on S0.8 (grass).

Although the PP controller does not follow the path as accurately, as shown by its higher CTE values in Table 4.9, it exhibits slightly lower longitudinal acceleration and jerk than the WM-RL controller. The maximum acceleration and jerk values are likely due to an anomalous events in the VIO. However, the general jerk and acceleration of the WM-RL controller could likely be reduced further through more explicit reward penalties on them.

**Table 4.9:** Performance metrics for singular path-following run on the S0.8 path (grass).

| Metric | WM-RL | PP |
|---|---|---|
| Mean CTE | 0.121 m | 0.334 m |
| Max CTE | 0.557 m | 0.730 m |
| Std CTE | 0.084 m | 0.212 m |
| RMS $\delta$ | 0.406 | 0.569 |
| RMS $v_x$ | 0.162 | 0.157 |
| max $a$ | 0.361 m/s$^2$ | 0.077 m/s$^2$ |
| RMS $a$ | 0.029 m/s$^2$ | 0.022 m/s$^2$ |
| max $\frac{da}{dt}$ | 0.347 m/s$^3$ | 0.024 m/s$^3$ |
| RMS $\frac{da}{dt}$ | 0.019 m/s$^3$ | 0.007 m/s$^3$ |

#### 4.2.1.8 High-amplitude sine wave path on sand

As mentioned in Section 4.2.1.2, the WM-RL controller generally performed better on sand than on grass. The same tendency was also observed for the high-amplitude sine wave path, S0.8. The controller followed the path with a mean CTE of 0.103 m, compared to 0.121 m on grass, while maintaining a higher RMS speed. The trajectory and control signals are shown in Figures B.22 and B.23, and the performance metrics are given in Table B.5. A notable difference compared to the grass run was that the controller reversed much less frequently to re-adjust its orientation.

#### 4.2.1.9 Sandbox backtrack


**Figure 4.10:** WM-RL following a path in large-grained sand going forwards. Left: Motion trail. Right: Birds eye view plot with blue as the original path to follow, green the adjusted path windows which the controller followed and in black its trajectory.

The PP controller had a hard time going around the sandbox, repeatedly crashing into the corners while trying to cut them. However, on one run it started to oscillate early and therefore managed to get around the sandbox, but with poor performance,

see Figure B.24 and Table B.6. The WM-RL performed much better, see Figure 4.10 and Table 4.10. It could probably go around the sandbox much closer. However, that type of path was chosen to give PP a chance. The zig-zag behaviour is from when the path was manually recorded by driving zig-zaggy, so the controller is actually following it really well. Note, however, that the paths the PP and WM-RL controllers drove are not identical.

Further, the WM-RL controller can be seen reversing to adjust in the bottom right corner in the left image in Figure 4.10. Finally, it is worth noting that the WM had very limited data on similar surfaces. Therefore, despite lossing traction constantly the WM-RL controller appears to handle it.

**Table 4.10:** Performance metrics for WM-RL following a path in large-grained sand.

| Metric | Value |
|---|---|
| mean CTE | 0.052 m |
| max CTE | 0.295 m |
| std CTE | 0.052 m |
| RMS $\delta$ | 0.389 |
| RMS $v_x$ | 0.179 |
| max $a$ | 0.087 m/s$^2$ |
| RMS $a$ | 0.023 m/s$^2$ |
| max $\frac{da}{dt}$ | 0.029 m/s$^3$ |
| RMS $\frac{da}{dt}$ | 0.007 m/s $^3$ |

#### 4.2.1.10 Forest backtrack

The WM-RL controller was also able to backtrack on difficult terrain in a forest area. The controller was able to track with a low CTE and high smoothness, as seen in Table 4.11, even though the terrain was very difficult, as can be seen in Figures 3.5 and 4.11.

The controller was able to handle driving on moving sticks and other objects, as well as the fact that the terrain changed during the run. Furthermore, the controller server was able to handle the lossy odometry in an environment with moving objects, most notably when driving through vegetation, as can be seen in the left picture in Figure 3.5.

**Figure 4.11:** Motion trail of WM-RL controller following a path in forest. Run 2a.

**Table 4.11:** Average performance over the forest backtrack runs. All runs follow the same course in the same direction, though not exactly the same trajectory, and the surface changed across runs, since the car displaced sticks and similar debris as it drove. The individual runs are given both in tabular form and as birds-eye-view plots in appendix Section B.6 and Section B.5. The runs included in the average are 1, 2a, 3 and 4a. Runs 2b and 4b are excluded, since both were driven in the reverse direction of the course, with 2b being the reverse of run 2a.

| Metric | Value |
|---|---|
| Time | 61.83 s |
| Mean CTE | 0.073 m |
| Max CTE | 0.421 m |
| Std CTE | 0.069 m |
| RMS Steering | 0.381 |
| RMS Speed | 0.183 |
| Max $a$ | 0.195 m/s$^2$ |
| RMS $a$ | 0.037 m/s$^2$ |
| Max $\frac{da}{dt}$ | 0.077 m/s$^3$ |
| RMS $\frac{da}{dt}$ | 0.015 m/s$^3$ |

### 4.2.2 Discussion of WM-RL and Pure Pursuit behaviour

The real-world results suggest that the main advantage of the WM-RL controller over Pure Pursuit is not only lower tracking error, but greater behavioural flexibility. Pure Pursuit is limited by its geometric formulation, the steering command depends strongly on the chosen lookahead distance, see Eq 2.2, which must balance stability against corner-cutting, see Sections 2.1.2.3 and 2.1.2.4. This makes the baseline sensitive to tuning and partly explains why its performance varies between path geometries. The same problem of tuning holds for adaptive pure pursuit, see Section 2.1.2.4. This was the actual algorithm implemented but was reduced to the vanilla pure pursuit, since no longitudinal control was developed.

The WM-RL controller is less tied to a fixed geometric rule. Since it outputs both steering and throttle, it can adapt speed, stop, reverse, and perform corrective manoeuvres when the vehicle pose becomes unfavourable. It also learned to reverse in order to adjust its orientation when a target could not be reached directly because of the vehicle's minimum turning radius. This can be seen in Figure 4.12, and partly in Figures 4.10 and 4.6. These behaviours were not explicitly programmed, and can therefore be interpreted as emergent recovery strategies learned through interaction with the world model.

However, the WM-RL controller also shows a clear failure mode. Its short reverse manoeuvres are useful, but it struggles with longer reversing sections, seen in Figure 4.7 and 4.6. Instead of continuing to reverse over a longer distance, the vehicle often turns around and proceeds forward. This indicates that the learned policy has a bias toward forward driving. Possible reasons are that forward motion dominates the training data, is easier for the world model to predict, and gives more useful camera observations than reverse motion. Longer reverse driving may therefore be underrepresented both in the learned dynamics and in the policy's preferred action distribution. Therefore, one area where the PP outperforms the WM-RL is long paths where only reversing is possible.

Terrain also affects the comparison. Pure Pursuit assumes the kinematic bicycle model, see Section 2.1.1. On grass, sand, or uneven surfaces where slip and rolling resistance matter a dynamics model would be needed to account for the forces. The WM-RL controller can in principle be more robust to such effects because it is trained on real sensor and control data rather than on a fixed kinematic model. However, since the current experiments do not isolate terrain effects from other sources of variation, it is unclear how much this contributed to the observed performance differences.

**Figure 4.12:** Orientation correction via reverse motion. The vehicle enters the frame from the right while turning right in the ego-centric frame. It then reverses while slightly turning left, before continuing forwards with an adjusted heading.

## 4.3 Sources of measurement error

The main source of uncertainty in the experimental results is the pose estimate produced by the visual-inertial SLAM system. The evaluation assumes that the estimated vehicle pose is a reliable representation of the real vehicle motion. In practice, this assumption was not always valid. In addition to gradual drift, the SLAM/VIO estimate occasionally produced large discontinuous jumps in the transform between the coordinate frames `odom` and `base_link`. This transform should vary smoothly during normal vehicle motion, since the physical vehicle cannot instantaneously change position or orientation. When such jumps occur, the measured trajectory no longer represents the true vehicle trajectory.

In the comparison between the WM-RL controller and the PP controller, identical reference paths were given, except in the case of backtracking. When backtracking, the paths were manually recorded during separate runs, even though the plugin architecture allowed for switching controllers during the same run. This is something that could be improved on if doing more evaluation.

Note the motion blur on the blue swing horse shaped in the background in Figure 4.10. It is a result of SAM3 believing it is the vehicle. This shows the errors in these motion blur images resulting from not keeping the camera completely still. Therefore the images does not exactly represent the true trajectory, but is able to capture important features.

# 5

# Conclusion

## 5.1 Answers to the research questions

The first research question asks how accurately a world model trained on real sensor and control data from a small off-road UGV predicts short-horizon vehicle motion across different terrain conditions, and how prediction accuracy varies between terrains and observation modalities. The 20-step open-loop rollout results, corresponding to 0.66 seconds of motion, show that the world model predicts short-horizon motion with low position error on all tested surfaces, as shown in Table 4.1. The lower error on the carpeted floor suggests that more uniform terrain produces more predictable vehicle motion, while the slightly higher errors on grass and gravel indicate that less uniform outdoor surfaces introduce greater variation in the vehicle response. The observation-masking results show that vector observations are more important than image observations in the current dataset. Masking image observations causes only small RMSE increases, while masking vector observations causes a larger degradation, especially on grass, where the RMSE increases from 0.0262 m to 0.0938 m. Overall, the world model predicts short-horizon vehicle motion well across the tested terrains, but the results suggest that the learned dynamics rely mainly on odometry- and IMU-based vector observations rather than on visual terrain cues.

The second research question asks whether a controller trained through imagined rollouts in the learned world model can perform real-world path following when provided with a reference path and pose estimate. The real-world trajectory-tracking results indicate that this is achieved within the tested conditions. The WM-RL controller is able to follow several reference paths tested on grass, including constant-curvature, clothoid, oscillating, and reversing-based paths. The controller reaches low CTEs across the evaluated paths and is also able to recover from unfavourable poses by reversing and readjusting its position. In addition it was able to follow oscillating and a clothoid turn on sand. On several tasks, it reaches and holds its upper speed target for large parts of the path, such as the C and S0.3 paths. It also drives smoothly on tasks that do not involve multiple readjustments. Furthermore, the controller performs well on the sand and forest backtracking tests. The forest-backtracking experiment extends the evaluation to a more irregular off-road environment than the main path-following tests. Although the training data mainly consisted of grass, sand, and gravel traversal, the controller was still able to complete the task, suggesting some transfer beyond the primary training conditions. These results support the conclusion that the learned world model provides a useful

training environment for transferring the policy to the physical UGV.

The third research question asks how the WM-RL controller compares to a PP controller. The WM-RL outperforms the PP in all evaluations regarding CTE. This performance may be attributed to its ability to constantly readjust by reversing to follow the path closely. On the one hand, this leads to close tracking. On the other hand, it may sometimes overconcentrate on the CTE and therefore produce rougher motions than desired by adjusting, see Section 4.2.1.7.

It is furthermore worth mentioning that longitudinal control is not implemented for PP, and therefore the adaptive part of adaptive pure pursuit is not used either. In addition, it is not tuned for the different tasks. In some scenarios, the PP could possibly achieve a similar CTE to the WM-RL but more smoothly if tuned. This could be evaluated in future work, along with more thoroughly benchmarking against state-of-the-art vehicle controllers. Nevertheless, the WM-RL shows that it can also drive smoothly on certain tasks: see the low-amplitude sine wave following in Table 4.8 and the circle tracking in Table 4.5.

The WM-RL also possesses a behavioral flexibility that the geometrically bounded PP lacks. As a result, the WM-RL can solve certain tasks that the PP cannot. Except for long-distance reversing, the experiments do not identify any task that the WM-RL cannot perform, highlighting the benefit of its adjusting behaviour. The PP, however, can handle longer reversing tasks in certain scenarios, for example see Figure B.24.

## 5.2 Contributions

- A flexible world-model-based reinforcement learning framework for vehicle control in which the world model is learned from real-world sensor and control data to predict possible vehicle movement. Since goal-directed path information and rewards are computed separately during imagined rollouts, the framework allows the initial actor policy to be trained without direct actor interaction with the physical environment, while retaining flexibility to modify path representations and reward formulations after data collection, enabling safe RL training.
- Showcased the possibilities of learning vehicle dynamics and control from a small amount of data for specific environments.
- A research platform to train and evaluate learning-based vehicle controllers, such as WM-RL-based ones.
  - A software stack based on ROS 2 that enables integration with other parts of the autonomous stack.
  - A solution for handling a jumping odometry frame.
  - A data pipeline converting ROS 2 bags into synchronized NumPy arrays.
  - Hardware capable of operating for over 40 minutes in challenging terrain without breaking when rolling over.
- A product that, with the addition of control over the internet, would be suitable for remote-control applications with autonomous backtracking.

## 5.3 Summary

Our work showcases the promise of WM-RL controllers and world-model-based methods in general. We were able to train a world model using a limited amount of data that was not collected through imitation learning for path following. Instead, the world model only learned the vehicle dynamics from the data. Apart from reward shaping, no parameter tuning was required, highlighting another advantage over supervised learning methods.

We also demonstrated that the sim-to-real gap can be bridged by learning a simulation within the world model rather than relying on a hand-built simulator. The controller learned to follow curves with varying properties on terrains such as sand and uneven grass. Furthermore, the WM-RL controller significantly outperformed the simple PP controller on all tested paths.

The WM-RL controller also exhibited signs of emergent behavior, such as reversing to adjust its position. Finally, the controller architecture shows considerable promise given the small amount of training data and modest model size used. Since inference times were low, the model could be scaled further.

A major limiting factor during the work was the robustness of the visual-inertial odometry used. This is one of the key areas that needs to be improved on in the

research platform. Still, the platform is ready to integrate more ROS 2 nodes containing other parts of the autonomous driving stack. The platform is ready for one real world application, driving manually and then backtracking if loosing connection.

The primary goal of developing a WM-RL based controller that can follow a diverse set of paths in challenging terrains was reached. The secondary goal of developing a research platform enabling machine learning based controllers and to further build on other parts of the autonomous stack such as planning has also been reached.

## 5.4 Future Work

In the current implementation, the actor and environment setup are not optimal for path-following control, since the path information is represented by a single waypoint. This limited representation leads to behavior in which the vehicle reaches the current waypoint without necessarily being favorably oriented toward subsequent path poses. A direct extension would be to provide the actor with a sequence of future waypoints expressed in the vehicle frame, or with alternative features that encode look-ahead information about the path. In combination with this modification, the current episode termination condition should be removed, since terminating the episode upon reaching a single waypoint would prevent the policy from learning behavior that accounts for subsequent path poses.

A useful direction for future work is to study the reward formulation more systematically. Several of the observed limitations of the WM-RL controller are related to what the actor is encouraged to do during imagined rollouts. In the current formulation, the controller is rewarded for reducing the distance to the next waypoint, while movement and changes in the action signal are penalized. This gives smoother actions, but it may also discourage large steering corrections, causing the controller to use repeated small adjustments or short reversing manoeuvres instead. This was especially visible on the high-amplitude sine wave and the cross pattern, where the controller tracked the path closely but sometimes appeared to rely too much on reversing or zig-zagging rather than taking a more direct forward-driving solution with a higher steering output.

Vehicle data was not collected while the actor was performing path following, since collecting a sufficiently broad set of vehicle–environment dynamics in this manner would be impractical, particularly when the actor is untrained or performs poorly. However, once a reasonably capable actor has been obtained, it may be valuable to investigate the use of actor-collected data, including the corresponding path information. Such data would provide the reinforcement learning algorithm with initial trajectories that already align with the current policy, potentially making it feasible to identify further improvements when performing imaginary environment rollouts. An iterative procedure where the actor is used to collect new data, followed by fine-tuning of both the world model and the actor, could potentially lead to progressively improved performance. This is more closely aligned with previous world-model-based reinforcement learning approaches [18, 2].

# Bibliography


[1] Mitchell Goff et al. *Learning to Drive from a World Model*. 2025. arXiv: 2504.19077 [cs.CV]. URL: https://arxiv.org/abs/2504.19077.

[2] David Ha and Jürgen Schmidhuber. *World Models*. 2018. DOI: 10.5281/ZENODO.1207631. URL: https://zenodo.org/record/1207631.

[3] Axel Brunnbauer et al. *Latent Imagination Facilitates Zero-Shot Transfer in Autonomous Racing*. 2022. arXiv: 2103.04909 [cs.LG]. URL: https://arxiv.org/abs/2103.04909.

[4] Zhenjie Yang et al. *Raw2Drive: Reinforcement Learning with Aligned World Models for End-to-End Autonomous Driving (in CARLA v2)*. 2025. arXiv: 2505.16394 [cs.RO]. URL: https://arxiv.org/abs/2505.16394.

[5] Huiqian Li et al. *Vehicle Dynamics Embedded World Models for Autonomous Driving*. 2025. arXiv: 2512.02417 [cs.RO]. URL: https://arxiv.org/abs/2512.02417.

[6] Qifeng Li et al. *Think2Drive: Efficient Reinforcement Learning by Thinking in Latent World Model for Quasi-Realistic Autonomous Driving (in CARLA-v2)*. 2024. arXiv: 2402.16720 [cs.RO]. URL: https://arxiv.org/abs/2402.16720.

[7] Sidharth Talia et al. "Demonstrating HOUND: A Low-cost Research Platform for High-speed Off-road Underactuated Nonholonomic Driving". In: *arXiv preprint arXiv:2311.11199* (2024). URL: https://arxiv.org/pdf/2311.11199.

[8] R. Craig Coulter. *Implementation of the Pure Pursuit Path Tracking Algorithm*. Tech. rep. CMU-RI-TR-92-01. Pittsburgh, PA: The Robotics Institute, Carnegie Mellon University, 1992. URL: https://publications.ri.cmu.edu/storage/publications/pub_files/pub3/coulter_r_craig_1992_1/coulter_r_craig_1992_1.pdf.

[9] Pierre Bouguer. "Sur de nouvelles courbes auxquelles on peut donner le nom de lignes de poursuite". French. In: *Mémoires de mathématique et de physique tirés des registres de l'Académie royale des sciences* (1732), pp. 1–15.

[10] N. A. Shneydor. "Chapter 3 - Pure Pursuit". In: *Missile Guidance and Pursuit: Kinematics, Dynamics and Control*. Elsevier, 1998, pp. 47–76. ISBN: 978-1904275374. DOI: https://doi.org/10.1533/9781782420590.47.

[11] Hiroki Ohta et al. “Pure Pursuit Revisited: Field Testing of Autonomous Vehicles in Urban Areas”. In: *2016 4th IEEE International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA)*. IEEE. 2016, pp. 7–12. DOI: 10.1109/CPSNA.2016.10.

[12] Stefan F. Campbell. “Steering Control of an Autonomous Ground Vehicle with Application to the DARPA Urban Challenge”. MA thesis. Cambridge, MA: Massachusetts Institute of Technology, Sept. 2007.

[13] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2nd ed. Cambridge, MA: MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

[14] Joshua Achiam. *Spinning Up in Deep Reinforcement Learning*. https://spinningup.openai.com/. 2018.

[15] Volodymyr Mnih et al. “Asynchronous Methods for Deep Reinforcement Learning”. In: *Proceedings of the 33rd International Conference on Machine Learning (ICML)*. Vol. 48. PMLR, 2016, pp. 1928–1937. URL: https://proceedings.mlr.press/v48/mniha16.html.

[16] Georg B. Keller, Tobias Bonhoeffer, and Mark Hübener. “Sensorimotor Mismatch Signals in Primary Visual Cortex of the Behaving Mouse”. In: *Neuron* 74.5 (2012), pp. 809–815. ISSN: 0896-6273. DOI: https://doi.org/10.1016/j.neuron.2012.03.040. URL: https://www.sciencedirect.com/science/article/pii/S0896627312003844.

[17] Danijar Hafner et al. *Learning Latent Dynamics for Planning from Pixels*. 2018. arXiv: 1811.04551. URL: http://arxiv.org/abs/1811.04551.

[18] Danijar Hafner et al. *Mastering Diverse Domains through World Models*. 2024. arXiv: 2301.04104 [cs.AI]. URL: https://arxiv.org/abs/2301.04104.

[19] Redcat Racing. *Redcat Gen8 V2 International Scout II 1/10 Electric RC Scale Crawler*. Accessed: June 2, 2026. URL: https://www.redcatracing.com/products/redcat-gen8-v2-scout-ii-1-10-electric-rc-scale-crawler.

[20] Steven Macenski et al. “Robot Operating System 2: Design, architecture, and uses in the wild”. In: *Science Robotics* 7.66 (2022), eabm6074. DOI: 10.1126/scirobotics.abm6074. URL: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[21] Alexander Korovko et al. *cuVSLAM: CUDA accelerated visual odometry and mapping*. 2025. arXiv: 2506.04359 [cs.RO]. URL: https://arxiv.org/abs/2506.04359.

[22] NVIDIA Corporation. *isaac_ros_visual_slam_realsense.launch.py (Release 3.2)*. GitHub repository. 2024. URL: https://github.com/NVIDIA-ISAAC-ROS/isaac_ros_visual_slam/blob/release-3.2/isaac_ros_visual_slam/launch/isaac_ros_visual_slam_realsense.launch.py.

[23] NVIDIA Corporation. *Isaac ROS Visual SLAM Documentation (Release 3.2)*. Accessed: June 2, 2026. 2024. URL: https://nvidia-isaac-ros.github.io/v/release-3.2/repositories_and_packages/isaac_ros_visual_slam/index.html.

[24] Intel RealSense. *Intel RealSense ROS 2 Wrapper Documentation*. Accessed: June 2, 2026. 2026. URL: https://dev.realsenseai.com/docs/ros2-wrapper/.

[25] Open Source Robotics Foundation. *nav_msgs/msg/Path Message Definition (ROS 2 Humble)*. Accessed: June 2, 2026. 2022. URL: https://github.com/ros2/common_interfaces/blob/humble/nav_msgs/msg/Path.msg.

[26] Steve Macenski et al. "The Marathon 2: A Navigation System". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020. URL: https://github.com/ros-planning/navigation2.

[27] Simple DirectMedia Layer Developers. *Simple DirectMedia Layer (SDL) 2.0 Source Tree*. GitHub repository. 2026. URL: https://github.com/libsdl-org/SDL/tree/SDL2.

[28] Ternaris. *Rosbags: Pure Python library for ROS bag files*. Accessed: June 3, 2026. 2026. URL: https://gitlab.com/ternaris/rosbags.

[29] Wikipedia contributors. *Euler spiral*. https://en.wikipedia.org/wiki/Euler_spiral. Accessed: June 2026.

[30] Enrico Bertolazzi and Marco Frego. *Clothoids: A C++ library for clothoids, G1 and G2 fitting, and clothoid splines*. GitHub repository. Accessed: June 2026. 2025. URL: https://github.com/ebertolazzi/Clothoids.

[31] Nicolas Carion, Laura Gustafson, et al. *SAM 3: Segment Anything with Concepts*. 2025. arXiv: 2511.16719 [cs.CV]. URL: https://arxiv.org/abs/2511.16719.

[32] Wikipedia contributors. *Principal component analysis*. https://en.wikipedia.org/wiki/Principal_component_analysis. Accessed: 16 June 2026.

[33] Wikipedia contributors. *Rodrigues' rotation formula*. https://en.wikipedia.org/wiki/Rodrigues%27_rotation_formula. Accessed: 16 June 2026.

# A

# Appendix Theory

## A.1 Principal component analysis

To correct the projection twist caused by odometry drift and jumps, the recorded trajectory was sometimes assumed to be approximately planar. The best-fit plane was estimated using principal component analysis (PCA), implemented via singular value decomposition (SVD) of the mean-centred trajectory. The singular vector corresponding to the smallest singular value defines the plane normal [32]. The rotation that aligns this normal with the global vertical axis was computed using Rodrigues' rotation formula [33] and applied about the trajectory centroid to both position estimates and vehicle orientations, producing the levelled BEV used for visualisation.

# B

# Appendix Data

## B.1 Reference paths

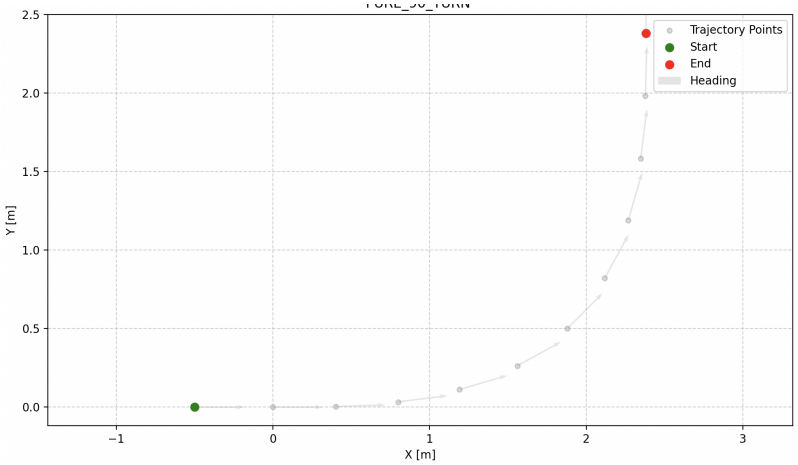



**Figure B.1:** Clothoid 90 degree turn (E90L) with a minimum radius of 2 m and 0.5 m added before the actual turn starts. The spacing between points is 0.3 m.

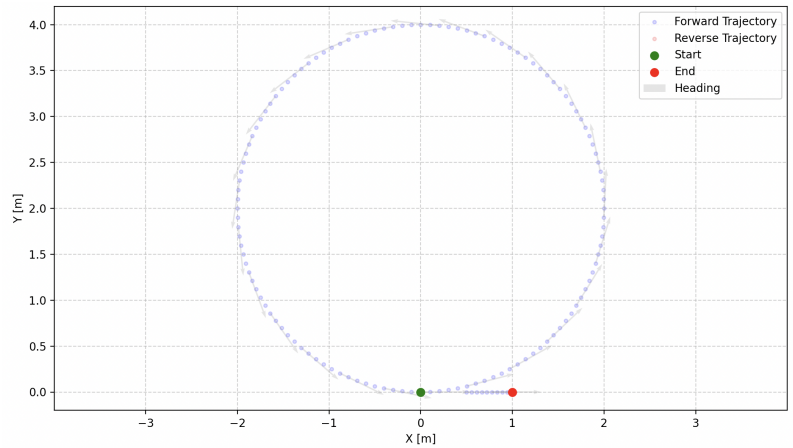



**Figure B.2:** Circle path (C), left turn, with a radius of 2.0 m. 1.0 m appended to not start and finish at same location. 0.1 m step size.

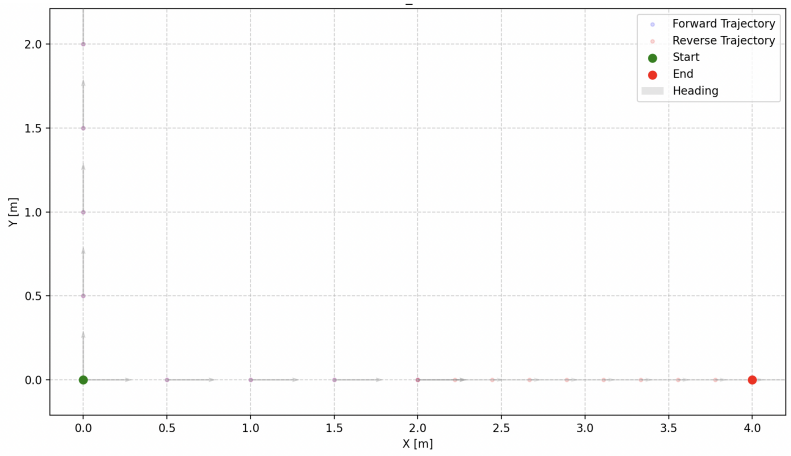



**Figure B.3:** Cross pattern (X) with arm length 2.0 m and 4.0 m appended to not start and finish at same location. 0.4 m step size.

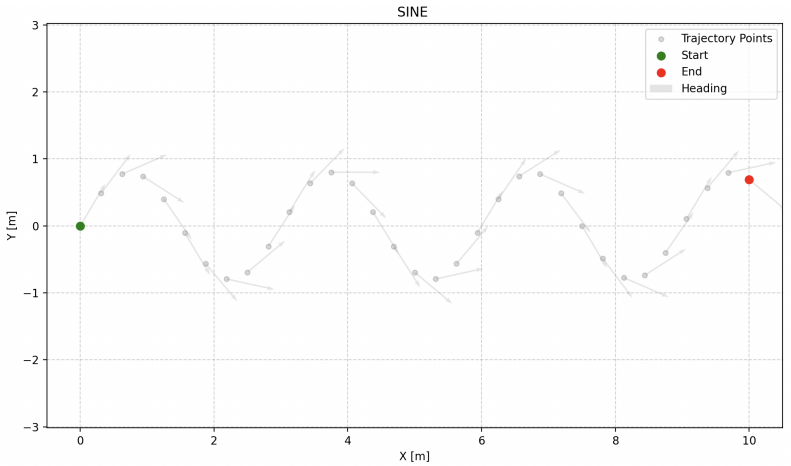



**Figure B.4:** High amplitude sine wave (S0.8) with 0.8 m amplitude and a wavelength of 3.0 m. Total length is 10 m and step size 0.3 m.

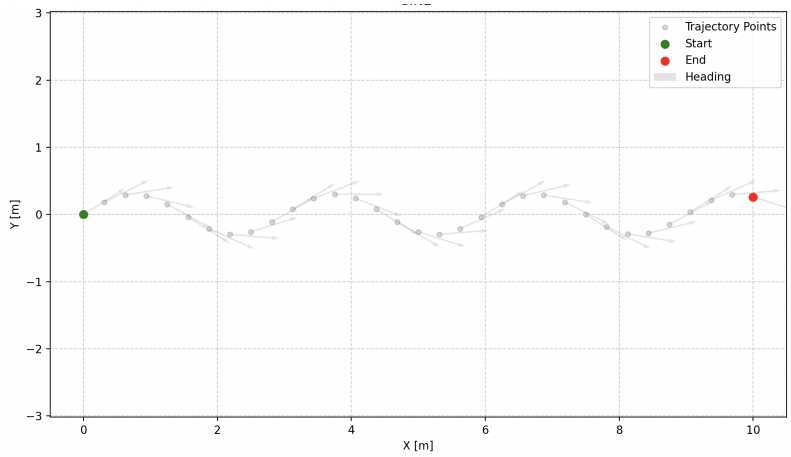



**Figure B.5:** Low amplitude sine wave (S0.3) with 0.3 m amplitude and a wavelength of 3.0 m. Total length is 10 m and step size 0.3 m.

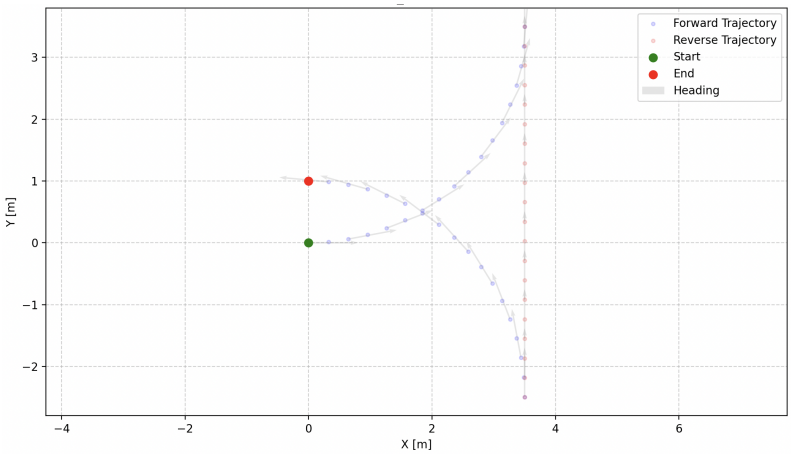



**Figure B.6:** K-turn with radius 3.5 m. 0.3 m step size.

## B.2 Plots from tracked reference paths on grass

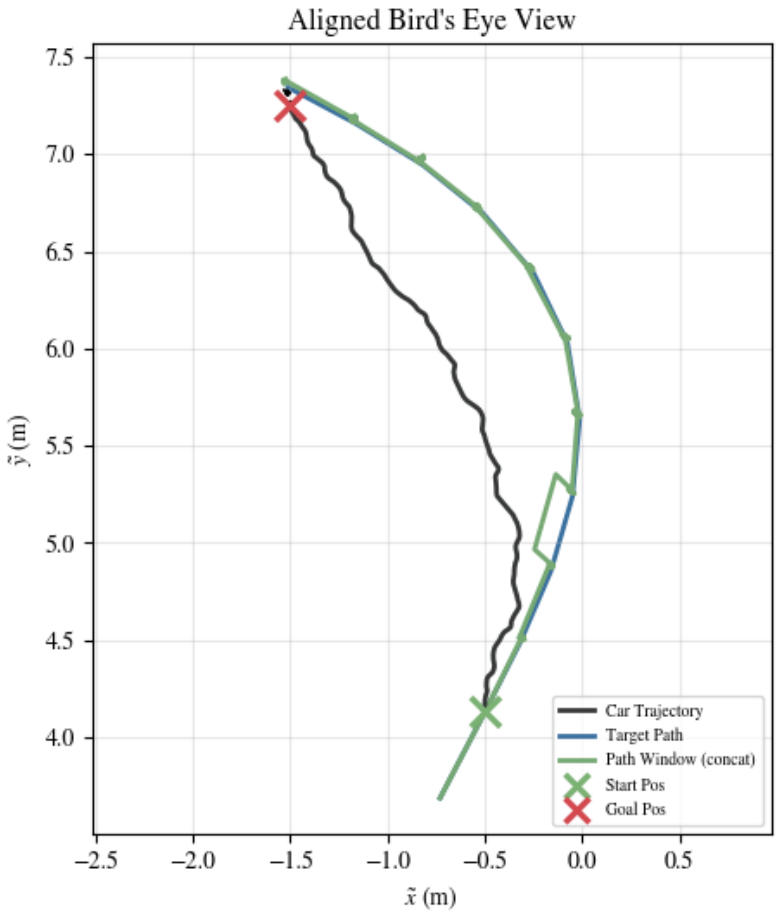



**Figure B.7:** PP traversing path E90L on grass. Motion trail showing the vehicle trajectory (black) from start (green cross) to end (red cross), the odometry-adjusted followed points (green), and the reference path (blue).

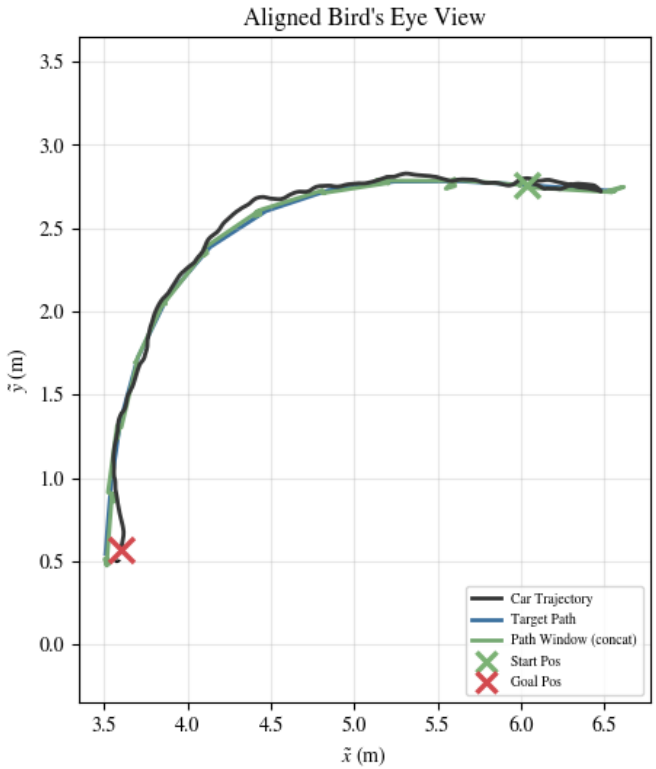



**Figure B.8:** WM-RL traversing path E90L on grass. Motion trail showing the vehicle trajectory (black) from start (green cross) to end (red cross), the odometry-adjusted followed points (green), and the reference path (blue).

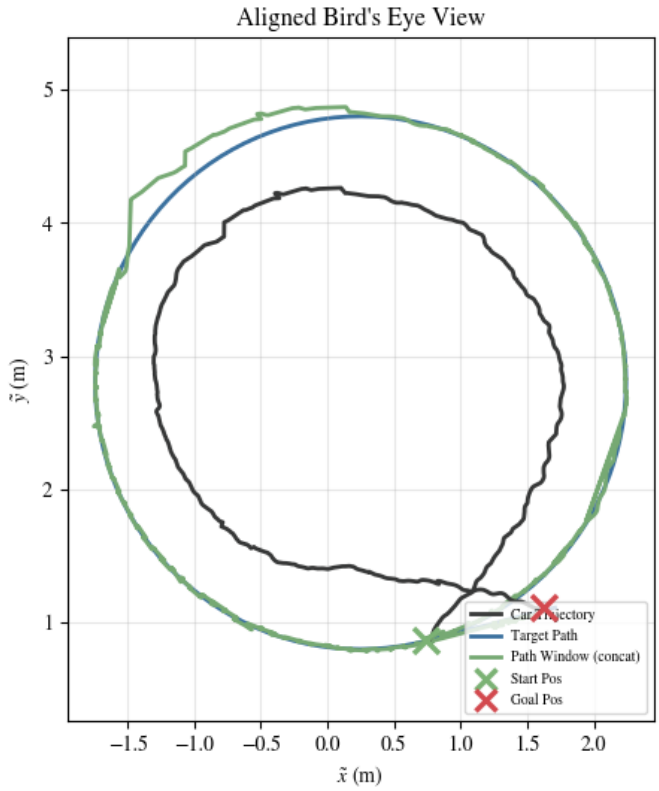



**Figure B.9:** PP traversing path C on grass. Motion trail showing the vehicle trajectory (black) from start (green cross) to end (red cross), the odometry-adjusted followed points (green), and the reference path (blue).

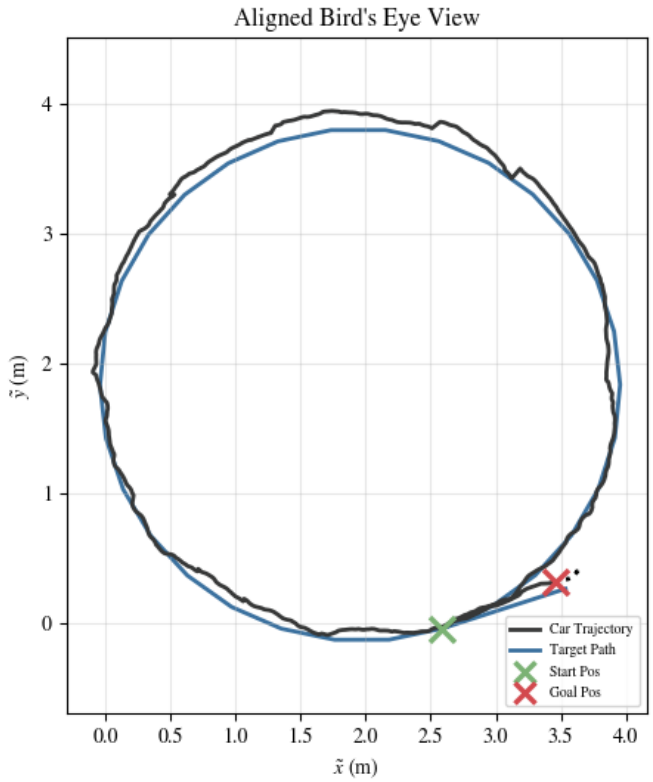



**Figure B.10:** WM-RL traversing path C on grass. Motion trail showing the vehicle trajectory (black) from start (green cross) to end (red cross), and the reference path (blue).

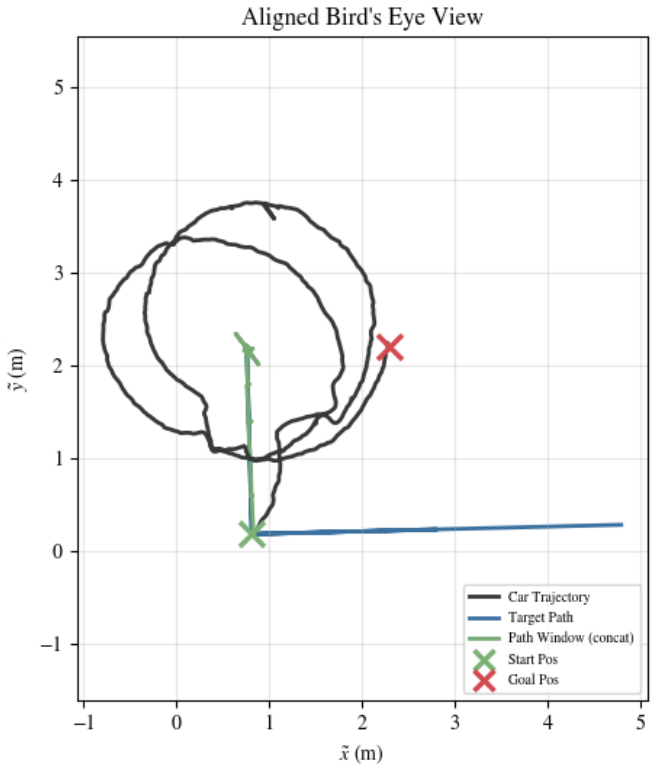



**Figure B.11:** PP traversing path X on grass. Motion trail showing the vehicle trajectory (black) from start (green cross) to end (red cross), the odometry-adjusted followed points (green), and the reference path (blue).

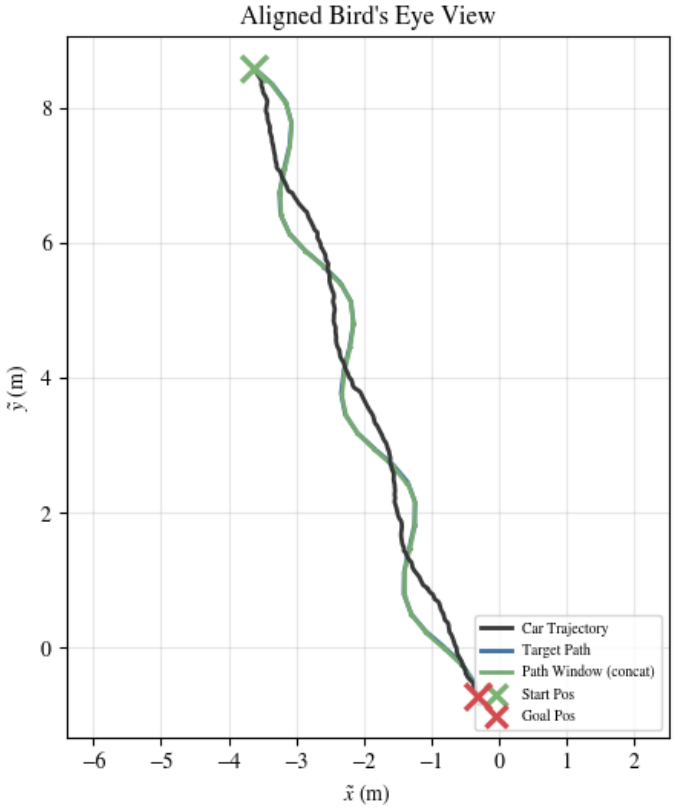



**Figure B.12:** PP traversing path S0.3 on grass. Motion trail showing the vehicle trajectory (black) from start (green cross) to end (red cross), the odometry-adjusted followed points (green), and the reference path (blue).

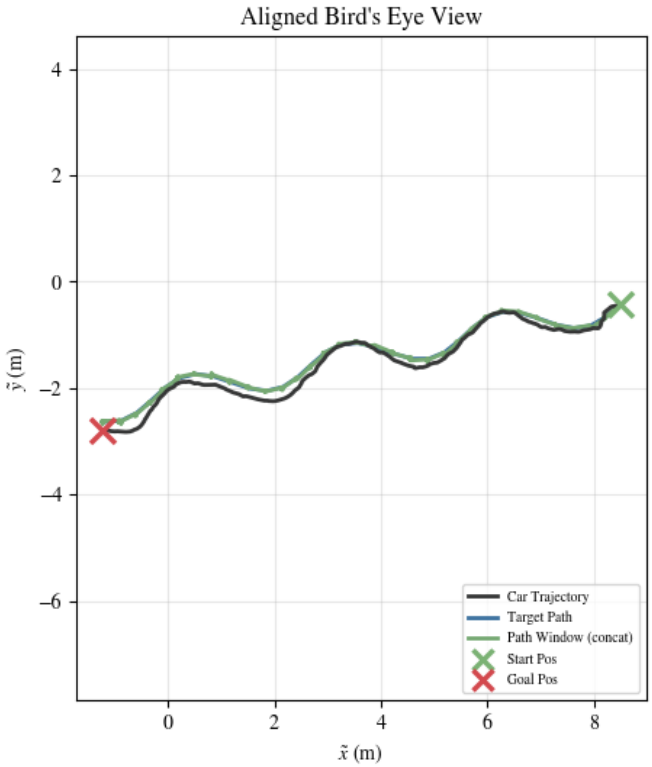



**Figure B.13:** WM-RL traversing path S0.3 on grass. Motion trail showing the vehicle trajectory (black) from start (green cross) to end (red cross), the odometry-adjusted followed points (green), and the reference path (blue).

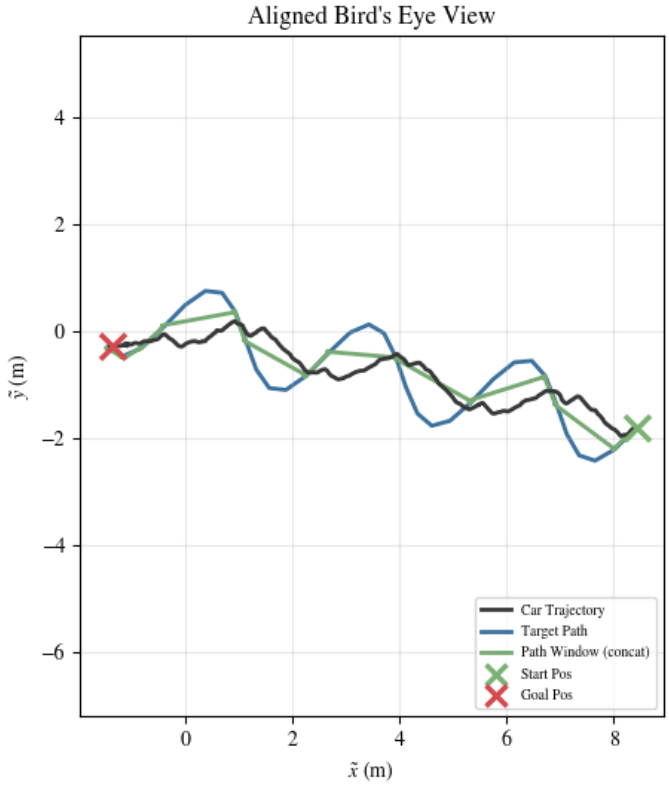



**Figure B.14:** PP traversing path S0.8 on grass. Motion trail showing the vehicle trajectory (black) from start (green cross) to end (red cross), the odometry-adjusted followed points (green), and the reference path (blue).

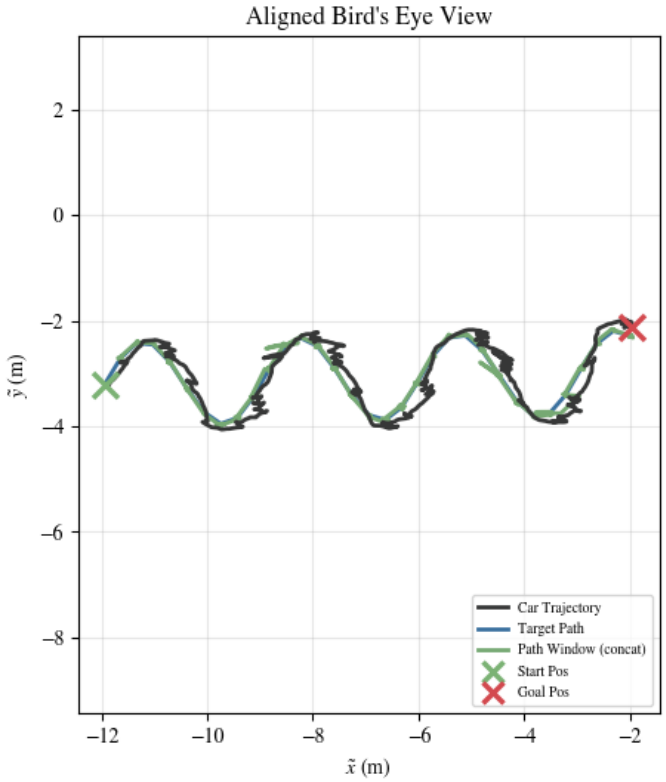



**Figure B.15:** WM-RL traversing path S0.8 on grass. Motion trail showing the vehicle trajectory (black) from start (green cross) to end (red cross), the odometry-adjusted followed points (green), and the reference path (blue).

## B.3 Plots from tracked reference paths on sand

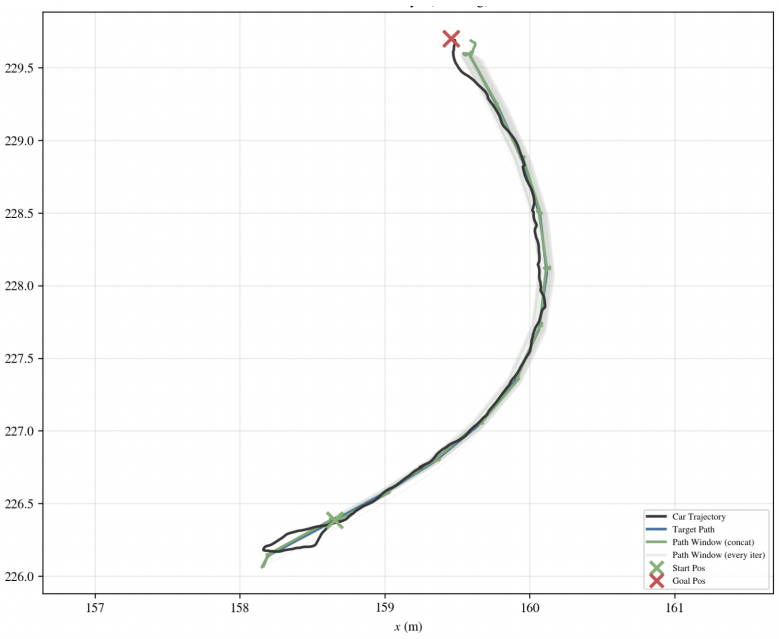



**Figure B.16:** WM-RL following E90L reference path on sand. Run 1.

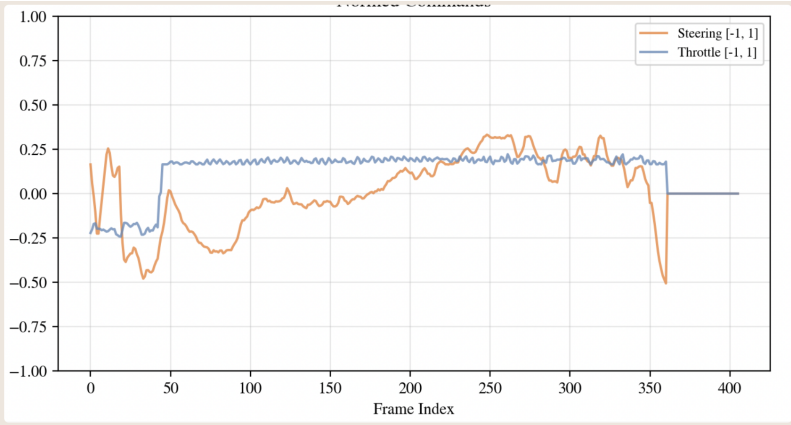



**Figure B.17:** WM-RL following E90L reference path on sand. Normed control signals, steering in yellow and throttle in blue. Run 1.

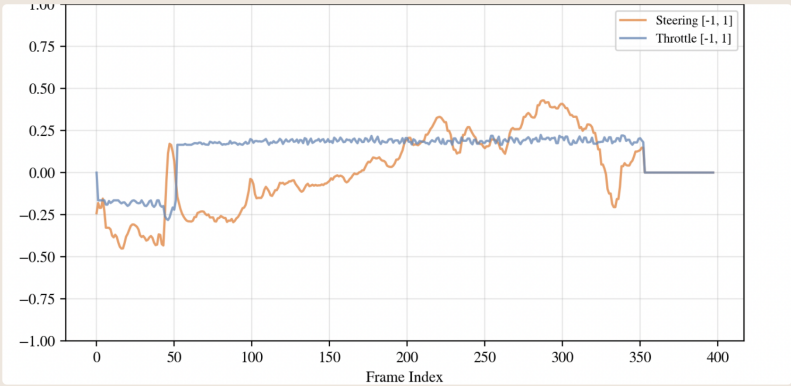



**Figure B.18:** WM-RL following E90L reference path on sand. Normed control signals, steering in yellow and throttle in blue. Run 2.

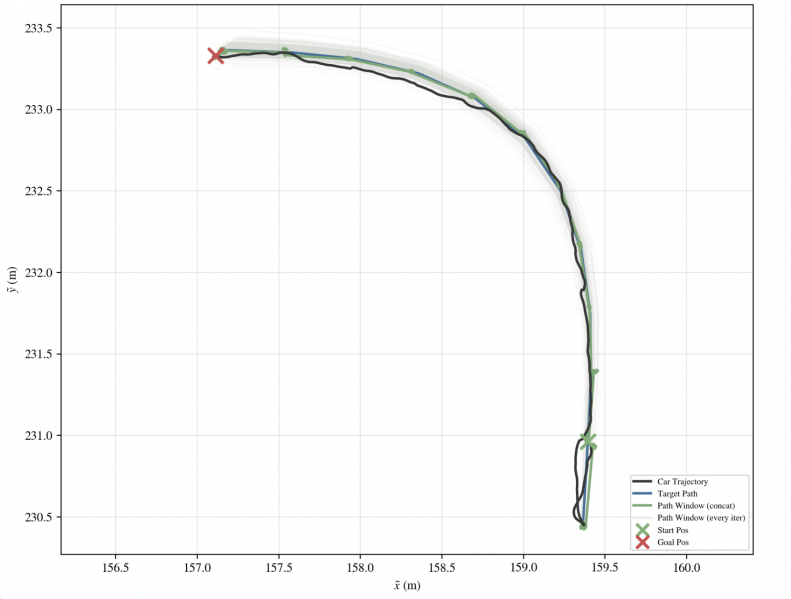



**Figure B.19:** WM-RL following E90L reference path on sand. Run 3.

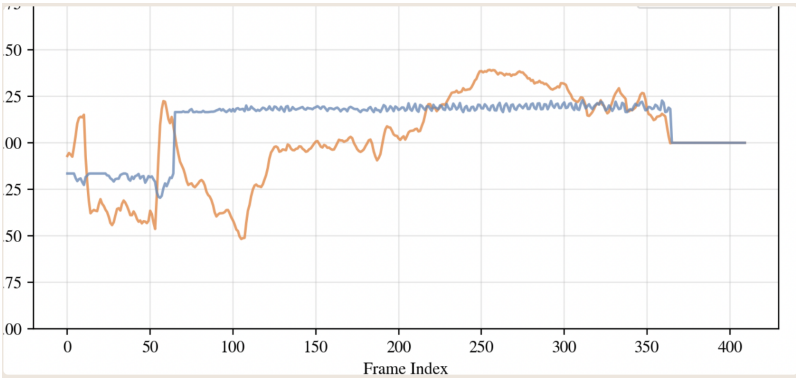



**Figure B.20:** WM-RL following E90L reference path on sand. Normed control signals, steering in yellow and throttle in blue. Run 3.

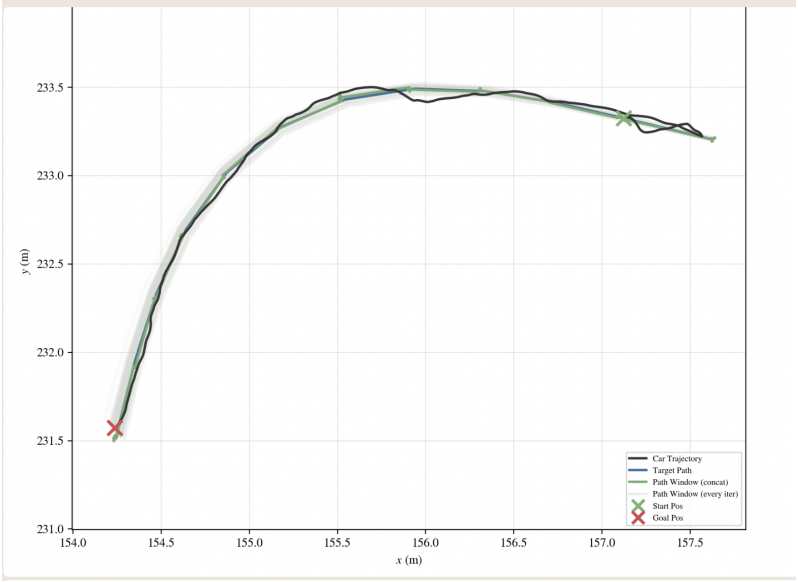



**Figure B.21:** WM-RL following E90L reference path on sand. Run 4.

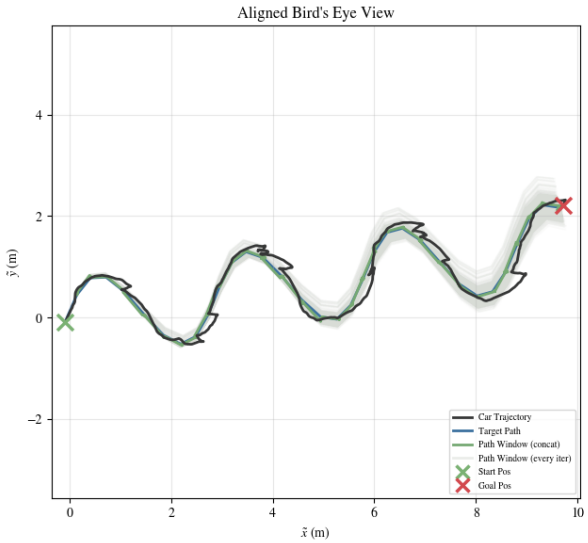



**Figure B.22:** WM-RL traversing path S0.8 on sand. Motion trail showing the vehicle trajectory (black) from start (green cross) to end (red cross), the odometry-adjusted followed points (green), and the reference path (blue).

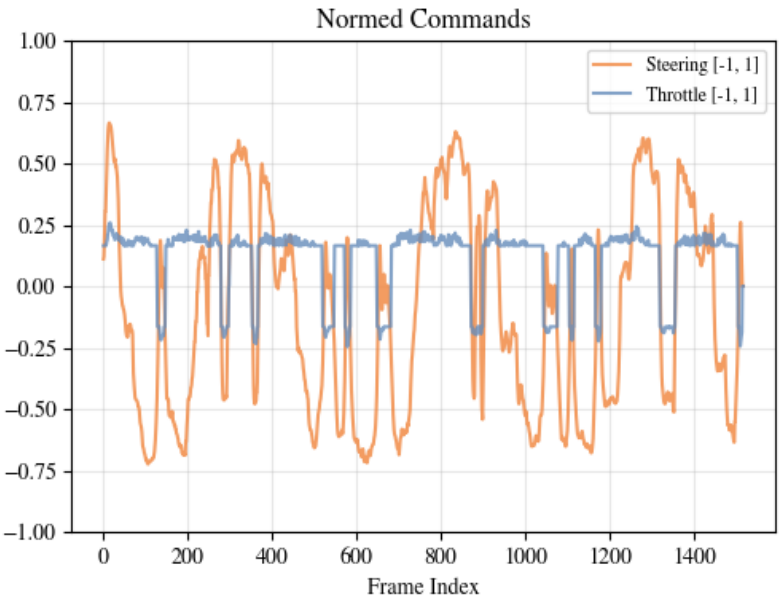



**Figure B.23:** Normed control outputs for WM-RL controller performing path following on S0.8 (sand).

## B.4 Performance metrics from tracked reference paths on sand

**Table B.1:** Performance metrics for WM-RL E90L sand, run 1.

| Metric | Value |
|---|---|
| Time | 13.51 s |
| Mean CTE | 0.029 m |
| Max CTE | 0.115 m |
| Std CTE | 0.030 m |
| RMS Steering | 0.198 |
| RMS Speed | 0.176 |
| Max $a$ | 0.101 m/s$^2$ |
| RMS $a$ | 0.019 m/s$^2$ |
| Max $\frac{da}{dt}$ | 0.050 m/s$^3$ |
| RMS $\frac{da}{dt}$ | 0.009 m/s$^3$ |

**Table B.2:** Performance metrics for WM-RL E90L sand, run 2.

| Metric | Value |
|---|---|
| Time | 13.24 s |
| Mean CTE | 0.017 m |
| Max CTE | 0.063 m |
| Std CTE | 0.015 m |
| RMS Steering | 0.221 |
| RMS Speed | 0.177 |
| Max $a$ | 0.078 m/s$^2$ |
| RMS $a$ | 0.017 m/s$^2$ |
| Max $\frac{da}{dt}$ | 0.042 m/s$^3$ |
| RMS $\frac{da}{dt}$ | 0.008 m/s$^3$ |

**Table B.3:** Performance metrics for WM-RL E90L sand, run 3.

| Metric | Value |
|---|---|
| Time | 13.64 s |
| Mean CTE | 0.026 m |
| Max CTE | 0.099 m |
| Std CTE | 0.023 m |
| RMS Steering | 0.239 |
| RMS Speed | 0.178 |
| Max $a$ | 0.058 m/s$^2$ |
| RMS $a$ | 0.016 m/s$^2$ |
| Max $\frac{da}{dt}$ | 0.024 m/s$^3$ |
| RMS $\frac{da}{dt}$ | 0.007 m/s$^3$ |

**Table B.4:** Performance metrics for WM-RL E90L sand, run 4.

| Metric | Value |
|---|---|
| Time | 12.91 s |
| Mean CTE | 0.022 m |
| Max CTE | 0.067 m |
| Std CTE | 0.014 m |
| RMS Steering | 0.193 |
| RMS Speed | 0.180 |
| Max $a$ | 0.048 m/s$^2$ |
| RMS $a$ | 0.015 m/s$^2$ |
| Max $\frac{da}{dt}$ | 0.017 m/s$^3$ |
| RMS $\frac{da}{dt}$ | 0.006 m/s$^3$ |

**Table B.5:** Performance metrics for WM-RL following path S0.8 on sand.

| Metric | Value |
|---|---|
| Time | 52.01 s |
| Mean CTE | 0.103 m |
| Max CTE | 0.579 m |
| Std CTE | 0.092 m |
| RMS Steering | 0.423 |
| RMS Speed | 0.178 |
| Max $a$ | 0.083 m/s$^2$ |
| RMS $a$ | 0.018 m/s$^2$ |
| Max $\frac{da}{dt}$ | 0.024 m/s$^3$ |
| RMS $\frac{da}{dt}$ | 0.006 m/s$^3$ |

## B.5 Plots from backtracked paths

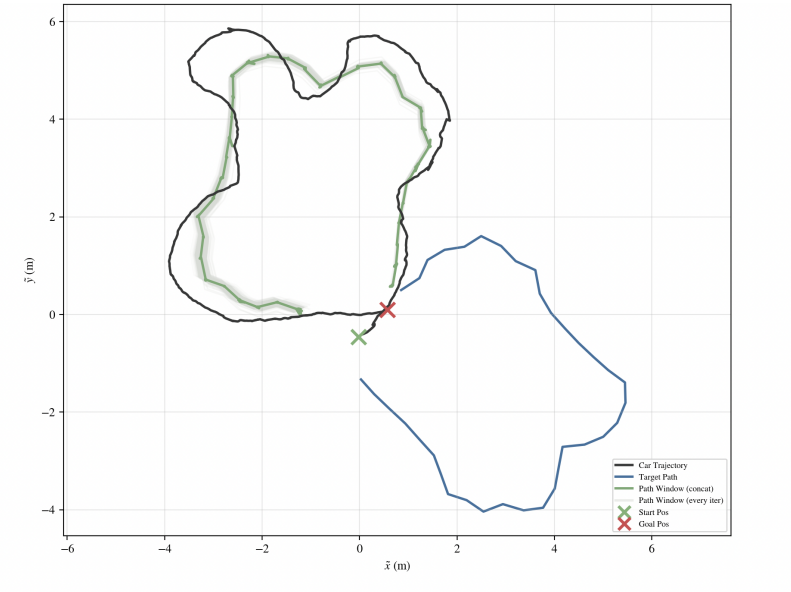



**Figure B.24:** PP traversing around the sandbox. Motion trail showing the vehicle trajectory (black) from start (green cross) to end (red cross), the odometry-adjusted followed points (green), and the reference path (blue).

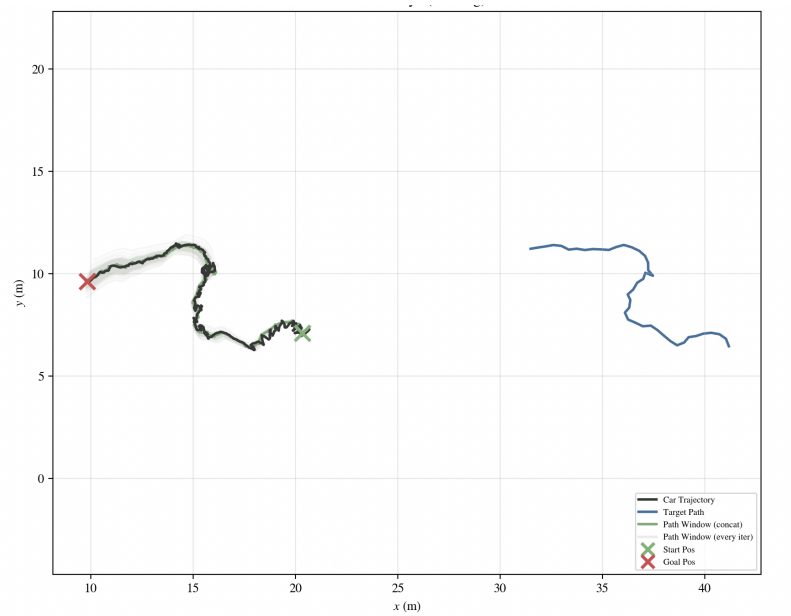



**Figure B.25:** WM-RL backtracking in forest, run 1. Motion trail showing the vehicle trajectory (black) from start (green cross) to end (red cross), the odometry-adjusted followed points (green), and the reference path (blue).

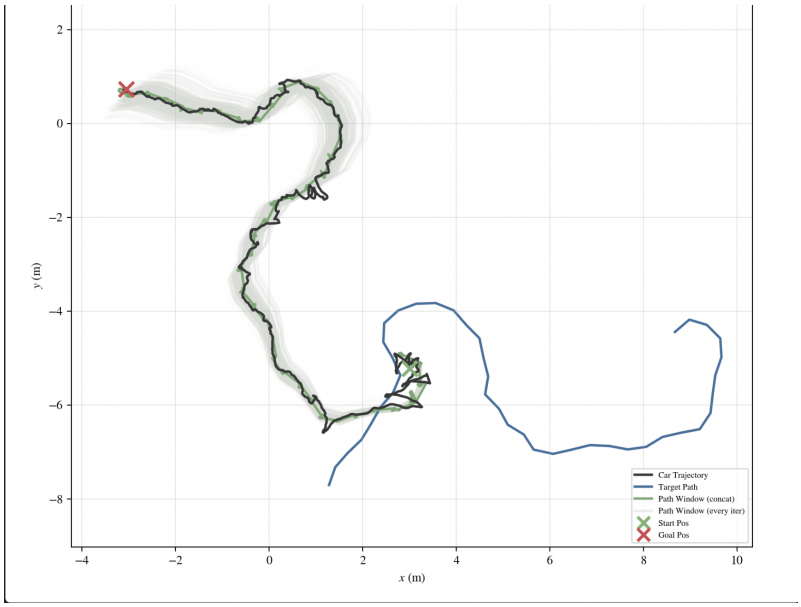



**Figure B.26:** WM-RL backtracking in forest, run 2a. Motion trail showing the vehicle trajectory (black) from start (green cross) to end (red cross), the odometry-adjusted followed points (green), and the reference path (blue).

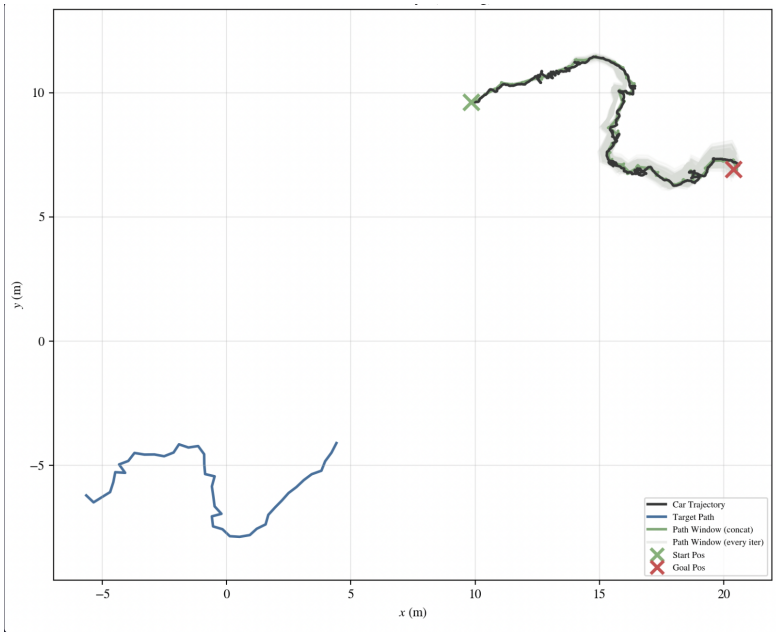



**Figure B.27:** WM-RL backtracking in forest, run 2b. Motion trail showing the vehicle trajectory (black) from start (green cross) to end (red cross), the odometry-adjusted followed points (green), and the reference path (blue).

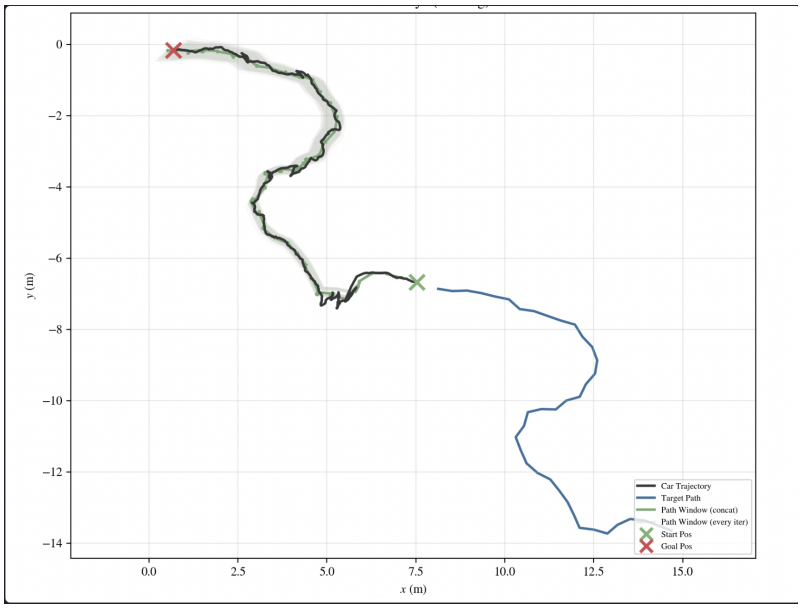



**Figure B.28:** WM-RL backtracking in forest, run 3. Motion trail showing the vehicle trajectory (black) from start (green cross) to end (red cross), the odometry-adjusted followed points (green), and the reference path (blue).

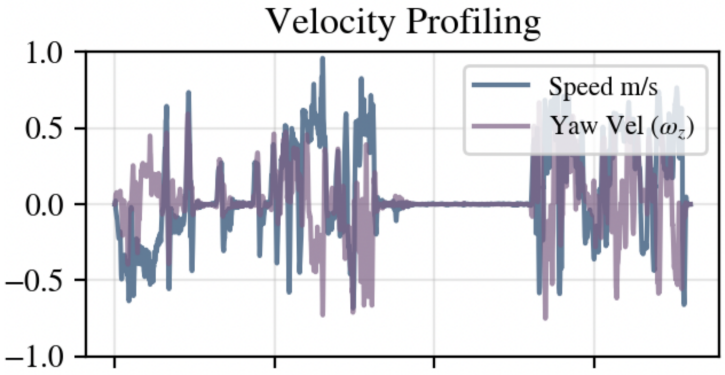



**Figure B.29:** Velocities of run 2b. The blue line is the the total 2D velocity $\sqrt{v_x^2 + v_y^2}$ in m/s and the purple is yaw rate in rad/s.

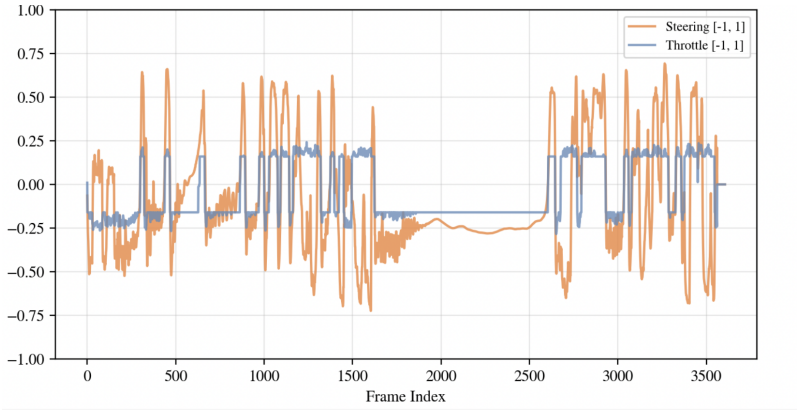



**Figure B.30:** WM-RL backtracking forest. Steering signals for run 2b.

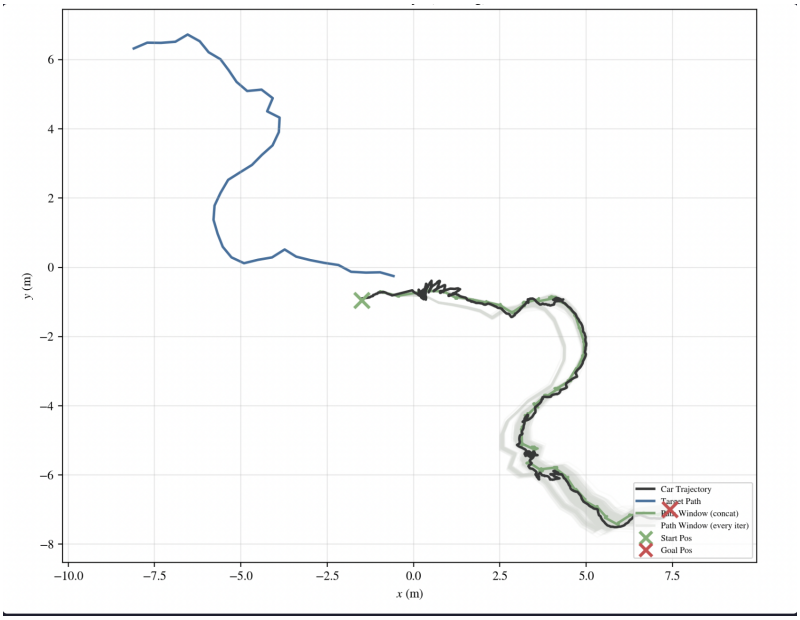



**Figure B.31:** WM-RL backtracking in forest, run 4a. Motion trail showing the vehicle trajectory (black) from start (green cross) to end (red cross), the odometry-adjusted followed points (green), and the reference path (blue).

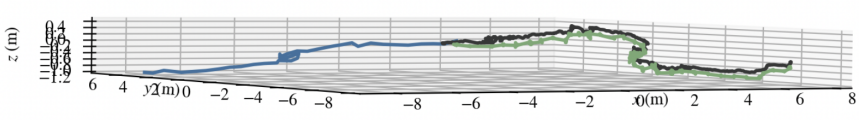



**Figure B.32:** Run 4a but in 3D showing what appears to be a correct elevation and slope illustration of the course.

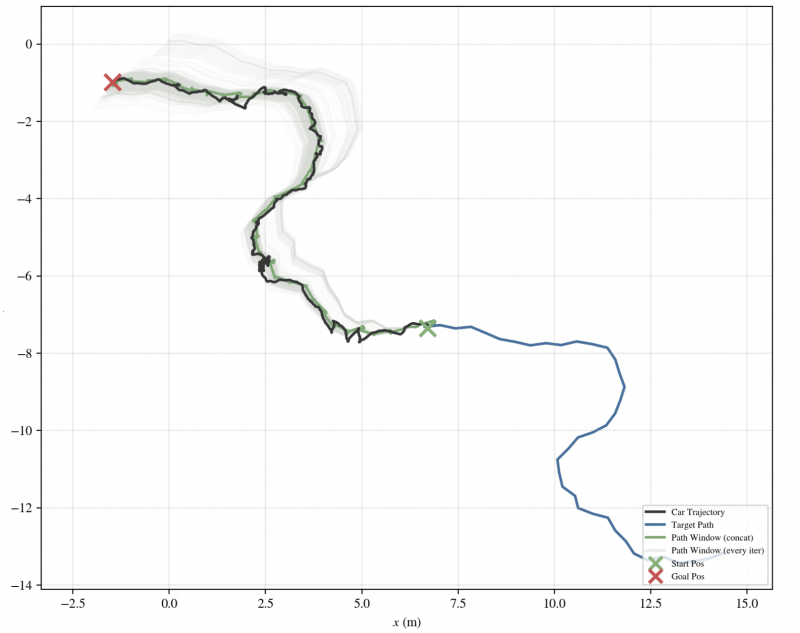



**Figure B.33:** WM-RL backtracking in forest, run 4b. Backtracking the backtrack of run 4a in Figure B.31. Motion trail showing the vehicle trajectory (black) from start (green cross) to end (red cross), the odometry-adjusted followed points (green), and the reference path (blue).

## B.6 Performance metrics backtrack

**Table B.6:** PP traversing around the sandbox.

| Metric | Value |
|---|---|
| Mean CTE | 0.403 |
| Max CTE | 1.210 |
| Std CTE | 0.262 |
| RMS Steering | 0.493 |
| RMS Speed | 0.182 |
| Max Accel | 0.096 $m/s^2$ |
| RMS Accel | 0.014 $m/s^2$ |
| Max Jerk | 0.046 $m/s^3$ |
| RMS Jerk | 0.006 $m/s^3$ |

**Table B.7:** Performance metrics for WM-RL, forest backtrack number 1.

| Metric | Value |
|---|---|
| Time | 77.60 s |
| Mean CTE | 0.066 m |
| Max CTE | 0.249 m |
| Std CTE | 0.049 m |
| RMS Steering | 0.381 |
| RMS Speed | 0.177 |
| Max $a$ | 0.195 $m/s^2$ |
| RMS $a$ | 0.036 $m/s^2$ |
| Max $\frac{da}{dt}$ | 0.058 $m/s^3$ |
| RMS $\frac{da}{dt}$ | 0.014 $m/s^3$ |

**Table B.8:** Performance metrics for WM-RL, forest backtrack number 2a.

| Metric | Value |
|---|---|
| Time | 57.12 s |
| Mean CTE | 0.076 m |
| Max CTE | 0.398 m |
| Std CTE | 0.078 m |
| RMS Steering | 0.403 |
| RMS Speed | 0.185 |
| Max $a$ | 0.202 $m/s^2$ |
| RMS $a$ | 0.042 $m/s^2$ |
| Max $\frac{da}{dt}$ | 0.088 $m/s^3$ |
| RMS $\frac{da}{dt}$ | 0.017 $m/s^3$ |

**Table B.9:** Performance metrics for WM-RL, forest backtrack number 2b, reverse of run 2a. Excluded from the average. Got also stuck a while on a small log, therefore the long time. See Figures B.30 and B.29.

| Metric | Value |
|---|---|
| Time | 120.22 s |
| Mean CTE | 0.053 m |
| Max CTE | 0.320 m |
| Std CTE | 0.047 m |
| RMS Steering | 0.335 |
| RMS Speed | 0.173 |
| Max $a$ | 0.160 m/s$^2$ |
| RMS $a$ | 0.028 m/s$^2$ |
| Max $\frac{da}{dt}$ | 0.063 m/s$^3$ |
| RMS $\frac{da}{dt}$ | 0.011 m/s$^3$ |

**Table B.10:** Performance metrics for WM-RL, forest backtrack number 3.

| Metric | Value |
|---|---|
| Time | 43.78 s |
| Mean CTE | 0.056 m |
| Max CTE | 0.428 m |
| Std CTE | 0.066 m |
| RMS Steering | 0.324 |
| RMS Speed | 0.188 |
| Max $a$ | 0.199 m/s$^2$ |
| RMS $a$ | 0.036 m/s$^2$ |
| Max $\frac{da}{dt}$ | 0.069 m/s$^3$ |
| RMS $\frac{da}{dt}$ | 0.014 m/s$^3$ |

**Table B.11:** Performance metrics for WM-RL, forest backtrack number 4a.

| Metric | Value |
|---|---|
| Time | 68.83 s |
| Mean CTE | 0.094 m |
| Max CTE | 0.608 m |
| Std CTE | 0.083 m |
| RMS Steering | 0.416 |
| RMS Speed | 0.183 |
| Max $a$ | 0.182 m/s$^2$ |
| RMS $a$ | 0.035 m/s$^2$ |
| Max $\frac{da}{dt}$ | 0.094 m/s$^3$ |
| RMS $\frac{da}{dt}$ | 0.014 m/s$^3$ |

**Table B.12:** Performance metrics for WM-RL, forest backtrack number 4b. This was driving the reverse direction on the path compared the 4a run. This table is not included in the average in Table 4.11.

| Metric | Value |
|---|---|
| Time | 62.09 s |
| Mean CTE | 0.113 m |
| Max CTE | 0.329 m |
| Std CTE | 0.094 m |
| RMS Steering | 0.422 |
| RMS Speed | 0.181 |
| Max $a$ | 0.150 $m/s^2$ |
| RMS $a$ | 0.038 $m/s^2$ |
| Max $\frac{da}{dt}$ | 0.058 $m/s^3$ |
| RMS $\frac{da}{dt}$ | 0.015 $m/s^3$ |

# C

# Research platform

## C.1 ROS 2 workspace

To test features without access to the camera, Jetson, or vehicle, a simple test node was implemented. It subscribes to a steering and throttle message and, using a kinematic bicycle model, publishes an odometry message remapped to the cuVSLAM topic name for compatibility with the rest of the workspace.

### C.1.1 Dealing with map jumps

Map jumps were handled by checking, every iteration, the 2D distance between the previous and current frame in which the full path was stored, both set to the map frame. If this distance exceeded a threshold of 30 cm, the full path was transformed into the latest map frame. The lower bound of the threshold is set by the vehicle's maximum speed, estimated at 4 m/s. The parameter could be lowered further, but a much lower value led to unstable behavior. Ideally the distance would be computed in 3D, since jumps occur in 3D, and orientation could be checked as well. This was done for the forest backtrack and the sloped reference paths in the sand. For all other path following, only a 2D distance check was used.

Figure C.1 shows one example where this handling was decisive for completing the path, here only 2D distance was used.

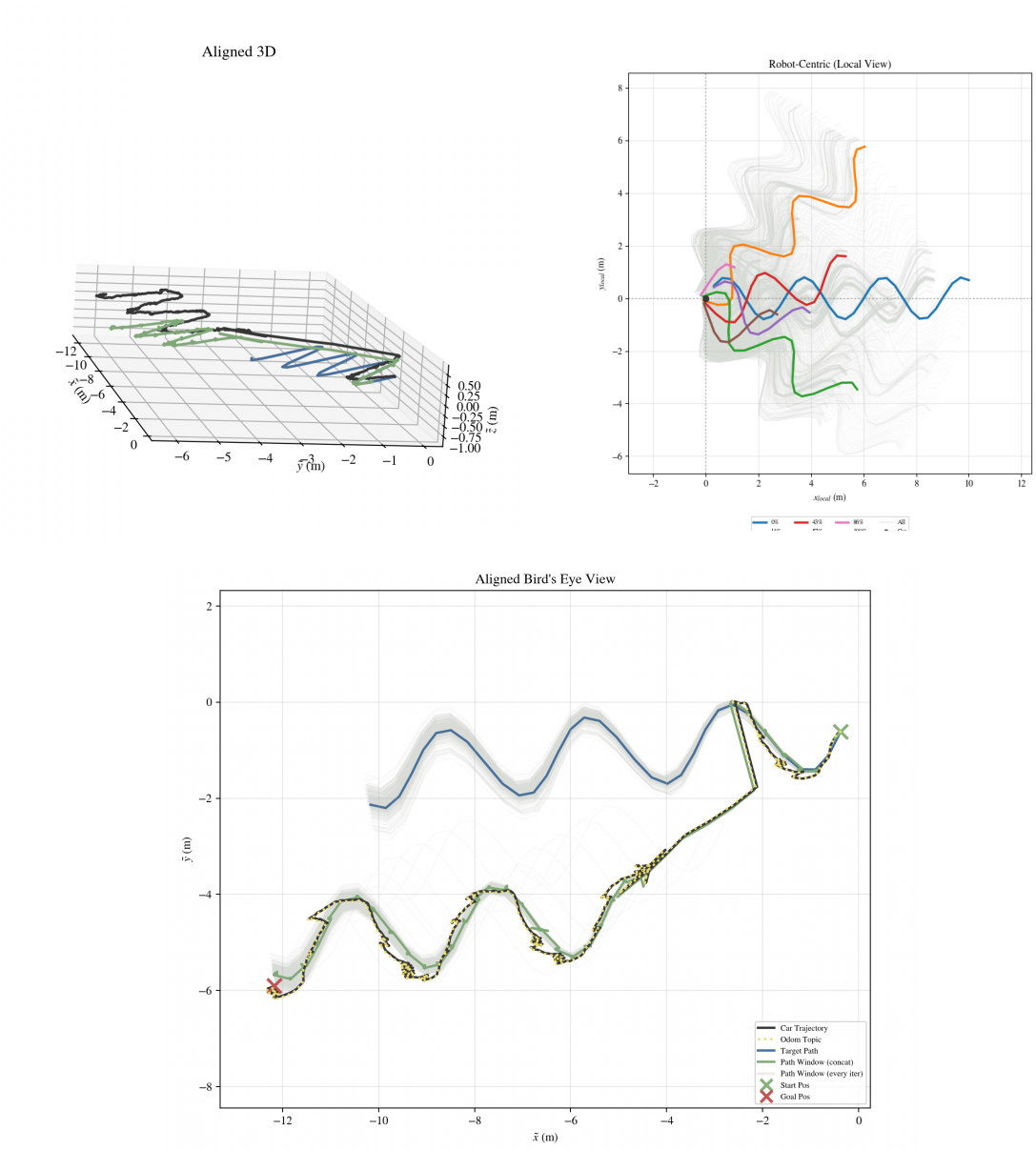



**Figure C.1:** WM-RL traversing path S0.8, but with a large jump in the map frame during the run. Bottom: 2D birds eye view, with the reference path in the original map frame in blue, the path window in the continuously updating map frame in green, the vehicle trajectory in black, and the odometry in dashed yellow. Top Left: The 3D view of the bottom plot. Top Right: All path windows sent to the controller in base frame, colored are the path windows at certain stages.

In Figure C.1 it was in fact the odometry that jumped, illustrating the localization problems discussed in Section 3.1.2.2. The jump handling let the controller finish the path by continuously supplying the correct path in the map frame, shown in the top-right frame. Note also the continuous vertical drift in the odometry in the top-left frame, which the controller server node did not handle, since only 2D distance was used here.

### C.1.2 Custom ROS 2 interfaces

**Listing C.1:** ROS 2 custom action definition for following a path by the controller server node.

```
#goal definition
nav_msgs/Path path

string controller_id
string goal_checker_id
bool reverse
---
#result definition
std_msgs/Empty result
---
#feedback definition
float32 distance_to_goal
float32 speed
```

**Listing C.2:** ROS 2 custom service definition for pushing a path to the path recorder node.

```
# Request (push a path)
nav_msgs/Path path

---

bool success
string message
```

**Listing C.3:** ROS 2 custom message definition used by the multiplexer node.

```
std_msgs/Header header
string mode

string MODE_KEYBOARD = "keyboard"
string MODE_XBOX = "xbox"
string MODE_AUTONOMOUS = "autonomous"
string MODE_STOPPING = "stopping"
```

# D

# Appendix Code

The code for the research platform and the WM-RL implementation can be found in: `https://github.com/ducksales/wm_rl_vehicle_control`.

**DEPARTMENT OF MECHANICS AND MARITIME SCIENCES**
**CHALMERS UNIVERSITY OF TECHNOLOGY**
Gothenburg, Sweden 2026
www.chalmers.se


CHALMERS
UNIVERSITY OF TECHNOLOGY