Model Predictive Control of Skid-Steered Mobile Robot with Deep Learning System Dynamics

by

Zhan Dorbetkhany

Submitted to the Department of Robotics and Mechatronics
in partial fulfillment of the requirements for the degree of

Master of Science in Robotics

at the

NAZARBAYEV UNIVERSITY

Apr 2023

Author .....
Department of Robotics and Mechatronics
Apr 29, 2023

Certified by .....
Almas Shintemirov
Associate Professor
Thesis Supervisor

Accepted by .....
Vassilios D. Tourassis
Dean, School of Engineering and Digital Sciences

Model Predictive Control of Skid-Steered Mobile Robot with Deep Learning System Dynamics

by

Zhan Dorbetkhany

Submitted to the Department of Robotics and Mechatronics
on Apr 29, 2023, in partial fulfillment of the
requirements for the degree of
Master of Science in Robotics

Abstract

This thesis project presents several model predictive control (MPC) strategies for control of skid-steered mobile robots (SSMRs) using two different combinations of software environment, optimization tool and machine learning framework. The control strategies are tested in WeBots simulator. Spatial-based path following MPC of SSMR with static obstacle avoidance is developed in MATLAB environment with ACADO optimization toolkit using spatial kinematic model of SSMR. It includes static obstacle and border avoidance strategy based on artificial potential fields. Simulations show that the controller is effective at driving SSMR on a track, while avoiding borders and obstacles. Several more MPCs are developed using Python environment, ACADOS optimisation framework, and Pytorch-Casadi integration framework. Two time-domain controllers are made in Python environment, one based on SSMR kinematic model and another based on data-driven state-space model using Pytorch-Casadi framework. Both are setup to reach a goal point in simulation experiment. Experiments show that both versions reliably reach a target point. Standard and data-driven versions of spatial path following MPC are developed. Standard is a re-implementation of MPC designed in MATLAB with modifications to cost function and border avoidance, without static obstacle avoidance. Data-driven path following MPC is an extension of standard variant with state-space model replaced with a hybrid of spatial kinematics and data-driven model. Simulation of both spatial controllers confirm their effectiveness in following reference path.

Thesis Supervisor: Almas Shintemirov
Title: Associate Professor

Acknowledgments

I would like to express my gratitude and appreciation to my supervisor, professor Almas Shintemirov, who has provided invaluable guidance and support throughout this thesis and prior research.

I would like to thank my co-supervisor, professor Matteo Rubagotti, for great help in improving academic writing and assistance in research work.

To the professors, teaching assistants and lab assistants of NU School of Engineering and Digital Sciences, thank you for helping to learn skills that were necessary to complete this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Autonomous mobile robots need to navigate complex and dynamic environments with accuracy and safety for various applications that involve interacting with humans, different terrains and other autonomous agents; some examples of these applications are different kinds of human helpers or delivery service robots. To navigate any environment effectively, robots have to follow the desired path precisely and dodge obstacles. Moreover, complex dynamic environments demand that robots cope with moving obstacles and changing terrain: robots need to avoid moving obstacles safely and maintain navigation accuracy on any terrain.

Model predictive control (MPC) computes optimal control input sequence within system constraints by predicting and optimizing controlled system's future behavior over a finite time horizon [30]. Ability to account for future behavior and system constraints allowed MPC to see wide adoption in field of trajectory tracking control and motion planning of autonomous vehicles and mobile robots [17, 23, 29].

Skid-steering driving scheme is defined by lack of explicit steering hardware. This considerably reduces size, weight and complexity of skid-steered mobile robots, improving robots' mobility and overall performance. Typical Ackermann steered robots turn front wheels to perform rotation, while skid steer mobile robots (SSMR) rotate due to different right and left side wheel velocities. Due to such advantages, skid-steering has been established as most popular driving scheme for mobile robots.

Path following control's defining feature is track-dependency of the system that

is suitable for case of absent reference velocity profiles. There are numerous path following approaches such as the model predictive contouring control [22] that was employed to control car-like vehicles [24, 32] and robots [5]. Another method, shown in [11, 10, 6], is based on spatial system models that allow for simpler obstacle and road bounds definitions.

Accurate kinematic modelling of SSMR is essential for performance tracking MPC. One of more widely used models is classical unicycle and differential drive models; they were used with different control methods as in [20] as well as with MPC [5, 26]. More accurate models are ones based on *instantaneous centre of rotation* (ICR) of an SSMR [4]. The first such models were presented in [7, 21], which were enhanced to consider wheel slippage via addition of terrain-dependent parameters in [25].

Having a fixed model is often not sufficient to ensure low model error in different conditions. Kinematic model learning is used to solve problem of mathematical model and system mismatch by actively optimizing the model for minimal error. This can be achieved by learning optimal parameters for ICR kinematic model with optimization-based method such as moving horizon estimation (MHE) as it was demonstrated in [26]. Another approach is to use machine learning based methods like Gaussian Mixture Models (GMM) [27] or neural networks [18] to perform inference of optimal parameters based on set of feedback metrics. Such techniques do improve model performance, but they are still constrained by representative capacity of conventional manually derived kinematic models and, consequently, have limited ability to model complex dynamics.

Recent efforts in learning-based MPC deploy system dynamics represented entirely with machine learning models. However, there are limitations caused by high computation load of such models relative to capabilities of embedded systems. In effort to use machine learning models of system dynamics for mobile robots, researchers balance model complexity and computational load. Authors of [36] and [13] employ Gaussian Process Model with paired back number of sampling points, while [8] and [35] use small neural networks. Meanwhile, in [31], authors use a *Gauss-Newton* algorithm to create a local approximation of large neural network that stays close to

network's performance. This approach allows to learn much larger models capable of representing complex dynamics caused by environmental conditions like complex surface properties, variable weight load, aerodynamic effects etc.

Inspired by [31] and [16], this work proposes a data-driven MPC of SSMR with a kinematic model represented by a deep neural network.

Chapter 2

Related Work

2.1 Kinematic modelling of SSMR

A kinematic model of a vehicle is a simplified mathematical representation of the motion of a vehicle that considers only the position and orientation of the vehicle, and the velocity of the vehicle. It is commonly used in robotics, control theory, and autonomous vehicle navigation. The kinematic model can be used to calculate the trajectory of the vehicle based on its current state, and to plan a path for the vehicle to follow. It is also used to design control systems for the vehicle. The model typically includes variables such as the position and orientation of the vehicle, its velocity and acceleration, and the steering angle of the front wheels. Although the kinematic model is a simplified representation of the motion of a vehicle, it can provide useful information for many applications, such as obstacle avoidance, lane tracking, and trajectory planning. More complex models, such as the dynamic model, take into account factors such as the mass and inertia of the vehicle, tire characteristics, and external forces, but they require more computational resources and are generally more difficult to implement [9]. In research area of skid-steered mobile robots, many different kinematic models were developed such as classical unicycle and differential drives as well as more complex models based on instantaneous center of rotation (ICR).

Paper by L. Caracciolo, A. de Luca and S. Iannitti [7] is one of the earlier de-

velopments in linear control based on instantaneous center of rotation kinematics for SSMRs with skidding taken into account. In the paper, a robust trajectory tracking control system for 4 wheel drive vehicles is presented. The authors begin by deriving a dynamic model of a 4 wheel drive vehicle and designing a model-based tracking controller using an approach utilized for motion planning and control of nonholonomic wheeled mobile robots. The method involves specifying the longitudinal coordinate of the ICR (instantaneous center of rotation) to ensure it remains inside the robot's wheelbase, which is an operational kinematic constraint. The authors then fully linearize the system using dynamic state feedback. The resulting closed-loop system is linear and input-output decoupled, making it easy to stabilize the vehicle to a desired trajectory using linear control techniques. However, the overall nonlinear control law depends on soil parameters. The authors study the effects of uncertain soil parameters on the closed-loop system's dynamics with respect to a specific class of trajectories and propose a robust control scheme that can reject these disturbances. Finally, simulation results are presented to demonstrate the tracking of robot trajectories over a virtual terrain with varying parameters.

Work by A. Mandow et al. [25] is an example of ICR-based model of 4-wheel skid-steered vehicle. The aim of the study was to enhance the real-time motion control and dead-reckoning of wheeled skid-steered vehicles by accounting for slippage effects without adding the complexity of dynamics computations to the loop. Previous research reported a method for obtaining an optimized kinematic model for skid-steer tracked vehicles by experimentally determining the boundedness of the instantaneous centers of rotation (ICRs) of treads on the motion plane. The paper further explores this method, now extended to wheeled skid-steer vehicles, and successfully applies it to a popular research robotic platform, the Pioneer P3-AT, with different types of tires and terrains. The result is a kinematic model for skid-steer vehicles that can incorporate the impact of vehicle dynamics by considering only two points in local coordinates, which are the ICR_l and ICR_r. These variables are constrained for skid-steer vehicles and correspond to the wheel contact points for an ideal differential drive system. Furthermore, authors present a method for experimental

identification of kinematic model parameters using genetic algorithm and simple expressions relating actual rotation as well as displacement to wheel velocities. Thus, the proposed approach derives an approximate kinematic model through experimental means, which takes into account the limited range of Instantaneous Centers of Rotation for both treads on the motion plane. The model establishes a relationship between the drive and vehicle velocities while also accounting for slippage. Additionally, the model includes adjustment parameters to account for any misalignment or undisclosed mechanical inaccuracies.

2.2 Model Predictive Control

Model-based predictive control (MPC) refers to a collection of advanced control techniques that employ a process model to anticipate the future behavior of the system under control. MPC predicts system states over a set time horizon and computes optimal control signals for the said horizon; only solution for the first time step is applied to the system before MPC recomputes the solution with updated initial state. So, MPC controls the system based on predictions of mathematical model, this moves the emphasis of designing a controller towards modeling the system that needs to be controlled. Such models are commonly available in various engineering fields, simplifying and broadening applications of MPC in control of various system. Its indirect formulation upholds the physical comprehension of the system parameters, which simplifies the adjustment of the controller. Being based on numerical optimisation and system models, MPC can manage systems that standard feedback controllers like PIDs cannot [33, 30]. Autonomous robotics is one of the fields where MPC has been widely utilized in recent years owing to computationally faster embedded systems that can manage solving MPC's optimal control problem in real-time.

Work by E. Kayacan, H. Ramon and W. Saeys [17] presents a MPC of an autonomous tractor-trailer system based on kinematic tricycle model. The kinematic model was simplified by neglecting dynamic force balances. The controller developed in this work uses a novel trajectory tracking error-based model. The contrast between

the conventional model based on trajectory tracking error and the suggested approach lies in the inputs used. The former utilizes speed and yaw rates as inputs, while the latter employs speed and yaw models to devise a controller. Consequently, in the new trajectory tracking error-based model, the inputs are the gas pedal position and steering angles. The control scheme combines MPC solutions with feed-forward and robust control actions. To compute the actual control inputs to be implemented in the real-time system from discrepancies between the reference and real control variables found by MPC, the outputs of the MPC had to be merged with a feed-forward control action. Because the trajectory tracking error-based model is derived through linearization of the system around the reference trajectory, any deviation between the model and the actual system can lead to suboptimal control performance when the system is far from the reference trajectory. Hence, an additional robust control action was introduced to move and sustain the system near the reference trajectory. Testing in the field has demonstrated that the designed controller can control the system with a satisfactory degree of precision, despite the occurrence of modeling errors and external disturbances.

Control system developed by J. Li et. al.[23] is an MPC-based approach made to combine trajectory planning and tracking control strategies for Autonomous Guided Vehicles(AGVs), allowing them to follow a reference path accurately and smoothly. The original path generated by the path planner is enhanced through the MPC-based approach. Then, a trajectory is devised based on the improved path utilizing constrained reference velocity. The AGV can subsequently follow this trajectory to achieve superior control performance. The proposed approach, which employs MPC to solve both the trajectory planning and tracking control issues, is called the MPC-based unified planning and tracking approach. Additionally, the framework accounts for obstacle avoidance. The AGV can avoid any static or dynamic obstacle obstructing its path through an obstacle detection and path replanning mechanism. Thus, the proposed technique allows to consider vehicle dynamics in both planning and tracking, avoiding unrealistic planning. The results from simulations and real manufacturing environment indicated that the MPC-based unified trajectory planning and tracking

control strategy has benefits in terms of enhancing the tracking accuracy and ensuring smooth movement.

R. Quirynen et al.[29] have developed an MPC-based control with obstacle avoidance capabilities. The paper presents an innovative approach for obstacle avoidance in self-driving systems that utilizes a hierarchical software architecture. This architecture involves a low-rate, long-term motion planning algorithm, as well as a high-rate, highly reactive model predictive controller. The proposed integrated framework features a particle-filter based motion planner, which computes a reference trajectory to be tracked, combined with a nonlinear model predictive control (NMPC) trajectory-tracking algorithm. The covariance from the motion planner is utilized to adjust the time-varying tracking cost in the NMPC problem formulation automatically. The covariance linked to the trajectory denotes the level of confidence that the path planner has in the efficacy of its computed trajectory. The objective is to permit greater deviations when the planner has a lower level of confidence, and fewer deviations when it has a relatively higher level of confidence. Preliminary experimental results conducted on a small-scale autonomous vehicle test platform demonstrate that the proposed approach can facilitate secure obstacle avoidance and dependable driving behavior in moderately complex scenarios.

2.3 Path following Control

Path following control is an approach that defines autonomous vehicle's movement with respect to desired reference trajectory. This method was extensively used with MPC to control a variety of vehicle types. There are variations of path following used with MPC, contouring control and spatial-based control. Contouring control parameterizes vehicles movement along reference trajectory, but keeps vehicles model and corresponding feedback data in fixed global frame. This allows precise online definition of reference points along a track without the burden of representation of feedback data and any other parameters in moving reference frame. Spatial-based control moves the entire control problem into reference frame of virtual vehicle that

moves along reference trajectory. Such approach simplifies optimization reference vector and definition of trajectory-adjacent obstacles like borders or other moving vehicles, but necessitates all positional information to be converted to new moving reference frame at each time sample.

B. Brito et. al.[5] present an Model Predictive Contouring Control (MPCC) approach with collision avoidance system. The method is based on a reformulation of the Model Predictive Contouring Control (MPCC) approach, specifically designed for real-time navigation of AGVs in complex environments with multiple agents. The conventional approach to address motion planning and control for AGVs involves solving two separate problems. First, the motion planner generates a path that avoids collisions, and then the motion controller directly controls the AGV's actuators to follow that path. Proposed method integrates motion planning and control into a single module, using constrained optimization techniques. This allows to generate local trajectories that are kinematically feasible and have fast replanning cycles. The design includes localization, environment perception, and detection of both static obstacles and pedestrians. It is lightweight and runs in real-time, using an open-source solver. Such design is able to perform local motion planning in unstructured environments with both static and moving obstacles, including humans. This is achieved using an optimization-based receding-horizon approach to compute a local trajectory that minimizes tracking error while avoiding obstacles, given a reference path and speed. The approach builds on MPCC and extends it to incorporate a static map by computing a set of convex regions in free space online. Moving obstacles are modeled as ellipsoids, and a correct bound is provided to approximate the collision region, given by the Minkowsky sum of an ellipse and a circle. A MPC is implemented to compute an optimal control command for the AGV, which incorporates predictions of dynamic obstacles, enabling it to smoothly avoid moving obstacles. As a contouring control implementation, the solution relies on global reference computation using limited reference track segments to save computational cost. The method is agnostic to the robot model and has been experimentally validated with a mobile robot navigating in indoor environments populated with humans. The approach is executed

fully onboard without the need for external support and can be applied to other robot morphologies, such as autonomous cars.

Paper by G. Huskić et. al.[15] proposes control strategy based on Lyapunov theory and kinematic model defined in path coordinates. The article presents a new control law for path following with skid-steered mobile robots that is able to account for the nonholonomic constraint that is inherent to skid-steered vehicles. The approach utilizes a terrain-dependent kinematic model in path coordinates with experimentally evaluated parameters. Based on spatial model, kinematic path following control was developed using the Lyapunov approach. Additionally, the authors propose a separate linear velocity control that takes reachable curvatures and actuator saturation into account. The work also provides formal proof of the robot's convergence to the path. The research also presents experimental evaluations of the approach in different terrain scenarios and compares it with two other state-of-the-art algorithms; tests have shown that the proposed approach is able to outperform them on three different terrain types. The experiments were conducted using the Robotnik Summit XL, a well-known commercial mobile robot.

2.4 Learning-based Model Predictive Control

The success of MPC heavily relies on having a precise dynamics model. A flawed model may result in meaningless trajectories and inadequate control performance. Selecting an appropriate model for MPC involves considering the model's complexity and parameters to achieve satisfactory performance. As models become more complex, the optimization process becomes more challenging, and the designer must balance complexity and computational time. Moreover, the dynamics of a vehicle can change due to various factors such as frictional, aerodynamic, or damping forces, which make a single fixed model parameterization insufficient [33]. Recent research is trying to solve the problem of vehicle dynamics modeling by employing machine learning models that learn vehicle dynamics from their environment.

G. Torrente et al. [36] developed an MPC of Quadrotor Unmanned Aerial Ve-

hicle (UAV) that uses Gaussian Processes to model aerodynamic effects. The work utilizes Gaussian Processes to learn the residual dynamics of a simplified quadrotor model that neglects aerodynamic effects. Model proposed in the paper is a nominal kinematic model of Aerial Vehicle with 6 degrees of freedom augmented with residual dynamics represented by Gaussian Processes. To train the Gaussian Processes, real-world flight data was collected using the nominal dynamics model. Collected data includes: for each sample at time t_k, the velocity at the subsequent sample point B^{v_k+1}, the predicted velocity at the next sample point $B^{\hat{v}_k+1}$, and the time step δt_k. Using this data, it was possible to compute the time-normalized velocity error as well as corresponding acceleration error. Only part of data was used with Gaussian Processes at a time. Hyperparameters of the GP kernel function were derived by applying maximum likelihood optimization on the acquired dataset. Since GP regression is a non-parametric technique, the complexity increases as the number of training points increases. As employing the full dataset would render real-time MPC optimization infeasible, a small number of inducing points were selected by subsampling the dataset. Considering the aerodynamic effects' smooth nature, subsampling was done by sampling the points at even intervals within the ranges of the training set. By learning the residual dynamics, the proposed solution simplifies the learning task and describes the model augmentation using low number of inducing points for Gaussian Processes. This compact model was effectively used with a MPC framework that was able to benefit from the combined dynamics formulation. The approach was validated by conducting a comprehensive comparison with a cutting-edge linear drag model through both synthetic and real-world experiments, covering speeds of up to 14 meters per second and accelerations exceeding 4g. It was demonstrated that the model can be optimized effectively within an MPC pipeline, facilitating control frequencies greater than 100Hz, and that the MPC augmented with this model enhances trajectory tracking performance by up to 70% compared to its nominal counterpart.

MPC of autonomous underwater vehicle and autonomous race cars proposed by L. Hewing, J. Kabzan and M. N. Zeilinger [13] also uses Gaussian Processes to learn residual vehicle dynamics. Proposed model predictive control strategy combines a

nominal model with a Gaussian Process model of an additive nonlinear component of the dynamics. The paper presents a method for defining the chance-constrained MPC problem, which accounts for the residual uncertainties supplied by the GP model to facilitate cautious control. This work emphasises computational efficiency of the learned model. Authors explore the approximate propagation of system uncertainty and the principled formulation of chance constraints in terms of probabilistic reachable sets resulting in a deterministic approximation of the stochastic optimal control problem, which is well-suited for numerical optimization. The nominal system description enables the reduction of GP model learning to a subspace of states and inputs, thereby lowering the dimensionality of the machine learning task. Introduction of dynamic sparse GPs based on inducing points as an approximation technique designed for MPC further reduces the computational burden of the approach. Together, these developments lead to an MPC formulation capable of controlling high-performance systems at sampling rates of a few milliseconds. Experimental evaluation is done in two application examples. The first involves a simulation of an autonomous underwater vehicle, which demonstrates key concepts and advantages in a simplified setting. The second example comprises a hardware implementation for autonomous racing of remote-controlled cars, demonstrating the real-world feasibility of the approach for complex high-performance control tasks. The results have shown that the approach leads to improved performance and safety. This indicates that cautious data-driven techniques have potential for improving high-performance control systems, and demonstrates the practical feasibility of learning approaches in these systems.

Work by K. Y. Chee, T. Z. Jiahao and M. A. Hsieh [8] presents an MPC of a quadrotor employing kinematic model augmented with knowledge-based neural ordinary differential equations (KNODE). A new category of neural networks called "neural ordinary differential equations" (NODE) has been demonstrated as a useful technique for deriving dynamic models from data. NODE is capable of approximating a neural network with continuous depth, which can directly represent differential equations. A variation of NODE called "knowledge-based NODE" (KNODE) has been

proposed to exploit NODE's compatibility with fundamental principles. KNODE combines first-principle models and neural networks to create hybrid models that enhance the accuracy of open-loop predictions of nonlinear dynamic models. This paper applies KNODE to construct a model of the quadrotor's dynamics. Specifically, the proposed solution employs the neural network in the hybrid model to accommodate residual and uncertain dynamics present in the system. Furthermore, this hybrid model was integrated into a model predictive control framework called KNODE-MPC. To train the KNODE system, authors utilize trajectory data obtained from closed-loop simulations or experiments, assuming that system dynamics and nominal model are fully separable. KNODE architecture is a compact neural network with two 64 and 16 unit wide hidden layers. The training approach is such that the training data requires the inclusion of control inputs in order to make one-step-ahead predictions using the model. So, the trajectory data is partitioned at each time step and the state at each of these time steps is used as initial conditions for one-step-ahead predictions. To train the hybrid model on multiple trajectories, the losses calculated were aggregated on each trajectory. The adjoint sensitivity method is used to propagate gradients from the loss function to the neural network parameters, while the parameters are iteratively updated based on the gradients using common optimizers such as Adam. Authors note that the adjoint sensitivity method has been recognized as a memory-efficient substitute for the traditional backpropagation. Simulation and physical experiments comparing KNODE, Gaussian Process (GP) and purely nominal model were also presented to demonstrate effectiveness of the approach. In simulation, when evaluating the median error across all trajectories at various radii and speeds, both KNODE and GP exhibit a superior performance compared to the nominal model, with an improvement of over 80%. Analysis of circular and lemniscate trajectories has shown that the KNODE model has a higher accuracy than the GP model, with an overall median prediction error improvement of 19.1%, and KNODE also demonstrates more consistent accuracy across all trajectories. In physical experiment, KNODE has also managed to outperform nominal model by 34.1% and GP by 12.4%.

Another example of vehicle dynamics learning with a small neural network is a paper by N. A. Spielberg, M. Brown and J. C. Gerdes [35], which presents a neural network based MPC (NNMPC) for autonomous driving in case of unknown friction dynamics. The proposed solution utilizes a combination of neural network models with input history and a second-order interior point solver that can effectively manage complex costs and constraints on the state or input. To optimize the balance between computation time and network complexity, network architecture is configured to a two-layer feedforward neural network with previous states and controls as inputs. A straightforward, streamlined network structure and introduction of a second-order optimization method has allowed to achieve real-time performance on a full-scale autonomous vehicle. Unlike [36, 13, 8], proposed NNMPC uses neural network to fully learn dynamics of a vehicle instead of simplifying the problem by learning residual dynamics of nominal kinematic model. The neural network model was trained on combined vehicle data from both icy and dry road conditions. This model is then integrated into a real-time NNMPC framework, which is responsible for tracking a trajectory near the limits on both high- and low-friction courses of an automated race car, without prior knowledge of the road-tire friction coefficient. This makes the NNMPC to use knowledge of past states and controls to choose the correct steering command, without explicit information on whether the vehicle is on high-friction asphalt or low-friction ice. The experiments have shown that the NNMPC approach not only handles this challenge but also demonstrates improved tracking performance compared to a physics-based dynamics model in the same MPC controller.

Previous examples dealt with computational complexity of machine learning models by reducing their complexity, a paper by T. Salzmann et. al.[31] proposes a framework that allows integration of significantly more complex deep neural networks with real-time MPC. This paper introduces Real-time Neural MPC (RTN-MPC), an efficient framework for incorporating large, data-driven dynamics models into MPC and deploying it on embedded devices in real-time. RTN-MPC can integrate arbitrary neural network architectures as dynamics constraints into the MPC formulation using CPU or GPU parallelization. The framework allows for unconstrained model

architecture selection, enables real-time capability for larger models, and provides GPU acceleration without a loss in performance, unlike a simple integration of a deep network into an MPC framework. Main contribution is a computational approach for RTN-MPC, which involves separating the computationally intensive data-driven model from the MPC optimization, thereby allowing for efficient online approximations that can handle larger and more complex models while still retaining real-time performance. In order to accelerate the generation of QP (Quadratic Programming) solutions, the authors employ a strategy that replaces the computationally demanding globally valid data-driven dynamics equation described by neural network with a computationally lightweight locally valid approximation, which is accurate up to second order around the current iterate. Approach in MPC design is similar to [36, 13, 8], system model used in RTN-MPC is a combination of nominal kinematic model with neural network based approximation of residual dynamics. The solution is implemented using the optimization framework built around CasADi[3] and ACADOS[37], and deep learning framework PyTorch[28]. This allows to utilize any neural network model that can be trained in PyTorch and implemented in CasADi within ACADOS solver. The authors conducted experiments to evaluate the real-time inducing approximations of their proposed controller and found that performance was not negatively impacted. This overcomes a previous limitation where sacrificing performance was necessary for efficiency. The usefulness of the proposed method was demonstrated by evaluating its real-time capability on various devices and applying it to the challenging task of trajectory tracking for a highly agile quadrotor, resulting in substantial reduction of tracking error while using powerful models on the device.

Chapter 3

Methodology

The first stage of the work was about implementation of path following control with nominal kinematics and static obstacle avoidance. It was completed using ACADO toolkit [14] as optimization solver, MATLAB as software environment and WeBots [2] as simulation environment.

Webots is a software that is both free and open source, designed for the development, modeling, programming, and simulation of robots. It is utilized extensively in research, education, and industry. As part of this project, Webots serves as the primary means for evaluating the effectiveness of the MPC approach as a robot motion planner. In experiments, pioneer 3-AT skid-steered robot was used as testing platform. The Webots simulator's robot model utilizes a low-level control strategy to follow the reference commands, which are the angular velocities and accelerations of the wheels that are provided by the MPC controller. The control strategy is based on the feedback information on the wheel speeds. Depending on type of tested controller, simulation environment was varied between empty floor with a goal flag on figure 3-1, a track without obstacles on figure 3-2 and track with obstacles on figure 3-3.

The ACADO toolkit is short for "Automatic Control and Dynamic Optimization," and as its name implies, it is a software environment that includes algorithms for automatic control and dynamic optimization. It offers various control algorithms, including model-predictive control, and allows for the generation of MPC controller's self-contained C code. This code can be integrated into Webots controller code, which

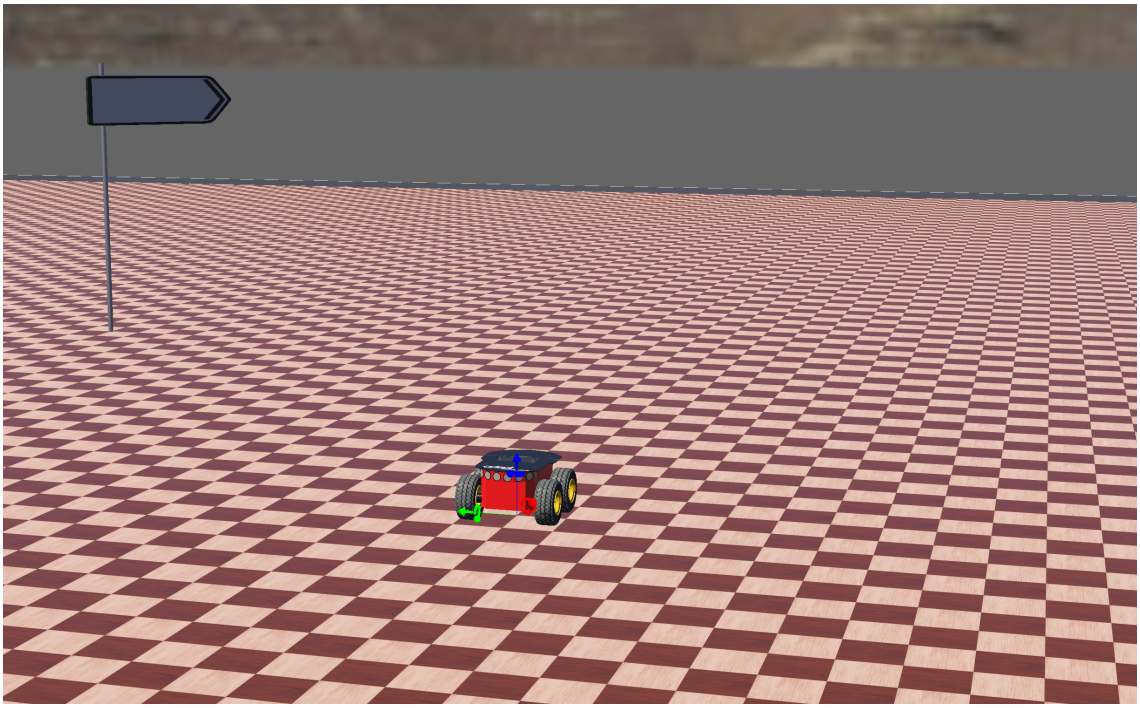


Figure 3-1: WeBots scene with empty floor and a Goal flag.

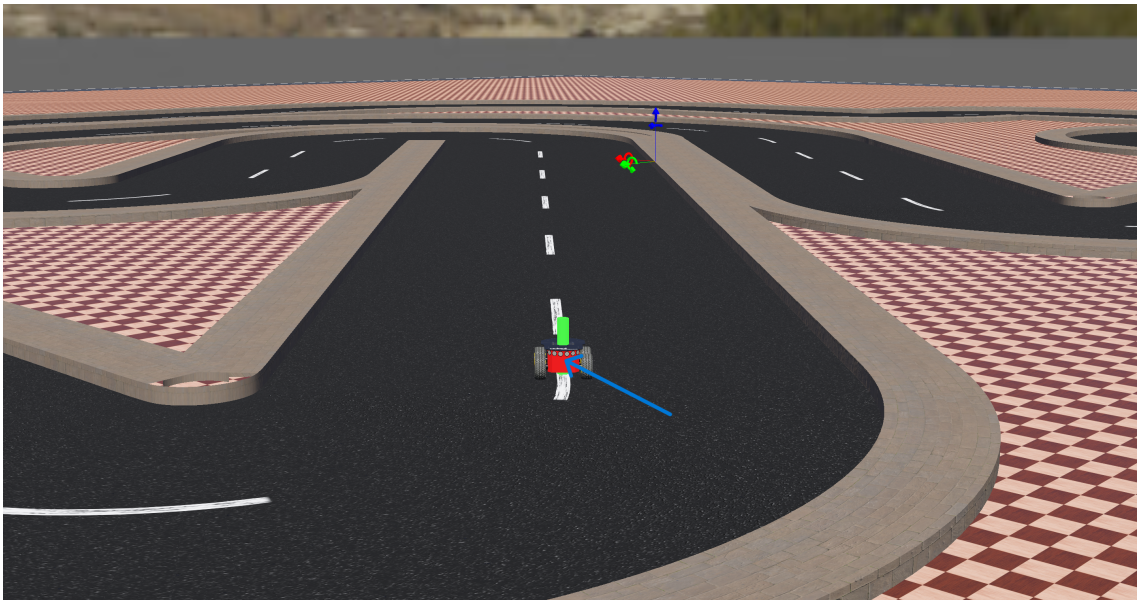


Figure 3-2: WeBots scene with a track (robot marked with blue arrow).

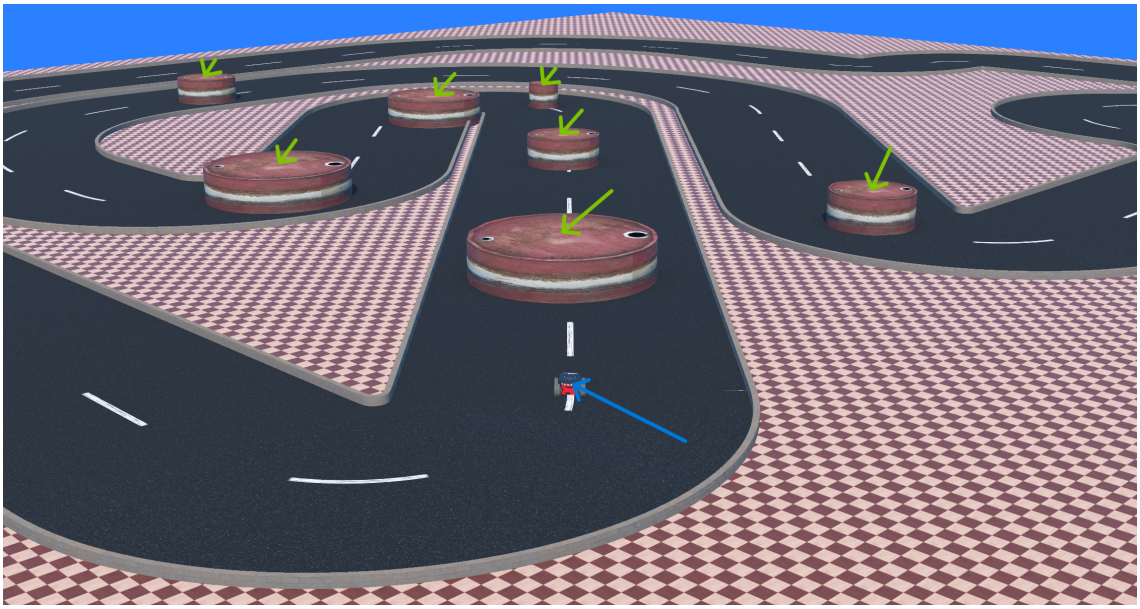


Figure 3-3: WeBots scene with track and obstacles (blue arrow points to the robot, green ones point to obstacles).

will be used to program robot controllers in the Webots simulation environment. ACADO serves as a solver to minimize a cost function constructed with the robot's position, direction, and speed parameters, which are defined by the differential system derived from the robot's kinematic model.

After thorough testing of pass following MPC implementation in MATLAB-ACADO framework, goal of the second stage of the project was set to develop a framework for data-driven path following MPC design using Python software environment, updated optimization software ACADOS [37], latest version of WeBots [2], and including integration of data-driven MPC design framework presented in [31]. Development of the controller using open libraries and Python environment was pursued for several key reasons:

1. It makes possible integration of popular machine learning and deep learning framework such as PyTorch either for parameter inference or full dynamics learning.

2. Allows for integration of code with larger variety of simulators such as NVIDIA Isaac Sim [1].

3. Simplifies integration with ROS framework as well as any prototyping platform that is able to run Python code, which will simplifies testing on real robotic platforms.

ACADOS is a software package designed to efficiently solve optimal control and estimation problems, and it serves as the successor to the ACADO software package developed by the team of Prof. Moritz Diehl at KU Leuven and the University of Freiburg. It provides a range of computationally efficient building blocks tailored for optimal control and estimation problems, including modules for integrating ordinary differential equations (ODE) and differential-algebraic equations (DAE), interfaces to advanced QP solvers such as HPIPM, qpOASES, qpDUNES, and OSQP, condensing routines, and nonlinear programming solvers that use the real-time iteration framework. The back-end of acados relies on the high-performance linear algebra package

BLASFEO to increase computational efficiency for small to medium-scale matrices, which are typical in embedded optimization applications. MATLAB, Octave, and Python interfaces are available to conveniently describe optimal control problems and generate self-contained C code that can be easily deployed on embedded platforms. ACADOS has several key advantages over ACADO: ability to work without generation of C code, updated solver algorithms, and integration with symbolic framework CasADi [3]. Compatibility with CasADi was crucial in work presented in [31].

CasADi is a tool that is open-source and designed for nonlinear optimization and algorithmic differentiation. CasADi's core is a symbolic framework that implements both forward and reverse mode of algorithmic differentiation on expression graphs. This allows for the construction of gradients, large-and-sparse Jacobians, and Hessians. The expression graphs are encapsulated in Function objects and can be evaluated in a virtual machine or exported as stand-alone C code.

Deep learning is a type of machine learning that involves training artificial neural networks to perform complex tasks, such as image and speech recognition, natural language processing, and decision-making. Deep learning is based on the architecture of the human brain and is designed to simulate the way that humans learn by processing information through multiple layers of interconnected nodes. Deep learning models typically consist of multiple layers of nodes, with each layer processing and transforming the input data to produce a more refined output. The layers are trained using a large dataset of labeled examples, allowing the model to learn to recognize patterns and features in the data. Some popular deep learning architectures include convolutional neural networks (CNNs) for image processing, recurrent neural networks (RNNs) for sequence data, and transformer models for natural language processing. Deep learning has been widely applied in various fields, including computer vision, speech recognition, natural language processing, and robotics[12].

In this work, one of the goals was to use deep learning models as learning-based system model for MPC. Framework developed in [31] allows trained deep learning PyTorch models to be utilized in CasADi graphs and, subsequently, in Acados optimal control problems. There are two ways in which this framework enables PyTorch

models in a CasADi graph:

1. Naively, where the operations of the PyTorch model are reconstructed in the CasADi graph, and the learned weights are copied over. However, this approach is limited to dense multi-layer perceptrons (MLPs) and can be slow for large networks as CasADi is not optimized for large matrix multiplications..

2. Approximated, where the PyTorch model is abstracted as a first or second-order approximation. The necessary parameters are passed to the CasADi function at every function call. This enables the use of any differentiable PyTorch module. The described approach can be used to efficiently apply a learned dynamics model in an MPC setting.

In this project, the approximation of PyTorch models are used as MPC system model; this saves large amount of computational cost and allows to use arbitrary PyTorch models as CasADi graphs instead of only dense MLPs.

Chapter 4

Implementation and Results

4.1 Path following MPC in MATLAB-ACADO

This implementation of MPC is based on the findings of [15, 16] and employs the benefits of MPC to introduce a new model predictive path following control (MPPFC) approach for SSMRs that operate in an environment with obstacles. The main results include: 1) a new MPC-based path following controller, which is derived from the spatial SSMR kinematic model of [16]; and 2) a static obstacle avoidance strategy that is incorporated into the controller to ensure that the robot follows the desired path with minimal deviations. Extensive simulation experiments were conducted using a physical robot simulator to demonstrate the effectiveness of the proposed MPPFC framework.

4.1.1 Spatial SSMR kinematics

The kinematic model of the skid-steer robot is established based on the diagram illustrated in Figure 4-1. The robot is modeled as a solid rectangular body propelled by four wheels that remain fixed in position. The robot is presumed to pursue a designated path, denoted by P, while concurrently varying the virtual instantaneous centers of rotation for the robot's center of mass (ICR), left wheel (ICR_l), and right wheel (ICR_r), in accordance with previous works [25]. In [21], the motion of the

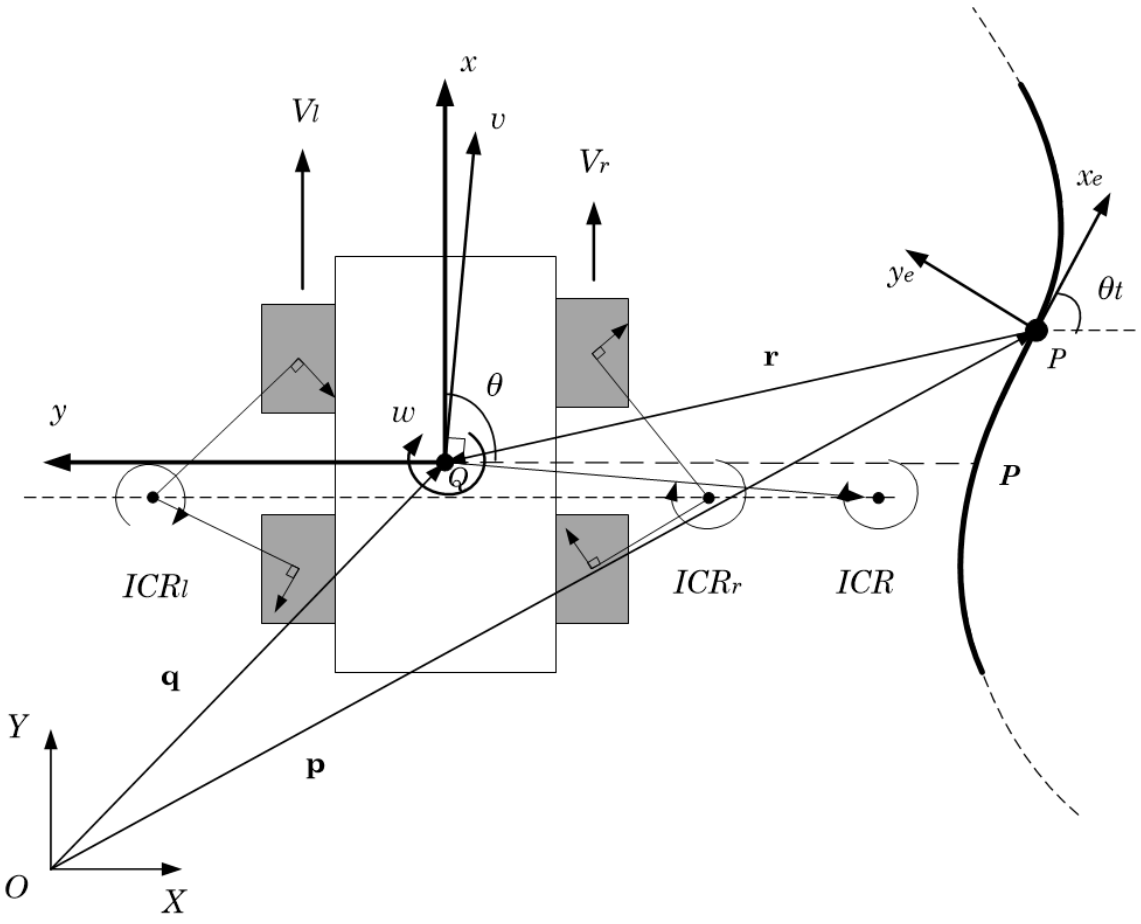


Figure 4-1: 4-wheeled skid-steer mobile robot spatial kinematics based on Fig. 1 in [15].

SSMR in global coordinates is presented and expressed as:

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & x_{ICR}\sin\theta \\ \sin\theta & -x_{ICR}\cos\theta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ \omega \end{bmatrix}, \tag{4.1}$$

The symbols X, Y, θ, v_x, and ω represent two position and one orientation coordinates, as well as the longitudinal and angular velocities of the robot's center of mass, respectively.

The fundamental principle of path following control is to guide the robot along a specified path, while maintaining a pre-determined forward speed profile. In path following control, the target reference frame is a moving frame that is represented by P, as illustrated in Figure 4-1. To convert a point (x, y) from the O frame to the P frame, the following expressions are applied to coordinates in O frame:

$$x_e = x - x_f, \quad y_e = y - y_f, \quad \theta_e = \theta - \theta_t, \tag{4.2}$$

The coordinates x_f and y_f indicate the position of P relative to O in the conversion of a point (x, y) from the O frame to the P frame, as stated previously.

In [34], the state-space model for path-following was formulated as follows:

$$\begin{aligned} \dot{x_e} &= \begin{bmatrix} \cos\theta_v & \sin\theta_v \end{bmatrix} \begin{bmatrix} \dot{X} \\ \dot{Y} \end{bmatrix} - \dot{s}(1 - c(s)y_e), \\ \dot{y_e} &= \begin{bmatrix} -\sin\theta_v & \cos\theta_v \end{bmatrix} \begin{bmatrix} \dot{X} \\ \dot{Y} \end{bmatrix} - c(s)\dot{s}x_e, \\ \dot{\theta_e} &= \dot{\theta} - c(s)\dot{s}, \end{aligned} \tag{4.3}$$

The path curvature $c(s)$, defined as a function of the arc length s of the path, is an external parameter in the path-following state-space model presented. $c(s)$ is a key parameter that defines movement of moving reference frame P in figure 4-1.

According to [20], the value of $c(s)$ can be calculated by employing the definition of the path coordinates as a function of the arc lengths $x_f(s)$, $y_f(s)$ as well as their

first and second derivatives. The expression is defined as follows:

$$c(s) = \frac{x'_f(s)y''_f(s) - y'_f(s)x''_f(s)}{(x'_f(s)^2 + y'_f(s)^2)^{\frac{3}{2}}}. \tag{4.4}$$

The original path following kinematics, as formulated in [34], utilized an ideal unicycle robot model. This model was later expanded in [15, 16] to incorporate the kinematics of the SSMR by substituting (4.1) into (4.3). The resultant state-space model for path following of the SSMR is presented below, as derived in [16]:

$$\begin{aligned} \dot{x_e} &= v_x \cos\theta_e + x_{ICR}\omega \sin\theta_e - \dot{s}(1 - c(s)y_e), \\ \dot{y_e} &= v_x \cos\theta_e - x_{ICR}\omega \sin\theta_e - c(s)\dot{s}x_e, \\ \dot{\theta_e} &= \omega - c(s)\dot{s}. \end{aligned} \tag{4.5}$$

As stated in [25], the longitudinal and angular velocities (v_x and ω) of the SSMR can be translated to the velocities of its left and right wheels (v_r and v_l), while taking into consideration correction factors α_r and α_l to account for the conditions of the wheels and/or the terrain surface characteristics:

$$v_x = \frac{\alpha_l y_{ICR_r} v_l - \alpha_r y_{ICR_l} v_r}{y_{ICR_r} - y_{ICR_l}}, \quad \omega = \frac{\alpha_l v_l - \alpha_r v_r}{y_{ICR_r} - y_{ICR_l}}. \tag{4.6}$$

4.1.2 MPC Configuration

Control Objective

The main goal of this model predictive control (MPC), is to guide the robot along a predetermined path while also avoiding obstacles. This is accomplished by solving an optimal control problem (OCP) in real-time and utilizing the computed control inputs to drive the robot. It is assumed that the low-level wheel drive controllers of the robot can accurately follow the given references, such that only the target wheel velocity (or acceleration) references need to be generated by the path following control system.

Path Following State-Space Model

The state-space model used by MPC involves a system state represented by a vector x, which includes variables such as x_e, y_e, θ_e, v_l, v_r, and s, and a control input represented by a vector u containing variables a_l and a_r. This model is based on the spatial SSMR kinematic model described previously and is defined as follows:

$$\begin{aligned}
\dot{x_e} &= v_x \cos\theta_e + x_{ICR}\omega \sin\theta_e - \dot{s}(1 - c(s)y_e), \\
\dot{y_e} &= v_x \cos\theta_e - x_{ICR}\omega \sin\theta_e - c(s)\dot{s}x_e, \\
\dot{\theta_e} &= \omega - c(s)\dot{s}, \\
\dot{v_l} &= a_l, \\
\dot{v_r} &= a_r, \\
\dot{s} &= v_x \cos\theta_e + x_{ICR}\omega \sin\theta_e.
\end{aligned} \tag{4.7}$$

The movement of the moving reference frame along the path is determined by the path state variable, s. The velocity of this frame is estimated by approximating the SSMR velocity along its x_e axis.

A compact representation of the system's state-space model is given by:

$$\dot{x} = f(x, u), \tag{4.8}$$

where $f : \mathbb{R}^6 \times \mathbb{R}^2 \to \mathbb{R}^6$ is derived from the equation (4.7).

Optimal Control Problem (OCP)

Formulation the optimal control problem (OCP) that will be solved at each sampling time is facilitated by introduction of the following variables:

- v_s represents the speed of the moving reference frame along the path, which is equal to the SSMR speed along the x_e axis:

$$v_s = v_x \cos\theta_e + x_i cr \sin\theta_e \omega \tag{4.9}$$

- ao, aiB, and aoB represent obstacle avoidance terms and are defined as:

$$ao = e^{-((y_e - y_{e_{obs}})^2 + (x_e - x_{e_{obs}})^2 - (r_{obs} + r_{car})^2)}, \tag{4.10}$$

$$aiB = e^{-((y_e + track\_width/2)^2 - r_{obs}^2)}, \tag{4.11}$$

$$aoB = e^{-((y_e - track\_width/2)^2 - r_{obs}^2)}, \tag{4.12}$$

- h is the state vector that includes the system state and the obstacle avoidance terms, i.e., $h = (x_e, y_e, \theta_e, v_l, v_r, v_s, ao, aiB, aoB)$.

- h_{ref} is the reference state vector, i.e., $h_{ref} = (0, 0, 0, 0, 0, v_{s_{ref}}, 0, 0, 0)$.

The OCP is then formulated as follows:

$$\begin{aligned}
h =& (x_e, y_e, \theta_e, v_l, v_r, v_s, ao, aiB, aoB), \\
h_{ref} =& \ (0, 0, 0, 0, 0, v_{s_{ref}}, 0, 0, 0). \\
\min \ & \frac{1}{2} \int_{t_0}^{t_n} ||h(\tau) - h_{ref}(\tau))||_W^2 \\
& \text{s.t. } \dot{x} = f(x, u), \\
& v_{min} \leq v_l \leq v_{max}, \quad v_{min} \leq v_r \leq v_{max}, \\
& a_{min} \leq a_l \leq a_{max}, \quad a_{min} \leq a_r \leq a_{max},
\end{aligned} \tag{4.13}$$

where W is a symmetrical positive-definite matrix that defines the weights of the cost function. The constraints are imposed point-wise in time for $\tau \in [t_0, t_f]$. The OCP is solved at each sampling instant in a receding-horizon fashion. The state vector h includes the system state and the obstacle avoidance terms (ao, aiB, aoB), while the reference state vector h_{ref} is defined as $(0, 0, 0, 0, 0, v_{s_{ref}}, 0, 0, 0)$. The speed of the moving reference frame along the path (v_s) is calculated as $v_s = v_x \cos\theta_e + x_i cr \sin\theta_e \omega$. The variables ao, aiB, and aoB are defined as repulsive artificial potential fields, where ao represents the obstacle avoidance term, aiB represents the inner border avoidance term, and aoB represents the outer border avoidance term.

MPC Cost, Reference and Constraints

The objective of the MPC controller is to minimize the error between the current state of the robot and the desired reference state, while also taking into account the obstacle avoidance terms. The error coordinates are defined as the deviation of the robot's position, orientation, and velocity from the desired trajectory. By minimizing these error coordinates, the controller can effectively steer the robot back onto the desired trajectory.

To ensure that the robot moves forward along the path, the path velocity term v_s is included in the OCP with a non-zero reference value. This term represents the speed of the moving reference frame along the path, which is approximated as the SSMR velocity along its x_e axis.

In terms of constraints, the velocities and accelerations of the left and right wheels are constrained not to exceed the SSMR motors' limits. This prevents the robot from making sudden, jerky movements that could destabilize it or cause it to lose traction. Additionally, the left and right wheel velocities are penalized in the cost function to prevent excessive wheel speeds during maneuvering.

Real-Time Feedback Procedure

Cubic splines are a common method used to interpolate data points in a smooth manner. They can be used to approximate functions $x_f(s)$ and $y_f(s)$, which are usually the x and y coordinates of a path in the global frame. Cubic splines are piecewise functions that consist of cubic polynomials connected at the waypoints, which are the data points provided. The polynomial coefficients are determined by imposing continuity constraints on the polynomial and its derivatives at each waypoint.

Given the cubic spline approximation of $x_f(s)$ and $y_f(s)$, the curvature $c(s)$ can be computed at each point s along the path using equation (4.4). The first and second derivatives of $x_f(s)$ and $y_f(s)$ can also be computed using the cubic spline approximation with respect to s resulting in values of $x'_f(s)$, $x''_f(s)$ and $y'_f(s)$, $y''_f(s)$. These derivatives are needed to compute the terms in the equation for curvature.

The value of s for each time step is computed using the SSMR's odometry and the MPC's prediction of the SSMR's future motion. This allows the MPC to determine the curvature of the path for each point along the prediction horizon, which is necessary for the SSMR to follow the desired path.

The MPC solver necessitates the initial time-step's system state to be defined, which it uses to compute subsequent time-step states via its system model. According to equation (4.7), the necessary states are x_e, y_e, θ_e, v_l, v_r, and s. However, this project's robot control system lacks a localization module, rendering it incapable of determining its spatial position (x_e, y_e) and orientation (θ_e). Therefore, as the Webots simulation environment keeps track of all objects' positions and orientations, the SSMR's location and orientation were obtained directly from Webots simulator data.

The simulator provides information on the robot's position and orientation relative to the world frame, as well as the velocity of each wheel: x, y, θ, v_l, and v_r. The velocities of the left and right wheels can be utilized directly, whereas the other states necessitate computation using the accessible data.

Initially, it is imperative to assess the path coordinate s of the robot by employing the x and y coordinates alongside the functions $x_f(s)$ and $y_f(s)$. The process for determining s involves seeking the value of s that minimizes the Euclidean distance between the robot's location (x, y) in the world frame and the reference point defined by splines $(x_f(s), y_f(s))$; the golden section search technique was implemented as the search method in this instance.

Once s has been determined, the error terms x_e, y_e, and θ_e can be calculated. The expressions for x_e and y_e are derived from (4.2). As the feedback value of θ from the Webots simulator may abruptly shift from π to $-\pi$ when θ approaches either of these values, θ_e cannot be obtained directly from the simulator data and corresponding expression in (4.2), otherwise it will cause chaotic rotation of SSMR when moving in direction $\pi/-\pi$. Therefore, it was necessary to introduce the vector that represents the robot's orientation as $o_r = (\cos\theta, \sin\theta)$, and the vector that represents the orientation of the moving frame as $o_f = (\dot{x}_f(s), \dot{y}_f(s))$. Subsequently,

θ_e could be determined by cross and dot products as follows:

$$\theta_e = \arctan\left(\frac{||o_r \times o_f||}{o_r \cdot o_f}\right). \tag{4.14}$$

4.1.3 Simulation

Simulations of MATLAB-ACADO MPC were run in WeBots R2020B using Pioneer 3-AT robot platform with manually increased motor speed as well as torque limits. The maximum values for the rotation speed and torque were established as 50 rad/s (which is equivalent to a linear velocity of approximately 5 m/s) and 150 Nm, respectively. Increased motor limits allow for larger variation of target speeds and, consequently, evaluation of MPC performance at higher speeds. Experiments were run on a track with obstacles depicted in figure 3-3.

The simulations were conducted on a computer equipped with an AMD Ryzen 9 5900HX CPU and 32 GB dual-channel RAM. The simulations ran at a frequency of 20Hz, which corresponds to a time-step of 50 ms. The sampling time for the controller integrator was also established as 50 ms.

System Parameters

The path following MPC system's parameters include various factors such as constraints on the robot's wheel speed and acceleration, dimensions of the robot and track, constants of the kinematic model, and weights associated with the cost function.

To replicate motor limitations, the constraints for wheel speeds were established as $|v_l| \leq 5$ and $|v_r| \leq 5$. In addition, the acceleration constraints were set as $|\dot{v_l}| \leq 5$ and $|\dot{v_r}| \leq 5$ to avert the robot from being overturned by excessive torque.

The robot's dimensions were defined based on the specifications mentioned in the Pioneer-3AT documentation, which included the robot's wheel radius and an estimated radius of a circle that encloses the entire robot frame. The wheel radius was used to convert linear speed to angular speed (since Webots motors use angular speed

targets), while the robot radius corresponded to the value utilized in the obstacle avoidance terms. Additionally, the track dimensions comprised solely of the track width, which was determined to be 5.5 m and was utilized in the border avoidance terms.

The constants associated with the kinematic model comprised of $\alpha_l = 1$, $\alpha_r = 1$, $x_{ICR} = 0.05$, $y_{ICR_l} = 0.25$, and $y_{ICR_r} = -0.25$. These values were established so that the kinematic model assumes no wheel slippage and presumes the instantaneous centers of rotation for the left and right wheels to be located in the middle of the robot's corresponding sides. The absence of slippage was assumed because Webots does not accurately simulate wheel slippage.

The weights were adjusted manually to ensure that the robot could successfully complete a lap around the track without any collisions or border violations while traveling at speeds of up to 3 m/s. This resulted in a weight vector of $W = \begin{bmatrix} 5 & 5 & 10 & 1 & 1 & 100 & 120 & 10 & 10 \end{bmatrix}$.

Results

Assessment the effectiveness of the MPC control system was done under different parameters for the target velocity and prediction horizon.

The initial evaluation of the controller involved assessing the robot's ability to follow a path at various target velocities. Figures **??** show that the SSMR was able to complete a lap accurately while adhering to several different target velocities. The diagrams reveal that the SSMR was able to maintain the target velocity for the majority of the lap, with only minor deviations around obstacles. The corresponding tracking error diagrams can be found in Figs. 4-5 and 4-6(b), which indicate that the tracking errors were low in the absence of obstacles. However, the tracking errors were noticeably higher at faster speeds, which was visually observed during real-time simulations and recorded as average errors in Table 4.1. To assess the tracking accuracy without obstacles, the average error was calculated from the second half of the track, where there are no obstacles.

The impact of the MPC prediction horizon was analyzed by maintaining a constant

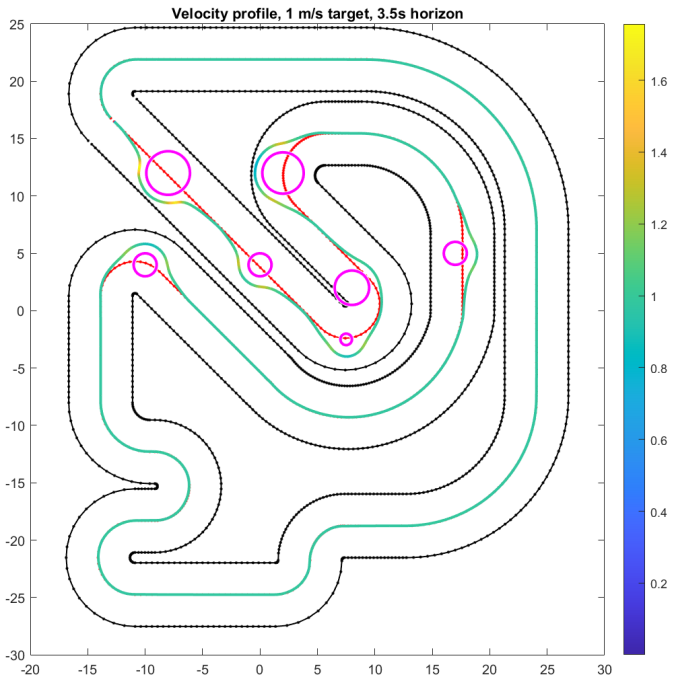


Figure 4-2: Velocity profiles, 1 m/s reference speed 3.5s horizon.

Table 4.1: MATLAB-ACADO MPC Computation Time

Target speed	1m/s	2m/s	3m/s
Average error	0.0155m	0.0255m	0.0501
Average computation time	4.56ms	4.66 ms	6.10ms

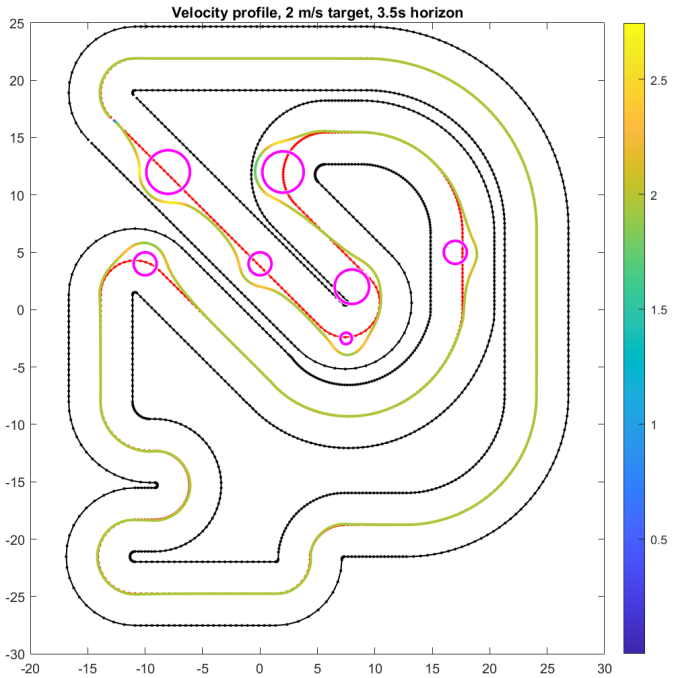


Figure 4-3: Velocity profiles, 2 m/s reference speed 3.5s horizon.

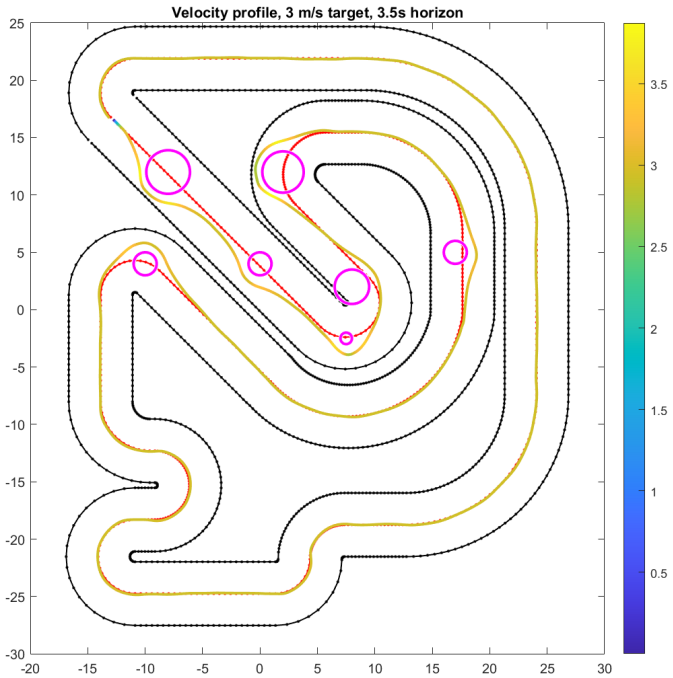
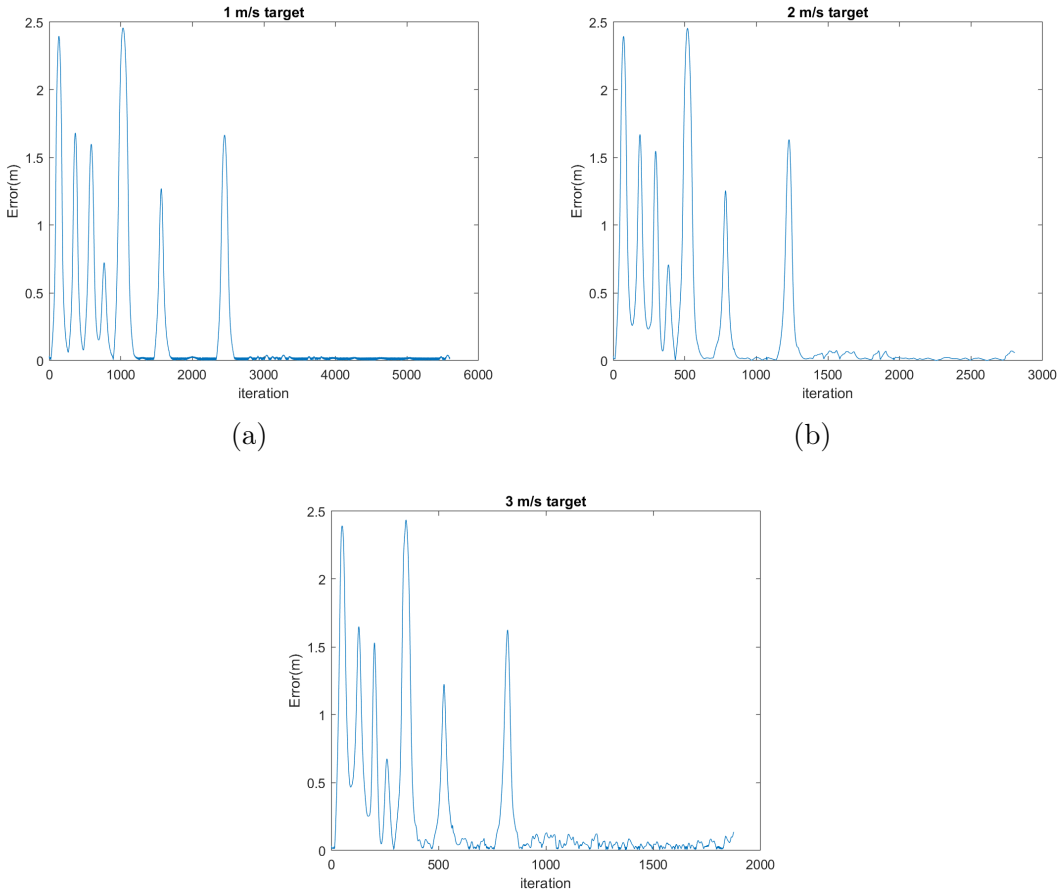


Figure 4-4: Velocity profiles, 3 m/s reference speed 3.5s horizon.



(a)

(b)

(c)

Figure 4-5: Deviation from path, 3.5s horizon.

target robot speed of 1 m/s while using prediction horizons of 3.5s, 7s, and 10.5s. Longer prediction horizons allow the MPC to generate control signals that minimize deviation from the target trajectory around obstacles, as illustrated in Fig. 4-6(a).

Figure 4-7 illustrates the computation time of the MPC for each sampling interval during a speed target variation test. It can be observed that, following the first time step, the worst-case computation time does not surpass the 50ms sampling interval and averages just 6.10ms for a 3 m/s target, as shown in Table 4.1.

The tests carried out on the MPC approach have indicated that the design objectives were successfully achieved while maintaining a reasonable computational cost. The MPC algorithm was able to generate control signals that allowed the SSMR to achieve speeds of up to 3 m/s while following the desired trajectory and avoiding obstacles.

4.2 Time-domain MPC in ACADOS-Python

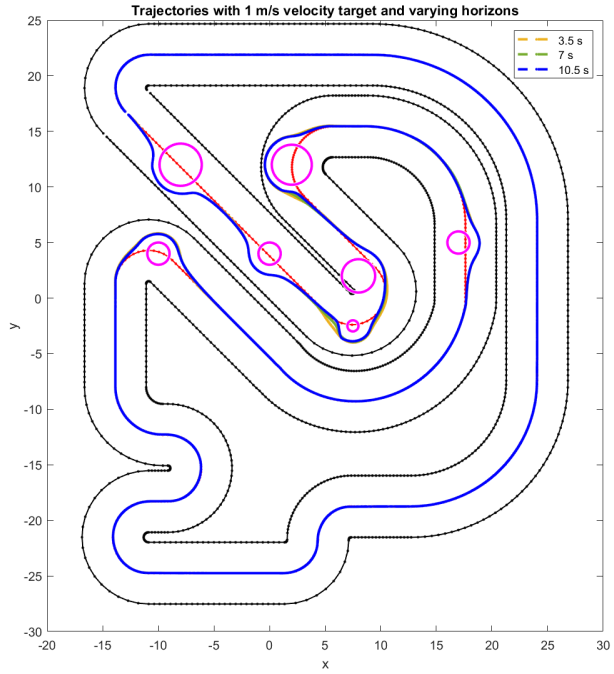
Implementation of MPC in time domain using ACADOS-Python is based on kinematic model shown in [21] and was designed to reach a desired point in fixed world reference frame.

4.2.1 Time-Domain Kinematic Model of SSMR

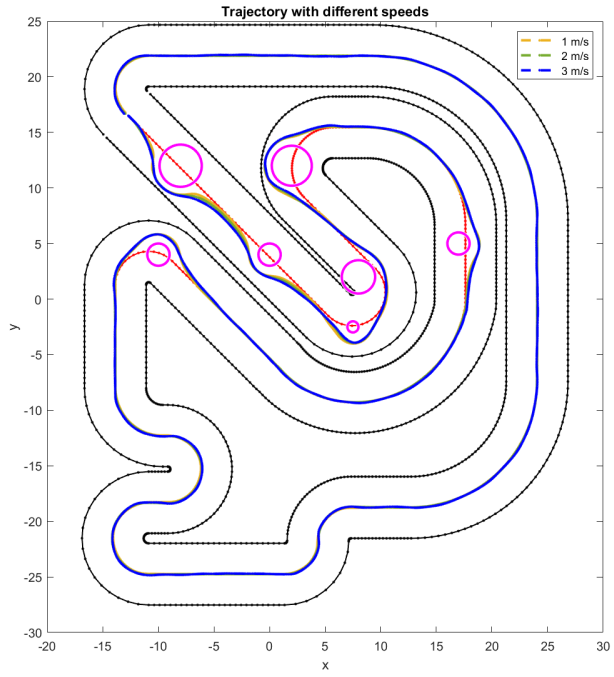
In [21], the motion of the SSMR in global coordinates is described as:

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & x_{ICR}\sin\theta \\ \sin\theta & -x_{ICR}\cos\theta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ \omega \end{bmatrix}, \tag{4.15}$$

Here, X, Y, θ denote two position and one orientation coordinates, while v_x, and ω are the longitudinal and angular velocities of the robot.

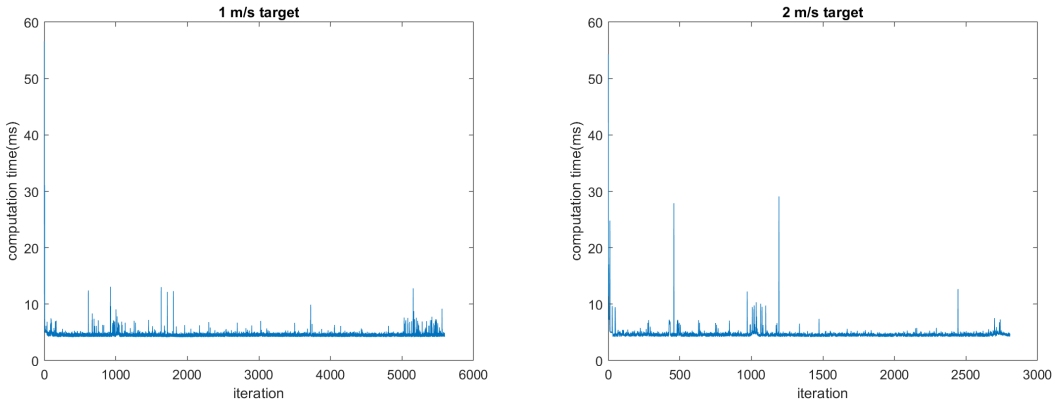


(a)



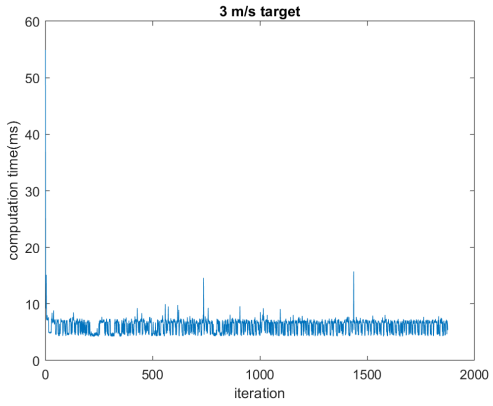
(b)

Figure 4-6: Deviation from path, 3.5 s horizon.



(a)

(b)



(c)

Figure 4-7: Computation times, 3.5s horizon.

4.2.2 MPC Configuration

Control Objective

MPC design objective here is to reach a designated target point in a simulation scene with empty floor depicted in 3-1. It is assumed that the robot's low-level wheel drive controllers can precisely track the specified references, so the control system only needs to generate the desired wheel velocity or acceleration references. Real-time control of the robot is achieved by solving an optimal control problem (OCP) and using the resulting control inputs to direct the robot.

State-Space Model of SSMR

Based on above kinematic model, SSMR system's state-space model includes a system state vector x comprised of x, y, θ, v_l, v_r as well as control vector u containing variables a_l and a_r. Variables a_l and a_r are accelerations of left and right wheels respectively. As stated previously, v_x, and ω can be mapped to wheel velocities using following SSMR model and wheel velocity mapping expressions:

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & x_{ICR}\sin\theta \\ \sin\theta & -x_{ICR}\cos\theta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ \omega \end{bmatrix}, \tag{4.16}$$

$$v_x = \frac{\alpha_l y_{ICR_r} v_l - \alpha_r y_{ICR_l} v_r}{y_{ICR_r} - y_{ICR_l}}, \quad \omega = \frac{\alpha_l v_l - \alpha_r v_r}{y_{ICR_r} - y_{ICR_l}}. \tag{4.17}$$

So, the state-space model can be defined as:

$$\begin{aligned} \dot{x} &= v_x \cos\theta + x_{ICR}\omega\sin\theta, \\ \dot{y} &= v_x \cos\theta - x_{ICR}\omega\sin\theta, \\ \dot{\theta} &= \omega, \\ \dot{v_l} &= a_l, \\ \dot{v_r} &= a_r \end{aligned} \tag{4.18}$$

And its compactified form:

$$\dot{x} = f(x, u), \tag{4.19}$$

Here, $f : \mathbb{R}^5 \times \mathbb{R}^2 \to \mathbb{R}^5$ represents the equation (4.18).

Optimal Control Problem (OCP)

First, state vector and corresponding reference vector are defined:

- h is the state vector that includes the system state and control signal, i.e., $h = (x, y, \theta, v_l, v_r, a_r, a_l)$.

- h_{ref} is the reference state vector, i.e., $h_{ref} = (x_{ref}, y_{ref}, 0, 0, 0, 0, 0)$.

Setting W as a symmetrical positive-definite matrix that defines the weights of the cost function. The OCP becomes:

$$\begin{aligned}
h =& (x, y, \theta, v_l, v_r, a_l, a_r), \\
h_{ref} =& (x_{ref}, y_{ref}, 0, 0, 0, 0, 0). \\
\min \frac{1}{2} &\int_{t_0}^{t_n} ||h(\tau) - h_{ref}(\tau))||^2_W \\
&\text{s.t. } \dot{x} = f(x, u), \\
&v_{min} \leq v_l \leq v_{max}, \quad v_{min} \leq v_r \leq v_{max}, \\
&a_{min} \leq a_l \leq a_{max}, \quad a_{min} \leq a_r \leq a_{max},
\end{aligned} \tag{4.20}$$

MPC Cost, Reference and Constraints

This MPC is made to reach a point in 2-dimensional space designated by x_{ref}, y_{ref} in reference vector h_{ref} by minimizing difference between current state and the reference. Optimization constraints were set for wheel speed and acceleration to account for limits of simulated robot model and prevent excess torque.

Real-time Feedback Procedure

Feedback data is taken directly from WeBots simulation, which keeps track of robot's position, orientation, linear and angular velocity, as well as its motors' speeds. This allows to construct a vector containing robot system's current state as (x, y, θ, v_l, v_r).

4.2.3 Simulation

ACADOS-Python MPC was tested in WeBots R2023a on pioneer 3-AT robot platform using empty floor environment shown in figure 3-1. MPC time-step was set to 50ms corresponding to 20 Hz control rate.

It must be noted that publicly available version of WeBots R2023a has a bug in its Python interface that prevents control over motors' acceleration. WeBots R2023a used in this project was modified to fix this issue.

System Parameters

Parameters present in the MPC system are: robot's wheel speed and acceleration limits, constants of the kinematic model, and weights associated with the cost function.

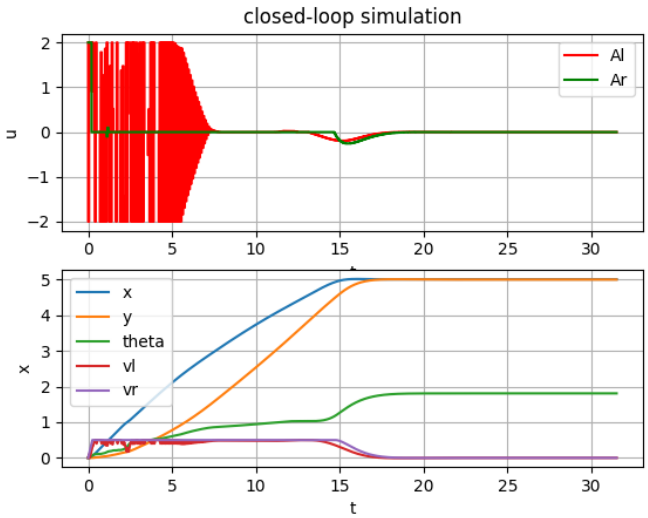
The constraints for wheel speeds were set according to motor limits of default pioneer 3-AT platform: $|v_l| \leq 0.5$ and $|v_r| \leq 0.5$. The acceleration constraints were set as $|\dot{v_l}| \leq 2$ and $|\dot{v_r}| \leq 2$ to ensure smooth movement.

Kinematic model constants were set to $\alpha_l = 1$, $\alpha_r = 1$, $x_{ICR} = 0.05$, $y_{ICR_l} = 0.25$, and $y_{ICR_r} = -0.25$, similar to section 4.1.3.

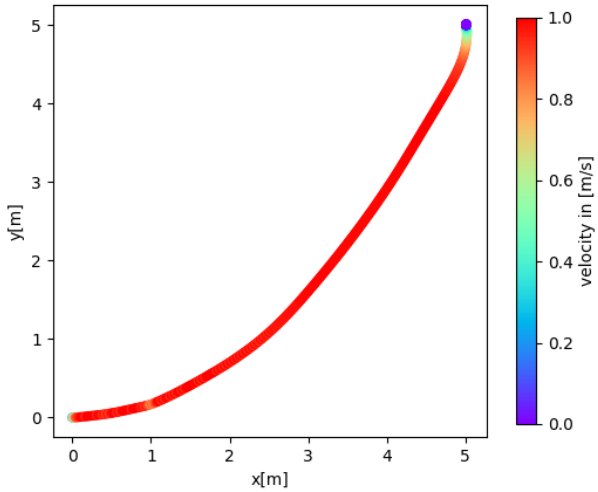
The weight vector was set as $W = \begin{bmatrix} 5 & 5 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$. These values are result of manual tuning with aim to ensure that the SSMR reliably reaches target point.

Result

In offline simulation the controller has successfully generated control signals and corresponding path to target location as illustrated on figures 4-8.



(a)



(b)

Figure 4-8: Time-Domain ACADOS-Python MPC offline simulation.

In WeBots real-time simulation, the MPC controller was successfully implemented and was able to repeatedly reach desired target point as shown on figure 4-9.

4.3 Time-Domain Data-Driven MPC in ACADOS-Python

Time-domain data-driven MPC in ACADOS-Python was implemented as an extension of MPC described in section 4.2 by integrating it with PyTorch-CASADI framework proposed in [31] and replacing nominal kinematics with a neural network's local approximation.

4.3.1 Data Generation

Data-driven dynamics model was designed similar to approach of [35] in a sense that the model was trained to be a standalone representation of SSMR movement. This was preferable for prototyping stage as it allowed to artificially generate arbitrary number of data points using a nominal model and uniform random distribution, avoiding time-consuming and computationally intensive data generation through simulations. The neural network model was trained on such data with an aim to replicate nominal model used previously. In addition to data generated via nominal model, data set was also supplemented with limited number of samples gathered from MATLAB-ACADO implementation described in section 4.1. Additional simulation data was useful in development of model training procedure as it made the data more realistic, complex and, consequently, more challenging to learn. Nevertheless, gathering of simulation data is computationally expensive, especially at lower speeds; simulation data collected for this project was not sufficient to produce robust dynamics model prompting combination of artificial and simulation data.

Training data set gathered in WeBots simulation contained data on system states and control signals $v_l, v_r, \theta, a_l, a_r$ for inputs and system state derivatives $\dot{x}, \dot{y}, \dot{\theta}, \dot{v_l}, \dot{v_r}$ as output labels. The parameters were gathered directly from simulator and, as such,

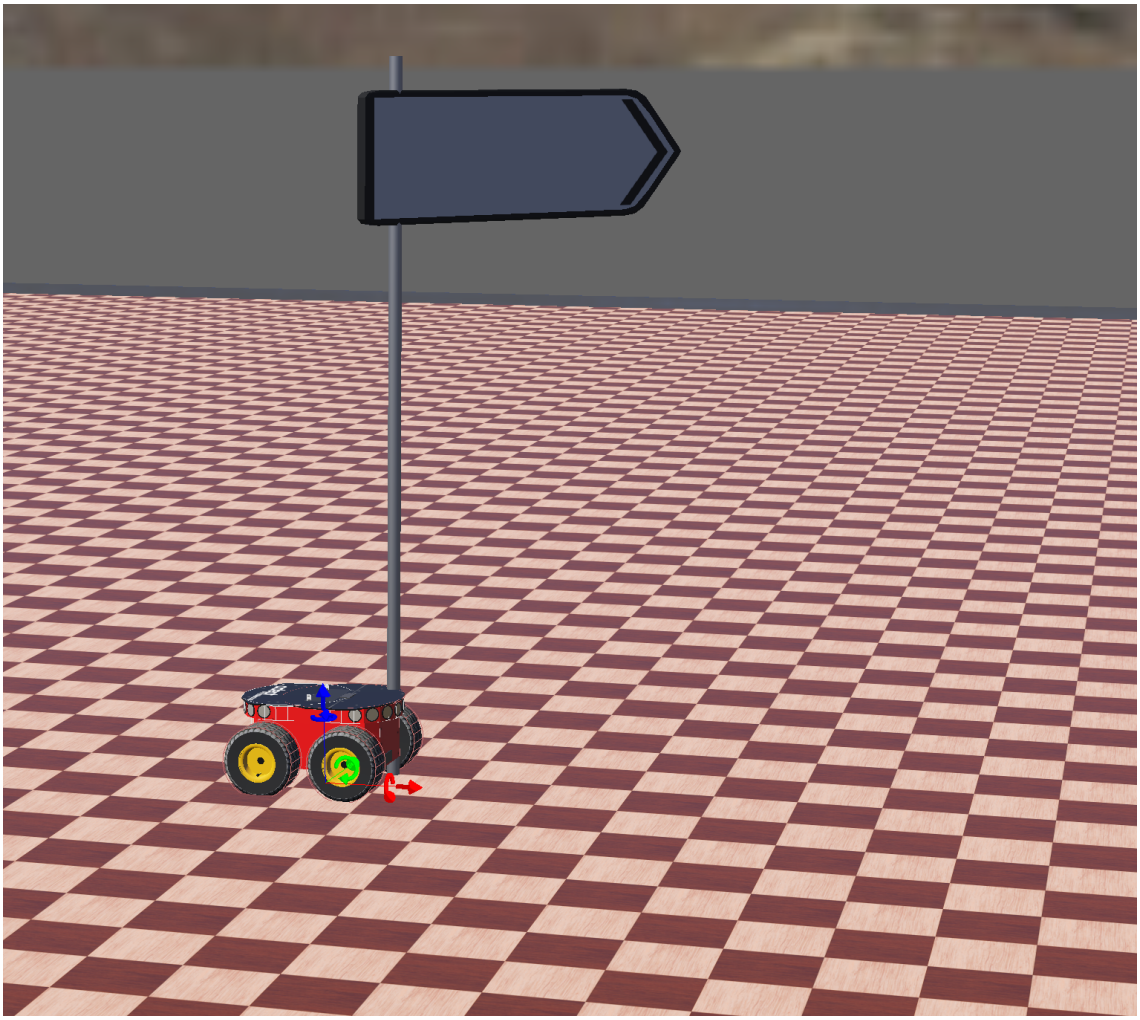


Figure 4-9: Time-domain implementation of MPC reaching the target.

were not subject to measurement errors. The simulation procedure was MPC driving an SSMR along a track with obstacles for 15 laps with target speeds varying from $0.125m/s$ to $0.5m/s$.

Artificial data was generated based on state space model 4.18 and uniform random distribution. Input vectors containing $v_l, v_r, \theta, a_l, a_r$ were generated using uniform random distributions with ranges based on simulation parameters discussed in section 4.2.3 for velocity as well as acceleration, and WeBots feedback range for SSMR orientation θ: $-0.5 \leq v_l \leq 0.5$, $-0.5 \leq v_r \leq 0.5$, $-2 \leq a_l \leq 2$, $-2 \leq a_r \leq 2$, $-\pi \leq \theta \leq \pi$. Output vectors with $\dot{x}, \dot{y}, \dot{\theta}, \dot{v_l}, \dot{v_r}$ were calculated by substituting generated inputs into expressions 4.18 and 4.17.

4.3.2 Architecture and Training

Neural network architecture is a simple multi-layer perceptron with 3 hidden layers, which are 32, 128 and 128 units wide, ReLU activation, dropout layer, and a linear output head. The architecture is depicted in figure 4-10.

Dataset was split into train and test sets with 20% ratio for test set, while both input and output data was normalized to range from -1 to 1. Normalization was done in accordance with limits on speed and acceleration discussed previously. For instanse, the distribution of $\dot{x}$ is depicted in figure 4-11.

Training was done for up to 500 epochs using Adam optimizer [19] with early stopping and learning rate schedule. Early stopping used test set for validation and was set for patience of 50 epochs. Learning rate schedule was configured with initial value of 0.01 and was factored by 0.1 every 50 epochs. The training was run for 300 epochs and resulted in 0.73 R2 score. Loss graph is depicted on figure 4-12.

4.3.3 MPC Configuration

Control Objective

As with time-domain MPC discussed previously, objective here is to reach a target point in the environment depicted in 3-1 under same assumptions. The difference

```
KinematicModel(
  (fc1): Linear(in_features=5, out_features=32, bias=True)
  (relu3): ReLU()
  (fc2): Linear(in_features=32, out_features=128, bias=True)
  (relu4): ReLU()
  (dropout): Dropout(p=0.2, inplace=False)
  (fc3): Linear(in_features=128, out_features=5, bias=True)
)
```

Figure 4-10: Neural network architecture.

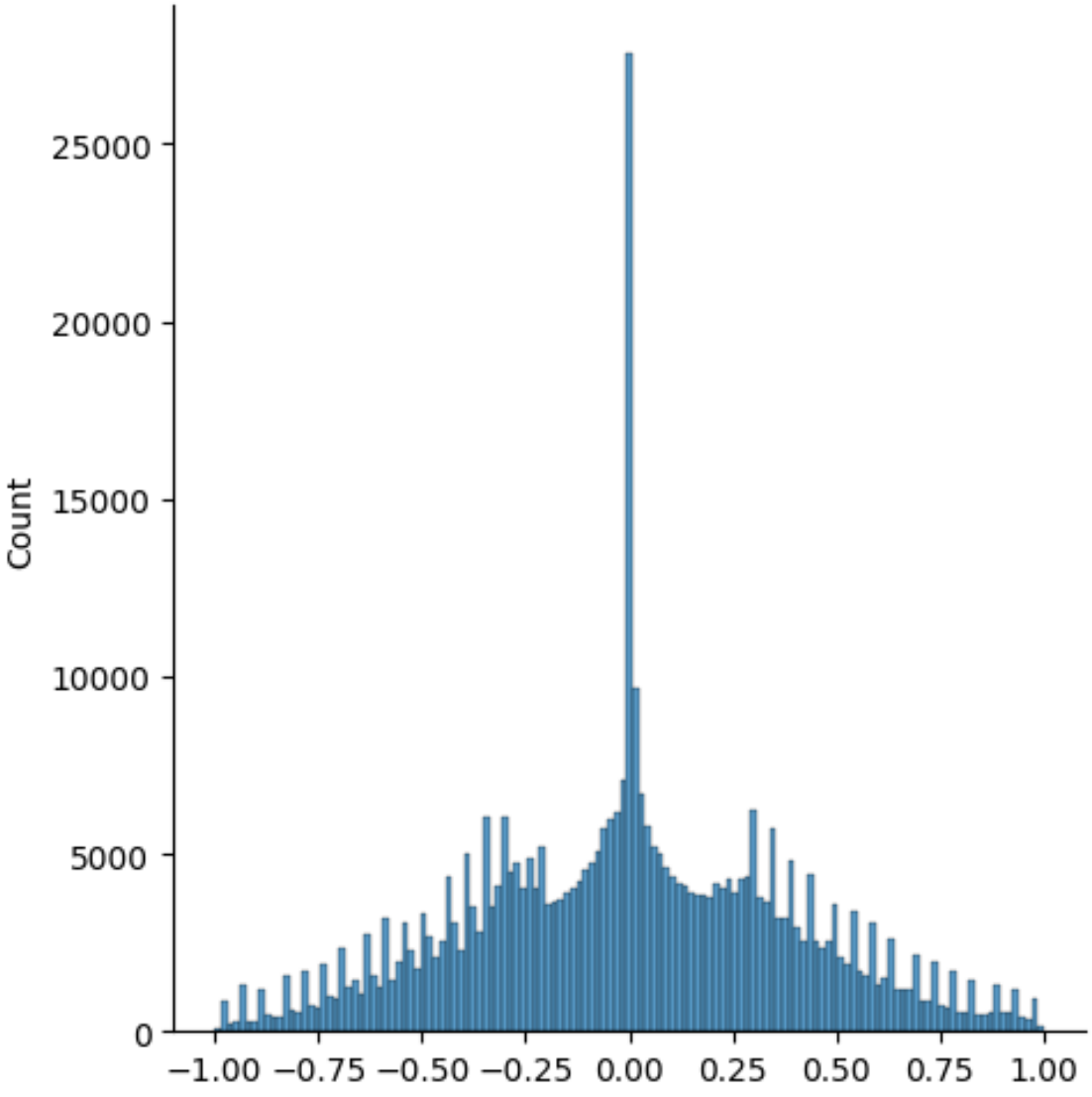


Figure 4-11: $\dot{x}$ distribution.

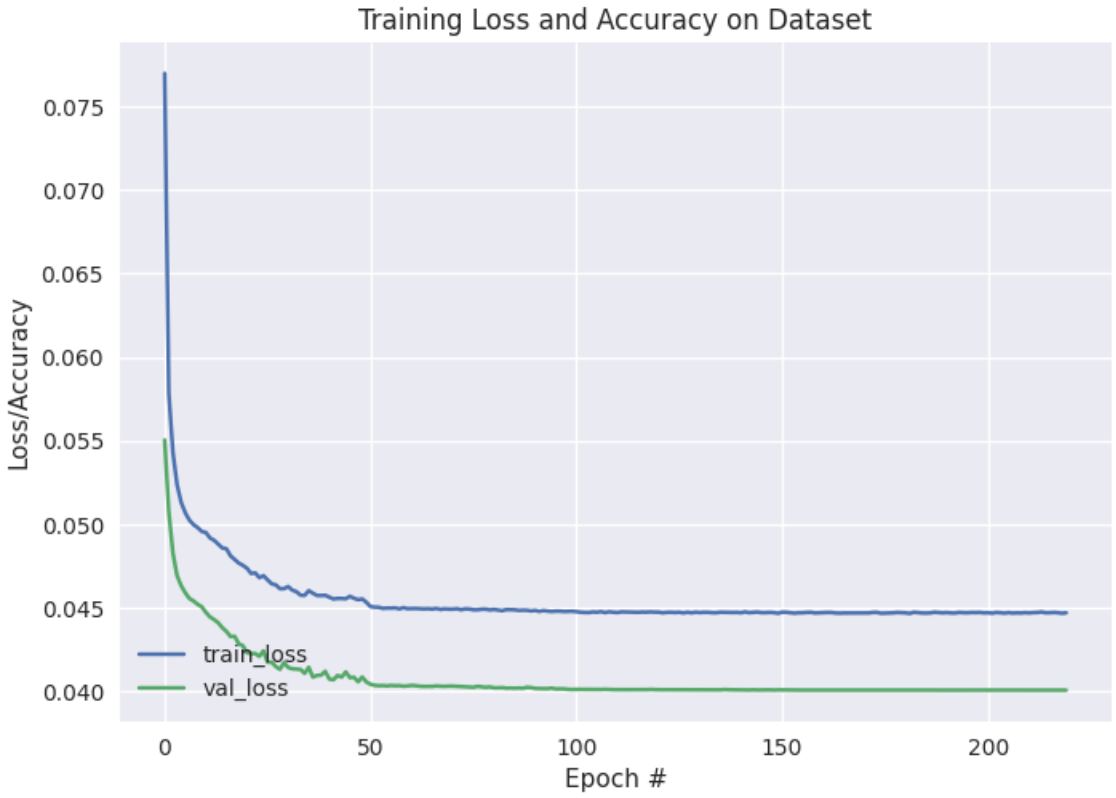


Figure 4-12: Data-driven system model training and validation loss.

lies in the system's state-space model that is now represented by an approximation of the trained neural network. As before, MPC uses learned state-space model to solve optimal control problem (OCP) and applies the resulting control signals to the SSMR.

State-Space Model of SSMR

In the learned state-space model derivatives of system states x, y, θ, v_l, v_r are predicted by the data-driven model based on system current state and control signal.

So, the state-space model is defined using short notation:

$$\dot{x} = f_{NN}(x, u), \tag{4.21}$$

Function $f : \mathbb{R}^5 \times \mathbb{R}^2 \to \mathbb{R}^5$ used previously is now replaced with function representing the learned model $f_{NN} : \mathbb{R}^5 \times \mathbb{R}^2 \to \mathbb{R}^5$.

4.3.4 Simulation

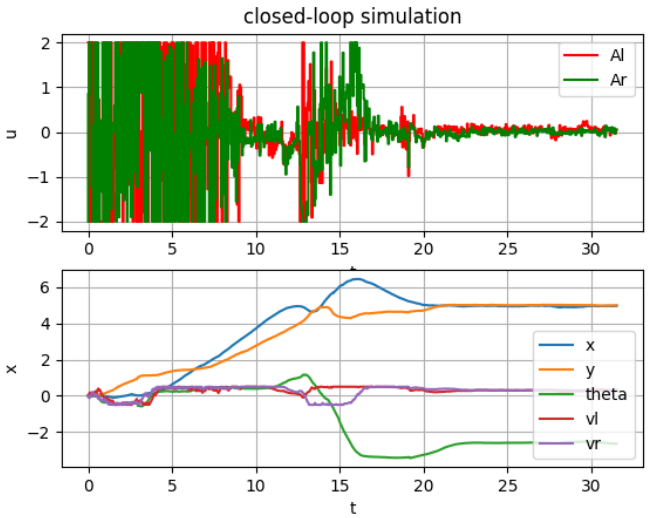
Optimal Control Problem, MPC cost, reference, constraints, system parameters and feedback data were unchanged relative to Time-Domain MPC of section 4.2.

Offline simulation demonstrated in figure 4-13 shows that the data-driven MPC successfully computes a path towards a goal with corresponding control inputs, albeit with some discontinuity inherent to neural network.

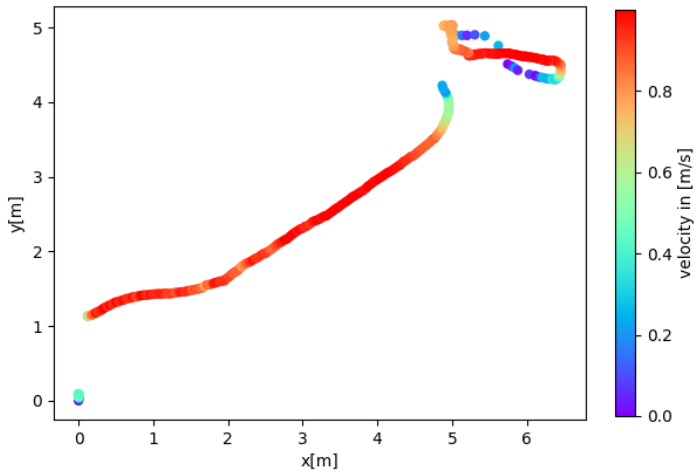
In real-time simulation, since SSMR movement is no longer estimated by learned model itself, data-driven MPC was able to reliably reach the target despite discontinuity seen in offline tests (figure 4-14).

4.4 Spatial-Based MPC in ACADOS-Python

Spatial-based MPC in ACADOS-Python is re-implementation of MPC strategy discussed in section 4.1 using ACADOS optimization framework and Python environment, with minor modification. At this stage, static obstacle avoidance feature was



(a)



(b)

Figure 4-13: Time-Domain data-driven ACADOS-Python MPC offline simulation.

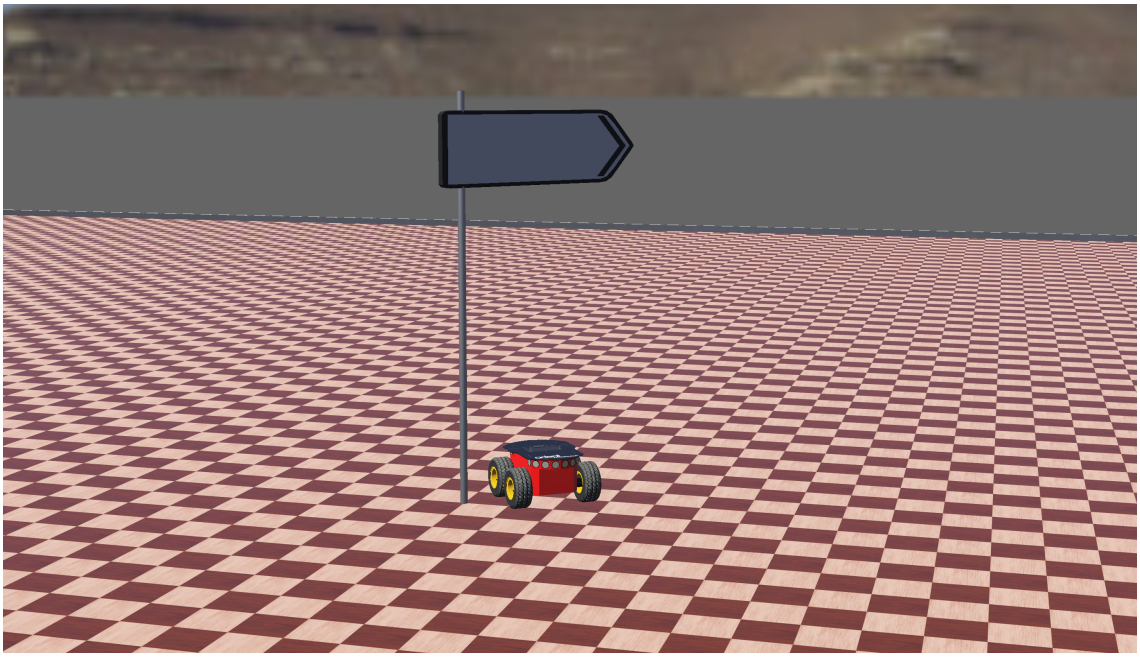


Figure 4-14: Time-domain implementation of data-driven MPC reaching the target.

omitted as it was not a priority in development of data-driven MPC, while border avoidance approach changed from repulsive potential field to soft-constraints. The control strategy is still derived from spatial kinematic model of [16].

4.4.1 MPC configuration

Control Objective

Similar to MATLAB-ACADO version, control objective is to drive the SSMR along a reference trajectory in real-time while avoiding borders, without static obstacle avoidance.

Optimal Control Problem (OCP)

SSMR kinematic model and state-space model remain as MATLAB-ACADO implementation of section 4.1. However, as ACADOS does not allow introduction of custom terms into its default cost function and requires programming of special cost function CASADI for that functionality, OCP was modified to target path progress s for simplicity.

Introducing following variables:

- $h = (x_e, y_e, \theta_e, v_l, v_r, s, a_l, a_r)$ – the state vector that includes the system states.
- $h_{ref} = (0, 0, 0, 0, 0, s_{ref}, 0, 0)$ – the reference state vector.
- W – a symmetrical positive-definite matrix containing the weights of the cost function.

Results in following OCP:

$$
\begin{aligned}
h =& (x_e, y_e, \theta_e, v_l, v_r, s, a_l, a_r), \\
h_{ref} =& \ (0, 0, 0, 0, 0, s_{ref}, 0, 0). \\
\min \frac{1}{2} &\int_{t_0}^{t_n} ||h(\tau) - h_{ref}(\tau))||^2_W \\
&\text{s.t. } \dot{x} = f(x, u), \\
&v_{min} \leq v_l \leq v_{max}, \quad v_{min} \leq v_r \leq v_{max}, \\
&a_{min} \leq a_l \leq a_{max}, \quad a_{min} \leq a_r \leq a_{max}, \\
&y_{e_{min}} \leq y_e \leq y_{e_{max}},
\end{aligned}
\tag{4.22}
$$

MPC Cost, Reference and Constraints

The OCP above aims to minimize error coordinates which represent difference between reference trajectory and SSMR trajectory. This makes the controller steer the SSMR along reference trajectory.

To facilitate SSMR movement forward along the reference path, the path targets at each time-step are set as:

$$
s_{i_{ref}} = s_{current} + iT_s v_{s_{ref}}, i \in [1, N] \tag{4.23}
$$

where

- i – time-step.
- $s_{i_{ref}}$ – path reference at time-step i.
- $s_{current}$ – current position on path found via search algorithm discussed in section 4.1.2.
- T_s – time-step duration.
- $v_{s_{ref}}$ – target speed along the path.

Wheel velocity and acceleration constraints were set according to limitations of default pioneer 3-AT model's motors. Constraint on y_e error coordinate was set to reflect track width.

Feedback Procedure

Feedback data collection and computation approach was unchanged from MATLAB-ACADO version in principle; the methods were re-implemented in Python using scipy's library for cubic splines and numpy.

4.4.2 Simulation

Experiments were run using modified WeBots R2023a simulator (discussed in section 4.2.3) and included pioneer 3-AT robot model. Control and simulation frequency were established at 20Hz.

System Parameters

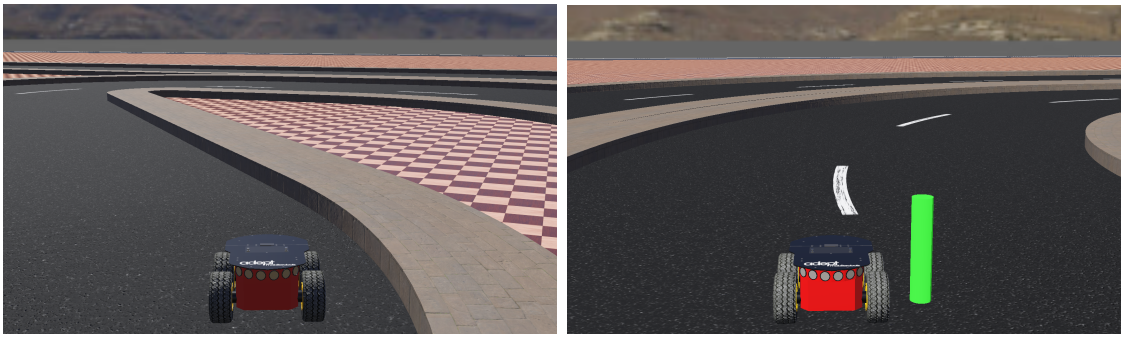
System parameters are values for constraints on the robot's wheel speed and acceleration as well as y_e coordinate, constants of the kinematic model, and weights associated with the cost function. Kinematic model constants were set to same values as in MATLAB-ACADO implementation.

Constraints on wheel speed and acceleration were set to $|v_l| \leq 0.5$, $|v_r| \leq 0.5$, $|a_l| \leq 2$, $|a_r| \leq 2$, in accordance with default pioneer 3-AT model. y_e coordinate was constrained based on track width of 5 meters as $|y_e| \leq 2.5$.

Weight vector was configured to facilitate smooth navigation along the track within road borders at maximum speed of 0.5 m/s. So, value of weight were set as $W = \begin{bmatrix} 25 & 25 & 5 & 1 & 1 & 10 & 0 & 0 \end{bmatrix}$.

Results

This implementation of path following in ACADOS-Python framework MPC was able to successfully achieve its design objectives and perform effective navigation of SSMR



(a) (b)

Figure 4-15: Spatial ACADOS-Python MPC WeBots simulation.

along a reference trajectory as depicted on figure 4-15.

4.5 Spatial-Based Data-Driven MPC in ACADOS-Python

Spatial-based data-driven MPC was developed by extending ACADOS-Python path following MPC with Pytorch-Casadi integration presented in [31] and combining nominal kinematic model with learned model described in section 4.3.

4.5.1 Learned Model Architecture and Training

Neural network model was taken from implementation of Time-Domain data-driven MPC discussed in section 4.3. The model was developed based on fixed global reference frame, so it was assumed that, considering absence of direction-dependent dynamics in simulated surface, the model is transferable to moving reference frame of spatial formulation.

4.5.2 MPC Configuration

Control Objective

Goal of MPC remains as in nominal model spatial MPC of section 4.4: drive SSMR along a reference path without border violation.

State-Space Model of SSMR

Compared to state-space model used in MATLAB and Python implementations of path following MPC discussed in sections 4.1 and 4.4. Data-driven state-space model employed here is a hybrid of nominal model proposed in [16] and learned model developed in section 4.3.

Considering system state vector $x = [x_e, y_e, \theta_e, v_l, v_r, s]$ and parameter K defining ratios of nominal and learned models' contributions to state-space model. The state-space model is defined using short notation:

$$\dot{x} = Kf(x, u) + (1 - K)f_{NN}(x, u), \tag{4.24}$$

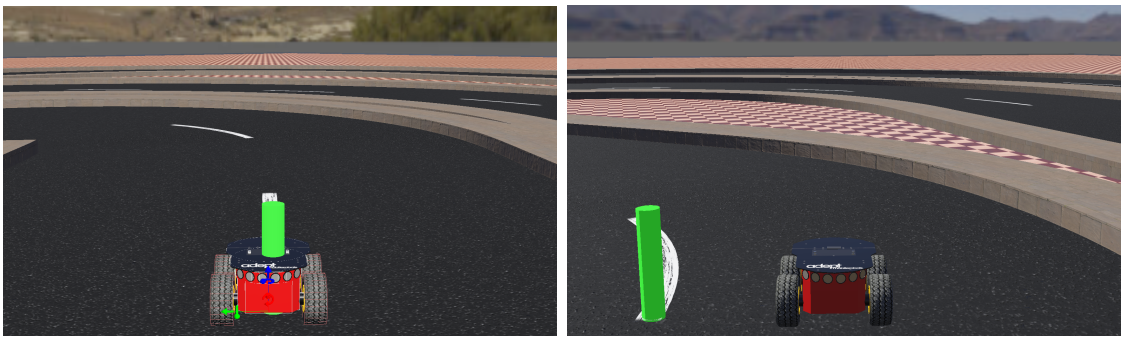
Function $f : \mathbb{R}^6 \times \mathbb{R}^2 \to \mathbb{R}^6$ is combined with learned $f_{NN} : \mathbb{R}^5 \times \mathbb{R}^2 \to \mathbb{R}^5$. f is a nominal model used in MATLAB-ACADO as well as Python-ACADOS versions of path following MPC, while f_{NN} is a data-driven model presented in section 4.3. f_{NN} predicts only 5 values out of 6, so the s state remains unaffected by data-driven model.

4.5.3 Simulation

Apart from state-space model, the OCP, cost function, reference computation, system constraints and parameters, weight vector, as well as feedback procedure were kept same as path following MPC of section 4.4.

Result

Figure 4-16 suggests that integration of Pytorch-Casadi framework was successful as, with contribution ratio K set 0.5, new MPC was able to drive the SSMR along reference path similarly to version without learned model component; model learned on fixed frame data is at least partially transferable to moving frame case. However, K set to 0 makes the system fully dependent on learned model, in such case SSMR exhibits inconsistent behavior and fails progress along the track. It appears that



(a) (b)

Figure 4-16: Spatial data-driven ACADOS-Python MPC WeBots simulation.

nominal component is required for OCP to resist inconsistency and discontinuity of learned model observed in offline simulation of section 4.3.4.

Chapter 5

Conclusion

This thesis project has focused on developing of model predictive controllers for skid-steered mobile robots (SSMR). Developed SSMR control strategies include spatial-based and obstacle avoiding path following control in MATLAB-ACADO framework, time-domain control in Python-ACADOS framework, data-driven time-domain control in Python-ACADOS framework, spatial-based path following control in Python-ACADOS framework, data-driven spatial-based path following control in Python-ACADOS framework.

First MPC implementation, completed using MATLAB environment and ACADO [14] optimization software, is a spatial-based and obstacle avoiding path following control of SSMR that was based on spatial kinematic model discussed in [15, 16]. Extensive testing has demonstrated that the MPC was effective at following reference trajectory without collisions with any stationary obstacles.

Other implementations were done using Python-ACADOS framework, which includes Python as software environment and ACADOS [37] as optimization framework. Time-domain control in Python-ACADOS framework was developed with a goal to navigate SSMR towards a target point using time-domain variant of SSMR kinematic model proposed in [21]. This control approach was able to reliably reach a target location in simulation experiments.

Data-driven time-domain control was created by expanding time-domain control in Python-ACADOS framework via PyTorch-Casadi framework presented in [31].

PyTorch-Casadi framework allows to approximate arbitrary PyTorch model in form of Casadi expression, which can be included into ACADOS optimal control problem. This has allowed to create MPC strategy that included data-driven components in its state-space model. To avoid time-consuming and computationally expensive data collection and model training, the approach was created using a compact two-layer neural network trained on combination of small batch of simulation data and nominal kinematic equations. The MPC was tested in WeBots simulation, which has shown that it is able to drive towards target point similar to version without learned dynamics.

Python-ACADOS implementation of spatial-based path following of MPC was made to be a re-implementation of the MATLAB-ACADO version in new framework. It used same kinematic model and new Python implementations of methods used in MATLAB-ACADO. Python-ACADOS version lacks static obstacle avoidance feature to simplify prototyping and has minor modification to cost function that is related to ACADOS. The re-implementation was able to successfully travel along reference path during testing in WeBots simulation with a track.

Similar to data-driven time-domain control, data-driven spatial-based path following control was developed as an extension of spatial-based path following control in Python-ACADOS framework. Data-driven dynamics model from data-driven time-domain control implementation was combined with nominal kinematics of spatial-based path following control to create a hybrid state-space model of SSMR. The hybrid model was incuded into optimal control problem to create a new data-driven approach. The approach was tested in WeBots track simulation confirming its ability to drive SSMR along reference trajectory.

Future development plan is to improve on latest python implementation by extending it path following contouring control version and designing an effective deep learning based system model. Furthermore, considering that python code much more portable, it is planned to deploy and test the MPC in more advanced simulators and real robotic platforms.

Bibliography

[1] NVIDIA Isaac Sim: a scalable robotics simulation application and synthetic data-generation tool. `https://developer.nvidia.com/isaac-sim`. Accessed: 2022-11-05.

[2] Webots: Open Source Robot Simulator. `https://cyberbotics.com`. Accessed: 2022-11-05.

[3] Joel A. E. Andersson, Joris Gillis, Greg Horn, James B. Rawlings, and Moritz Diehl. Casadi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, Mar 2019.

[4] Dominic Baril, Vincent Grondin, Simon-Pierre Deschênes, Johann Laconte, Maxime Vaidis, Vladimír Kubelka, André Gallant, Philippe Giguère, and François Pomerleau. Evaluation of skid-steering kinematic models for subarctic environments. In *2020 17th Conference on Computer and Robot Vision (CRV)*, pages 198–205, 2020.

[5] Bruno Brito, Boaz Floor, Laura Ferrari, and Javier Alonso-Mora. Model predictive contouring control for collision avoidance in unstructured dynamic environments. *IEEE Robotics and Automation Letters*, 4(4):4459 – 4466, 2019.

[6] Alexander Buyval, Aidar Gabdulin, Ruslan Mustafin, and Ilya Shimchik. Deriving overtaking strategy from nonlinear model predictive control for a race car. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2623–2628, 2017.

[7] Luca Caracciolo, Alessandro De Luca, and Stefano Iannitti. Trajectory tracking control of a four-wheel differentially driven mobile robot. In *Proceedings 1999 IEEE international conference on robotics and automation (Cat. No. 99CH36288C)*, volume 4, pages 2632–2638, 1999.

[8] Kong Yao Chee, Tom Z. Jiahao, and M. Ani Hsieh. Knode-mpc: A knowledge-based data-driven predictive control framework for aerial robots. *IEEE Robotics and Automation Letters*, 7(2):2819–2826, 2022.

[9] Nikolaus Correll, Bradley Hayes, Christoffer Heckman, and Alessandro Roncone. *Introduction to Autonomous Robots: Mechanisms, Sensors, Actuators, and Algorithms*. Mit Press, 2022.

[10] J. V. Frasch, A. J. Gray, M. Zanon, H. J. Ferreau, S. Sager, F. Borrelli, and M. Diehl. An auto-generated nonlinear MPC algorithm for realtime obstacle avoidance of ground vehicles. In *2013 European Control Conference (ECC)*, page 4136–4141, 2013.

[11] Yiqi Gao, Andrew Gray, Janick V. Frasch, Theresa Lin, Eric Tseng, J. Karl Hedrick, and Francesco Borrelli. Spatial predictive control for agile semi-autonomous ground vehicles. In *11th International Symposium on Advanced Vehicle Control*, 2012.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[13] Lukas Hewing, Juraj Kabzan, and Melanie N. Zeilinger. Cautious model predictive control using gaussian process regression. *IEEE Transactions on Control Systems Technology*, 28(6):2736–2743, 2020.

[14] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. ACADO toolkit - an open-source framework for automatic control and dynamic optimization. *Optim Contr Appl Met*, 32(3):298–312, 2011.

[15] G. Huskic, S. Buck, and A. Zell. Path following control of skid-steered wheeled mobile robots at higher speeds on different terrain types. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3734–3739, 2017.

[16] Goran Huskić, Sebastian Buck, Matthieu Herrb, Simon Lacroix, and Andreas Zell. High-speed path following control of skid-steered vehicles. *The International Journal of Robotics Research*, 38(9):1124–1148, 2019.

[17] Erkan Kayacan, Herman Ramon, and Wouter Saeys. Robust trajectory tracking error model-based predictive control for unmanned ground vehicles. *IEEE/ASME Transactions on Mechatronics*, 21(2):806–814, 2016.

[18] Tran Quoc Khai and Young-Jae Ryoo. Design of adaptive kinematic controller using radial basis function neural network for trajectory tracking control of Differential-Drive mobile robot. *The International Journal of Fuzzy Logic and Intelligent Systems*, 19(4):349–359, December 2019.

[19] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[20] Gregor Klancar, Andrej Zdesar, Saso Blazic, and Igor Skrjanc. *Wheeled Mobile Robotics: From Fundamentals Towards Autonomous Systems*. Butterworth-Heinemann, USA, 1st edition, 2017.

[21] Krzysztof Kozłowski and Dariusz Pazderski. Modeling and control of a 4-wheel skid-steering mobile robot. *International journal of applied mathematics and computer science*, 14(4):477–496, 2004.

[22] Denise Lam, Chris Manzie, and Malcolm Good. Model predictive contouring control. In *49th IEEE Conference on Decision and Control*, pages 6137–6142, 2010.

[23] Juncheng Li, Maopeng Ran, Han Wang, and Lihua Xie. MPC-based unified trajectory planning and tracking control approach for automated guided vehicles. In *2019 IEEE 15th International Conference on Control and Automation (ICCA)*, pages 374–380, 2019.

[24] Alexander Liniger, Alexander Domahidi, and Manfred Morari. Optimization-based autonomous racing of 1:43 scale RC cars. *Optimal Control Applications and Methods*, 36:628–647, 2015.

[25] A. Mandow, J. L. Martinez, J. Morales, J. L. Blanco, A. Garcia-Cerezo, and J. Gonzalez. Experimental kinematics for wheeled skid-steer mobile robots. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1222–1227, 2007.

[26] Mohit Mehndiratta, Erkan Kayacan, Siddharth Patel, Erdal Kayacan, and Girish Chowdhary. *Learning-Based Fast Nonlinear Model Predictive Control for Custom-Made 3D Printed Ground and Aerial Robots*, pages 581–605. Springer International Publishing, 2019.

[27] Lorenzo Nardi and Cyrill Stachniss. Actively improving robot navigation on different terrains using gaussian process mixture models. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 4104–4110, 2019.

[28] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[29] Rien Quirynen, Karl Berntorp, Karthik Kambam, and Stefano Di Cairano. Integrated obstacle detection and avoidance in motion planning and predictive control of autonomous vehicles. In *2020 American Control Conference*, pages 1203–1208, 2020.

[30] J.B. Rawlings, D.Q. Mayne, and M.M. Diehl. *Model Predictive Control: Theory, Computation, and Design.* Nob Hill Publishing, 2nd edition, 2020.

[31] Tim Salzmann, Elia Kaufmann, Jon Arrizabalaga, Marco Pavone, Davide Scaramuzza, and Markus Ryll. Real-time neural mpc: Deep learning model predictive control for quadrotors and agile robotic platforms. *IEEE Robotics and Automation Letters*, pages 1–8, 2023.

[32] Wilko Schwarting, Javier Alonso-Mora, Liam Paull, Sertac Karaman, and Daniela Rus. Safe nonlinear trajectory generation for parallel autonomy with a dynamic vehicle model. *IEEE Transactions on Intelligent Transportation Systems*, 19(9):2994–3007, 2018.

[33] Max Schwenzer, Muzaffer Ay, Thomas Bergs, and Dirk Abel. Review on model predictive control: an engineering perspective. *The International Journal of Advanced Manufacturing Technology*, 117(5):1327–1349, Nov 2021.

[34] D. Soetanto, L. Lapierre, and A. Pascoal. Adaptive, non-singular path-following control of dynamic wheeled robots. In *42nd IEEE International Conference on Decision and Control*, volume 2, pages 1765–1770 Vol.2, 2003.

[35] Nathan A. Spielberg, Matthew Brown, and J. Christian Gerdes. Neural network model predictive motion control applied to automated driving with unknown friction. *IEEE Transactions on Control Systems Technology*, 30(5):1934–1945, 2022.

[36] Guillem Torrente, Elia Kaufmann, Philipp Föhn, and Davide Scaramuzza. Data-driven mpc for quadrotors. *IEEE Robotics and Automation Letters*, 6(2):3769–3776, 2021.

[37] Robin Verschueren, Gianluca Frison, Dimitris Kouzoupis, Jonathan Frey, Niels van Duijkeren, Andrea Zanelli, Branimir Novoselnik, Thivaharan Albin, Rien Quirynen, and Moritz Diehl. acados: a modular open-source framework for fast embedded optimal control, 2020.