Correcting Nominal Models with Learned Residuals for Aggressive Real-Time Control

By

Si Heng Teng
CMU-RI-TR-25-80

School of Computer Science
The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania

Submitted in partial fulfillment of the requirements for
the Degree of Master of Science

Abstract

Car-like robots are mechanically simple and dynamically stable, making them well-suited for applications such as inspections, surveillance, and search-and-rescue operations. Given a model of the robot's dynamics, model predictive control (MPC) plans optimal actions with specified constraints. The performance of MPC is limited by inaccuracies in the underlying system model - high fidelity models are complex and brittle, while simple models trade accuracy for computational tractability.

This work presents a hybrid control framework that corrects predictions from a low-dimensional kinematic model through a data-driven residual network, termed the Corrective Residual Network (CRN). CRN significantly reduces model errors compared to the nominal model. CRN is integrated with MPC in a proposed algorithm called Corrective Sequential Linear-Quadratic MPC (CSLQ-MPC). CSLQ-MPC demonstrates strong performance and robustness towards model mismatch, including in scenarios with incorrect model parameters given to the nominal model. CSLQ-MPC sees an improvement in cross-track error of 54.3% and an improvement in time taken of 14.3%.

Acknowledgments

To my wife, Sarah, and my son, Cassius, who have inspired me to be better, do better, and remind me daily what matters most.

List of Tables

List of Figures

Table of Contents

Chapter 1

Introduction

Mobile robots have been deployed across multiple domains such as search-and-rescue, infrastructure inspections, surveillance, logistics, and security. [3, 4]. Car-like robots are a popular choice of mobile robots capable of high efficiency and stability on flat, firm ground. [5] Their long history in robotics has led to much work in modeling the dynamics of such systems [6]. As opposed to their legged or aerial counterparts, they benefit from mechanical simplicity, stable equilibria and higher energy efficiency while requiring less degrees of freedom than other forms of locomotion. Due to these factors, car-like robots are well-suited for deployment in man-made, urban environments.

The control of car-like robots has been studied extensively over the years [6]. Model Predictive Control (MPC) remains a powerful tool deployed across all manners of robots [7–10]. MPC optimizes for a sequence of control actions over a finite time horizon, using a model of the system's dynamics to predict state propagation. As MPC requires a model of the system dynamics, the efficacy of the algorithm also depends heavily on having an accurate model of the system. Inaccurate models can lead to unstable or suboptimal operation of the robot. Small mismatches in the model or in identifying model parameters can degrade performance further.

There are several modeling approaches commonly used for car-like robots. Classical physics-based models range in complexity from simple kinematic representations to detailed models incorporating suspension dynamics and terrain interactions[11]. While high-fidelity models are accurate within well-defined operating points, they are computationally expensive and can be brittle, limiting real-time applicability. Such models are often limited in continuous differentiability, creating additional challenges when used alongside optimization based controls. Additionally, these models require identification of a large number of parameters, many of which are limited to specific vehicles and terrains.

Simplified models, such as the Kinematic Bicycle Model explored within this thesis, are lightweight and widely used [8, 12–15]. Such models are often computationally efficient, differentiable, numerically stable, and capable of capturing the bulk of the car-like robot's nonholonomic constraints. However these models often make assumptions and simplifications that can introduce significant approximation error.

Data-driven models provide an alternative approach by learning system behavior directly from data [16, 16–22]. These models are powerful and allow users to incorporate a variety of information from sensor information to semantic information, and can identify temporal patterns from environments.

End-to-end learning approaches, where sensor inputs are mapped directly to control actions, have demonstrated impressive performance in recent years [20, 23–25]. Given a large enough dataset, data-driven models have better asymptotic performance and versatility [26]. However, end-to-end models often suffer from a lack of explainability, as well as brittleness under distributional shift, making them hard difficult to trust in real-world applications.

With these considerations, this thesis investigates how learning-based residual models can augment simple nominal vehicle dynamics models enabling accurate, data-efficient, robust model predictive control for car-like robots. This thesis proposes a three-stage approach:

First, the thesis begins by formalizing Sequential Linear-Quadratic Programming MPC (SLQP-MPC), a commonly used, real-time iteration subset of Sequential Quadratic Programming-MPC. SLQP-MPC is capable of using a nonlinear dynamics model through iterative linearization, modeling the finite-horizon control problem as a quadratic program subproblem and solving it iteratively to convergence or for a certain number of iterations.

To overcome the limitations of simple nominal models, a Corrective Residual Network (CRN) is trained to predict the difference between the true system dynamics and nominal model predictions. The CRN uses state and action history as inputs to predict a correction to the nominal model's prediction of the state derivative.

Finally, the thesis proposes Corrective SLQ-MPC (CSLQ-MPC), where corrections from the CRN is integrated into SLQ-MPC for a computationally efficient and powerful control method capable of tracking difficult paths. CSLQ-MPC leads to an improvement in cross-track error of 54.3% and an improvement in time taken of 14.3%.

Experiments and verifications are conducted within simulation of a custom car-like robot capable of front and rear steering in simulation.

Chapter 2

Background

Car-like robots are nonholonomic. Most car-like robots equipped with standard wheels have their center-of-mass velocities constrained to specific directions according to their wheel directions. A car-like robot, unless equipped with specific wheels, generally will not be capable of moving sideways. [5, 27, 28]. Many models for nonholonomic car-like robots exist. Kinematic models are a family of simple and computationally efficient models that are typically differentiable and numerically stable [8, 12, 27]. These models provide insight into the behavior of the robot under idealized conditions, with minimal system parameters to be identified.

While computationally efficient and stable, kinematic models for car-like robots are often nonlinear. Nonlinear functions have numerous solutions, and are computationally expensive to solve [7, 22, 29]. Linearization allows for simplification of nonlinear models around specific operating points into a linear model, greatly increasing computational efficiency and stability of solutions.

Model Predictive Control is a widely used control framework deployed on varying robots from small robots to spacecraft that provide closed-loop feedback control through a model of the robot's dynamics. This thesis uses MPC alongside a linearized kinematic model to solve for optimal control actions for the robot.

Data-driven methods have proven to be effective for learning the dynamics of robot systems. Long Short-Term Memory (LSTM) are a tested architecture of reasoning about time-series data [2, 30, 31]. Given a window of past inputs, LSTMs can learn to discern patterns in temporal data and make accurate predictions. In this thesis, LSTMs are explored in the context of a corrective residual network to correct inaccuracies in simple kinematic models.

This chapter briefly covers necessary background as they relate to the proposed control architecture of a double-steer car-like robot. Topics within this chapter include kinematic modeling of the custom double-steer car-like robot, state space models, linearization, model-predictive control and long-short term memory in order.

2.1 Dynamical Systems

A dynamical system is one whose behavior evolves over time [28, 32–34]. In robotics, dynamical systems often evolve in response to control inputs. When an action is applied to a dynamical system, the effect of the action does not occur instantaneously. Applying throttle to a car-like robot does not immediately change the velocity. Instead, it induces an acceleration that gradually increases the robot's velocity over time. Understanding the dynamics of a system provides insight into how the system's state will evolve given an input.

A model is a mathematical representation of the true dynamics of a system. Models vary in fidelity, from simple approximations that rely on assumptions and geometry, to high-fidelity, computationally expensive formulations that capture physical effects more accurately. Regardless of their complexity, models predict future states given the current state and control input.

The state of a system is a set of variables that fully characterizes the dynamics of a system at a given time [32]. Given the current state and input, one can predict the system's future behavior without requiring any additional past information.

In robotics and other engineering systems, models often incorporate actuator and sensor measurements. This leads to the development of the state-space model formulation, which generalizes the relationship between states, inputs and observations. The standard, non-linear state-space form is given by the differential equation:

$$\dot{x} = f(x, u) \tag{2.1}$$

$$y = h(x, u) \tag{2.2}$$

Here:

- $x \in \mathbb{R}^n$ is a vector of state variables
- $u \in \mathbb{R}^m$ is a vector of control inputs
- $y \in \mathbb{R}^q$ is a vector of measurements
- $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is the dynamics function, a smooth map from state and action to the state derivative with respect to time.
- $h : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^q$ is a smooth map from state and action to an observation obtained by the sensors.

To focus on the dynamics function, f, measurements of the states are assumed to be perfect within this thesis.

This variant of the state space model is time-invariant as neither f nor h depend on time explicitly. If the functions f and h are also linear, linear, the system is said to be Linear Time-Invariant (LTI). The LTI form of the state-space model is given by:

$$\dot{x} = Ax + Bu \tag{2.3}$$

where $A \in \mathbb{R}^{n \times n}$ is called the dynamics matrix. The matrix $B \in \mathbb{R}^{m \times n}$ is called the control matrix.

Note that while the dynamics of the system is written as a first-order differential equation, given proper definition of the state and smooth map f, the dynamics of higher order differential equations may be captured as well.

The geometric interpretation of the dynamics of a robot is to view the state space as pieces of $\mathbb{R}^n$ for some n. Within each point x in the state space, there exists a collection of tangent vectors called a tangent space $T_x x$. The tangent space consists of all possible instantaneous velocities (tangent vectors) at that point. The dynamics function $f(x)$ defines a vector field over the state space by assigning to each point x a vector $f(x)$ in the corresponding tangent space. This concept is illustrated in Figure 2.1.

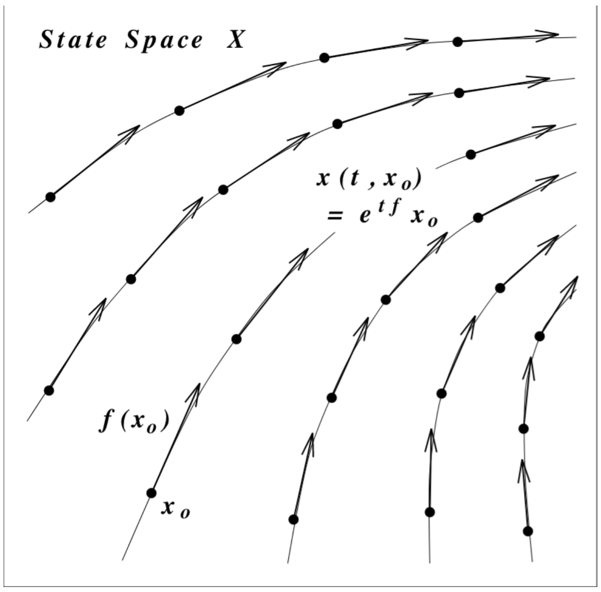


Figure 2.1: Geometric visualization of the dynamics function f in state space [1].

2.2 Double-Steer Kinematic Bicycle Model (DSKBM)

Kinematic models provide a basic study of how robotic systems behave [5, 8, 35]. The Kinematic Bicycle Model (KBM) is a simple, differentiable, numerically stable yet sufficient representation of a car with accuracy that rivals that of the dynamic bicycle model [8].

The KBM is commonly modeled in the Lie group $\mathbb{SE}(2)$ with three degrees of freedom, where the car-like robot is assumed to be constrained to a plane [18]. This is in contrast to simple vehicle models, which can have more than 10 degrees of freedom. Despite its simplicity, the 3-DoF kinematic model effectively captures the nonholonomic constraints arising from the robot's geometry and steering capabilities. [36] Kong et. al [8] compares predictions from a KBM to a dynamic bicycle model with linear tire models, and concluded that, given discretization choices, the KBM can be favorable.

Variants of the KBM has also been very commonly used in conjunction with MPC. Qian et. al uses KBM in conjunction with MPC for formation convoy control.

Cardoso et. al uses a similar setup for the Intelligent Autonomous Robotic Automobile, a fully autonomous car capable of driving at speeds up to 9 m/s.

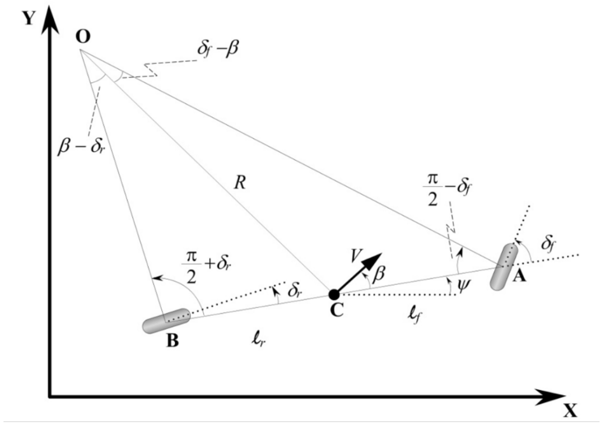


Figure 2.2: Double-Steer Kinematic Bicycle Model [28]

The model groups the front and rear pairs of wheels into two single effective wheels located at the center of their respective axles.

In this section, we derive expressions for the time derivative of the state vector, denoted $\dot{X}$, of a car-like robot with independently actuated front and rear steering.

The KBM model makes a few major assumptions that typically limit its use to low-speed applications.

1. A four wheeled car behaves the same as a two wheeled bicycle.
2. There is no slip - velocity at each wheel are in the same direction as their steering angles.
3. There is no skip.
4. The rate of change of the radius of the vehicle's path is small, therefore $\dot{\psi} = \frac{v}{R}$.

Figure 2.2 provides an overview of the Double-Steer Kinematic Bicycle Model (DSKBM). Table 2.1 summarizes the different symbols found within Figure 2.2.

Polack et. al studied the consistency of using a kinematic bicycle model [36], as compared to a complex, realistic dynamics model of a vehicle. In their study, they stress the importance of limiting lateral acceleration of the vehicle, as model errors can become large when lateral acceleration constraints are not respected, concluding that more accurate vehicle models should be used when dealing with higher accelerations.

Table 2.1: Symbols used in the Double-Steer Kinematic Bicycle Model

Symbol	Description
β	Sideslip angle of vehicle's center of mass (COM) (in radians)
δ_f	Effective front steering angle (in radians)
δ_r	Effective rear steering angle (in radians)
ℓ_f	Distance from vehicle's COM to front axle (in m)
ℓ_r	Distance from vehicle's COM to rear axle (in m)
v	Velocity vector of the vehicle's COM (in m/s)
O	Point corresponding to vehicle's instantaneous center of rotation
A	Point corresponding to front wheel-base
B	Point corresponding to rear wheel-base
C	COM of vehicle chassis

2.2.1 State and Control Definitions

We define the continuous state vector $X(t)$ as:

$$X(t) = \begin{bmatrix} x(t) \\ y(t) \\ v(t) \\ \psi(t) \end{bmatrix} \tag{2.4}$$

where $x(t)$ and $y(t)$ define the position of the vehicle's COM in the inertial frame, $v(t)$ is the magnitude of the forward velocity in the COM, and $\psi(t)$ is the vehicle's heading angle, defined as the angle between the vehicle's longitudinal axis and the inertial x-axis.

The control input vector, $U(t)$ is defined as:

$$U(t) = \begin{bmatrix} a(t) \\ \delta_f(t) \\ \delta_r(t) \end{bmatrix} \tag{2.5}$$

where $a(t)$ is the magnitude of the longitudinal acceleration applied at the COM, $\delta_f(t)$ is the steering angle applied to the front wheels, and $\delta_r(t)$ is the steering angle applied to the rear wheels. Steering angles are defined with respect to the center line of the vehicle.

2.2.2 Derivation of the DSKBM

Obtaining an expression for the time derivative of the first two components of the state x, y is straightforward using trigonometry. The derivation within this section is heavily referenced from [28].

Assuming the vehicle moves with a velocity in the global frame, v, along a heading angle ψ, the rate of change of its position in the global frame is given by:

$$\dot{x} = v\cos(\psi + \beta) \tag{2.6}$$

$$\dot{y} = v\sin(\psi + \beta) \tag{2.7}$$

An expression for the angular velocity, $\dot{\psi}$ is derived using the geometry of the vehicle using the front and rear steering triangles formed by the triangles OCA and OCB in Figure 2.2. With the no-slip assumption, the velocity vectors at both points A and B are aligned with the orientation of the wheels.

$$\frac{\sin(\delta_f - \beta)}{l_f} = \frac{\sin(\frac{\pi}{2} - \delta_f)}{R} \tag{2.8}$$

$$\frac{\sin(\delta_f)\cos(\beta) - \sin(\beta)\cos(\delta_f)}{l_f} = \frac{\cos(\delta_f)}{R} \tag{2.9}$$

$$\frac{l_f}{\cos(\delta_f)}\frac{\sin(\delta_f)\cos(\beta) - \sin(\beta)\cos(\delta_f)}{l_f} = \frac{l_f}{\cos(\delta_f)}\frac{\cos(\delta_f)}{R} \tag{2.10}$$

$$\tan(\delta_f)\cos(\beta) - \sin(\beta) = \frac{l_f}{R} \tag{2.11}$$

$$\frac{\sin(\beta - \delta_r)}{l_r} = \frac{\sin(\frac{\pi}{2} + \delta_r)}{R} \tag{2.12}$$

$$\frac{\cos(\delta_r)\sin(\beta) - \cos(\beta)\sin(\delta_r)}{l_r} = \frac{\cos(\delta_r)}{R} \tag{2.13}$$

$$\frac{l_r}{\cos(\delta_r)}\frac{\cos(\delta_r)\sin(\beta) - \cos(\beta)\sin(\delta_r)}{l_r} = \frac{\cos(\delta_r)}{R}\frac{l_r}{\cos(\delta_r)} \tag{2.14}$$

$$\sin(\beta) - \tan(\delta_r)\cos(\beta) = \frac{l_r}{R} \tag{2.15}$$

An expression of the slip angle can be obtained through equations 2.11 and 2.15. Multiply equation 2.11 by l_r and subtract it by the equation 2.15 multiplied by l_f.

$$\frac{l_f l_r}{R} - \frac{l_r l_f}{R} = l_f(\sin(\beta) - \tan(\delta_r)\cos(\beta)) - l_r(\tan(\delta_f)\cos(\beta) - \sin(\beta)) \tag{2.16}$$

$$(l_f + l_r)\sin(\beta) = \cos(\beta)(l_f \tan(\delta_r) + l_r \tan(\delta_f)) \tag{2.17}$$

$$\tan(\beta) = \left(\frac{l_f \tan(\delta_r) + l_r \tan(\delta_f)}{l_f + l_r}\right) \tag{2.18}$$

$$\beta = \arctan\left(\frac{l_f \tan(\delta_r) + l_r \tan(\delta_f)}{l_f + l_r}\right) \tag{2.19}$$

Given that the turning radius R is approximately constant over the interval of interest, and that the vehicle satisfies the no-slip assumption, the yaw rate $\dot{\psi}$ may be expressed in terms of the vehicle speed and steering angles. Using equations 2.11 and 2.15, and the relationship $\dot{\psi} = \frac{V}{R}$, we obtain:

$$\dot{\psi} = \frac{v}{R} \tag{2.20}$$

$$= \frac{v\cos(\beta)}{l_f + l_r}\left(\tan(\delta_f) - \tan(\delta_r)\right) \tag{2.21}$$

$$\tag{2.22}$$

The system dynamics can be defined as:

$$\dot{X} = f(X, U) \tag{2.23}$$

$$f(X, U) = \begin{bmatrix} v\cos(\psi + \beta) \\ v\sin(\psi + \beta) \\ a \\ \frac{v\cos(\beta)}{l_f + l_r}\left(\tan(\delta_f) - \tan(\delta_r)\right) \end{bmatrix} \tag{2.24}$$

where the slip angle β is given by equation 2.19.

2.2.3 Kinematic Models vs. Dynamic Models

Another common, simplified model for car-like robots is the lateral dynamic bicycle model [35]. Unlike the kinematic model, which assumes no tire slip, the dynamic model incorporates lateral tire forces and vehicle inertial effects. A basic form of dynamic model, the dynamic bicycle model, assumes only lateral tire forces that are approximated through a tire slip model. A common simplification is the linear tire model, which relates lateral force to slip angle through a linear relationship.

To illustrate the effect of lateral tire forces, picture a car-like robot undergoing a steady turn along a curve of radius R (straight line motion in this case corresponds to $R = \infty$). Assuming the turn follows a perfect circular arc, the lateral acceleration experienced by the robot is given by $a = \frac{v^2}{R}$ [27].

Through Newton's second law, this implies that the lateral force, $F_{lateral} \propto v^2$, demonstrating a quadratic relationship between lateral force and speed. At low speeds, the resulting lateral forces are small, which makes the added complexity of a dynamic model less justifiable.

This thesis chooses a kinematic model over a dynamic model due to several reasons. A kinematic model:

- Is more computationally efficient, giving more computational bandwidth for trained corrective models.
- Has fewer parameters to identify. In the case of the DSKBM, only the distances from the center of mass to the front and rear axle, (l_f, l_r) need to be identified.
- Numerically stable, including at tasks involving stop-and-go maneuvers. [8, 35]

In contrast, the dynamic bicycle model, even using the linear tire model, requires identification of additional parameters including the front and rear cornering stiffness coefficients (C_{α_f}, C_{α_r}), the vehicle's yaw moment of inertia (I_z) in addition to the distances from center of mass to front and rear axle (l_f, l_r) [8].

Furthermore, prior studies have shown that dynamic models are unstable at low speeds, and require special treatment to avoid numerical instability [35].

2.3 Linearization and Discretization

The dynamics model we derived in the previous chapter was both continuous and non-linear. However, using nonlinear models within a predictive controller introduces significant computational challenges. This section discusses the background necessary to linearize and discretize a model, resulting in a time-invariant linear approximation that is both computationally tractable and suitable for use with model predictive control.

2.3.1 Linearization

Non-linear optimization problems are typically non-convex, resulting in multiple local minima and no guarantees of uniqueness or global optimality. Non-linear problems are expensive to solve in real-time and typically require a good initial guess to improve convergence and stability [37].

Nonlinear MPC requires globally accurate models of the underlying dynamics of a system. Inaccuracies can lead to poorer performance or instability in solutions. While these may be handled with robust formulations of MPC, they typically increase computational cost further. Furthermore, due to the computational burden of using non-linear solvers, typically only a limited number of iterations are performed [7].

To address these challenges, the non-linear dynamics derived in the previous section can be linearized around a current operating point, denoted as $(\bar{x}_k, \bar{u}_k)$.

In a small neighborhood around the operating point, the non-linear function appears to be linear.

A tangent may be used to approximate a nonlinear function within this neighborhood. The nonlinear function $f(x_k, u_k)$ is approximated using the first-order Taylor expansion:

$$f(x_k, u_k) \approx f(\bar{x}_k, \bar{u}_k) + \frac{\partial f}{\partial x}|_{x=\bar{x}}(x - \bar{x}_k) + \frac{\partial f}{\partial u}|_{u=\bar{u}}(u - \bar{u}_k) \tag{2.25}$$

The Jacobians in the Taylor Series expansion are denoted A, B, and an affine offset

C is introduced.

$$A = \frac{\partial f}{\partial x}|_{x=\bar{x}_k, u=\bar{u}_k} \tag{2.26}$$

$$B = \frac{\partial f}{\partial u}|_{x=\bar{x}_k, u=\bar{u}_k} \tag{2.27}$$

$$C = f(\bar{x}_k, \bar{u}_k) - A\bar{x}_k - B\bar{u}_k \tag{2.28}$$

The term C ensures that the linearized model matches the nonlinear dynamics at the point $(\bar{x}_k, \bar{u}_k)$. Without it the approximation would treat $f(\bar{x}_k, \bar{u}_k) = 0$, which is untrue unless linearizing about the origin.

Linearization leads to the system below, an affine approximation valid in a small neighborhood around $(\bar{x}_k, \bar{u}_k)$.

$$\dot{x}_k = f(x_k, u_k) \tag{2.29}$$

$$= Ax_k + Bu_k + C \tag{2.30}$$

The neighborhood in which the linearization is accurate about depends highly on the nonlinear function itself. Linearization allows transcription of the control problem into a convex quadratic program when used in MPC, allowing for efficient, globally optimal solutions.

2.3.2 Discretization

Model Predictive Control (MPC) often operates in discrete time. Discretization involves approximating the dynamics in continuous time with a discrete time model with a fixed timestep, h. Discretization assumes that over a sufficiently small timestep, the dynamics of the function remains approximately constant.

A common way to discretize a continuous function is by using numerical integrators. Integrators propagate the solution of an initial value problem forward by h, assuming a zero-order hold in the controls over h. The solution is evaluated using an expression for the time derivative evaluated before taking the time step h.

Runge-Kutta methods provide a good trade-off between accuracy and computational cost [38]. The fourth-order Runge-Kutta method (RK4) is commonly used, providing a higher-order approximation and more accurate estimate than the commonly used Forward

Euler method. RK4 computes intermediate slopes to approximate the solution over each time-step:

$$k_1 = f(x_k, u_k) \tag{2.31}$$

$$k_2 = f(x_k + \frac{h}{2}k_1, u_k) \tag{2.32}$$

$$k_3 = f(x_k + \frac{h}{2}k_2, u_k) \tag{2.33}$$

$$k_4 = f(x_k + \frac{h}{k_3}, u_k) \tag{2.34}$$

$$x_{k+1} = x_k + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{2.35}$$

For a linear system, RK4 produces a discretized model that closely approximates a Taylor expansion e^{Ah} of the matrix exponential. With the linearized dynamics integrated with RK4, assuming a zero-order hold on the controls over h:

$$k_1 = Ax_k + Bu_k + C \tag{2.36}$$

$$k_2 = A\left(x_k + \frac{h}{2}k_1\right) + Bu_k + C \tag{2.37}$$

$$k_3 = A\left(x_k + \frac{h}{2}k_2\right) + Bu_k + C \tag{2.38}$$

$$k_4 = A\left(x_k + hk_3\right) + Bu_k + C \tag{2.39}$$

$$x_{k+1} = x_k + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{2.40}$$

$$\tag{2.41}$$

Expanding these expressions and collecting terms in front of x_k and u_k gives the following linear system:

$$x_{k+1} = A_d x_k + B_d u_k + C_d \tag{2.42}$$

$$A_d = I + hA + \frac{h^2}{2}A^2 + \frac{h^3}{6}A^3 + \frac{h^4}{24}A^4 \tag{2.43}$$

$$B_d = h\left(I + A + \frac{h^2}{2}A^2 + \frac{h^3}{6}A^3\right)B \tag{2.44}$$

$$C_d = h\left(I + hA + \frac{h^2}{2}A^2 + \frac{h^3}{6}A^3\right)C \tag{2.45}$$

2.4 MPC

Model Predictive Control (MPC) is a robust, modern control technique capable of working with nonlinear dynamics, input and state constraints [7, 39, 40]. MPC achieves this by solving an optimization problem at each discretized time step, determining the best sequence of control actions over a prediction horizon. The optimization relies on a model of the system to predict future states, and these predictions are enforced as equality constraints within the problem. In practice, only the first control input from the optimized sequence is applied, and the optimization is repeated at the next step in a receding horizon fashion.

2.4.1 Linear Quadratic Regulator

MPC is closely related to the Linear Quadratic Regulator (LQR). In the absence of constraints, and when both the prediction and control horizon approach infinity, MPC converges to the LQR solution and can be solved using the same methods [41]. LQR seeks an optimal state-feedback control law that drives the system to a desired equilibrium while minimizing a quadratic cost function. LQR's solutions can be derived through applying Bellman's Principle of Optimality and Ricatti Recursion.

LQR is adaptable and could potentially work with non-linear dynamics through linearization about an operating point, provided the system is sufficiently close to the operating point. However, it fails when constraints are introduced. Naively enforcing constraints (clipping torque commands for example) can lead to instability or suboptimal behavior.

Through adding a horizon and repeatedly solving a constrained quadratic program over a finite horizon, MPC is able to plan control actions that respect system limitations.

This thesis focuses on convex MPC formulations, where cost function, constraints, and linearized dynamics are all convex, allowing the use of efficient and reliable optimization solvers.

2.4.2 Convex MPC

Convex MPC is typically formulated in the state-space domain using discrete-time linear dynamics, with a specified initial condition corresponding to the robot's current state:

$$x_{k+1} = A_d x_k + B_d u_k + C_d \tag{2.46}$$

$$x(0) = x_{\text{init}} \tag{2.47}$$

At each time step, MPC solves an open-loop finite-horizon optimization problem to compute the optimal control sequence. A basic variant of MPC uses a cost defined by:

$$\begin{aligned} J = \sum_{k=1}^{H-1} \Big((x_k - x_{\text{ref},k})^T Q (x_k - x_{\text{ref},k}) + \frac{1}{2}(u_k - u_{\text{ref},k})^T R (u_k - u_{\text{ref},k}) \Big) \\ + \frac{1}{2}(x_k - x_{\text{ref},k})^T P_H (x_k - x_{\text{ref},k}) \end{aligned} \tag{2.48}$$

The cost function is quadratic and convex, provided that the cost matrices Q, R, P_H are positive semi-definite. Assuming the dynamics and constraints are also convex, the overall optimization problem remains convex and can be efficiently solved using standard quadratic programming (QP) solvers.

Convexity ensures that the optimization problem has a unique global minimum. Efficient solvers, such as QPOASES, can find this minimum efficiently and reliably. By ensuring the problem stays convex, MPC becomes both predictable and computationally tractable.

The complete optimization problem takes the following basic form:

$$J = \sum_{k=1}^{H-1} \left((x_k - x_{\mathrm{ref},k})^T Q (x_k - x_{\mathrm{ref},k}) + \frac{1}{2}(u_k - u_{\mathrm{ref},k})^T R (u_k - u_{\mathrm{ref},k}) \right)$$

$$+ \frac{1}{2}(x_H - x_{\mathrm{ref},H})^T P_H (x_H - x_{\mathrm{ref},H}) \tag{2.49}$$

$$\text{s.t.} \quad x_{k+1} = A_d x_k + B_d u_k + C_d, \quad k = 0, \dots, H-1 \tag{2.50}$$

$$x_0 = x_{\mathrm{init}} \tag{2.51}$$

$$x_k \in \mathcal{X}, \quad u_k \in \mathcal{U}, \quad \forall k \tag{2.52}$$

In this formulation, solving the QP optimizes for all control inputs u_0 through u_{H-1}, with the initial state x_0 fixed to the current state of the system. Introducing the linearized dynamics as a constraint ensures that the predicted trajectory is constrained to follow the system dynamics at each time step. For MPC to be feasible, the origin of the problem has to be within the interior of the feasible region [39].

2.4.3 MPC Formulation Used in This Work

The designed MPC formulation in this thesis builds on the convex MPC framework by incorporating additional convex cost terms and constraints that reward smoother control behavior and ensure actuator feasibility. The cost terms and constraints maintain convexity while better aligning the control commands with physical limitations of the system.

The resulting optimization problem is:

$$\min_{u_0:u_{H-1}} \quad J = \sum_{k=1}^{H-1}\left((x_k - x_{\text{ref},k})^T Q(x_k - x_{\text{ref},k}) + \frac{1}{2}(u_k - u_{\text{ref},k})^T R(u_k - u_{\text{ref},k})\right) \tag{2.53}$$

$$+ \sum_{k=2}^{H-1}(u_l - u_{l-1})^T \bar{R}(u_l - u_{l-1}) \tag{2.54}$$

$$+ \frac{1}{2}(x_H - x_{\text{ref},H})^T P_H(x_H - x_{\text{ref},H}) \tag{2.55}$$

$$\text{s.t.} \quad x_{k+1} = Ax_k + Bu_k + C_d \tag{2.56}$$

$$x_0 = x_{init} \tag{2.57}$$

$$u_{lower} \leq u_k \leq u_{upper} \tag{2.58}$$

$$\Delta u_{lower} \leq u_l - u_{l-1} \leq \Delta u_{upper}\textbf{??} \tag{2.59}$$

Introducing a cost penalizing change in action (Eqn. 2.54), as well as constraints for upper and lower limits of actions (Eqn. 2.58) as well as changes in action (Eqn. **??**) allows the MPC algorithm to choose actions that are feasible, and reason about change in action commands reasonably without huge fluctuations that the system will be unable to track. It also dissuades the algorithm from changing commands unnecessarily, leading to smoother performance. This also accounts for the fact that steering angle as a direct input is an approximation - in practice, steering angle cannot be changed instantaneously.

2.5 Learning Time-series Data

2.5.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) excel at pattern detection within data sequences. They have been commonly used to handle predictions given time-series data [30, 31].

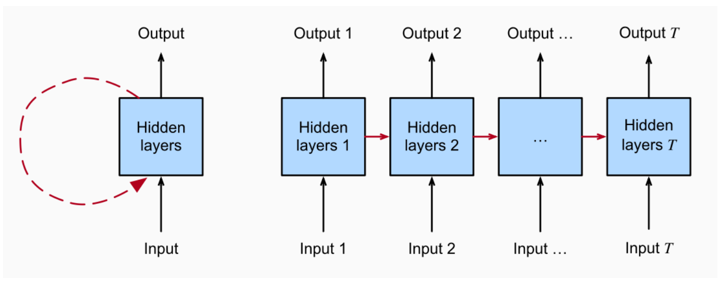


Figure 2.3: Architecture of an unrolled Recurrent Neural Network (RNN) [2]

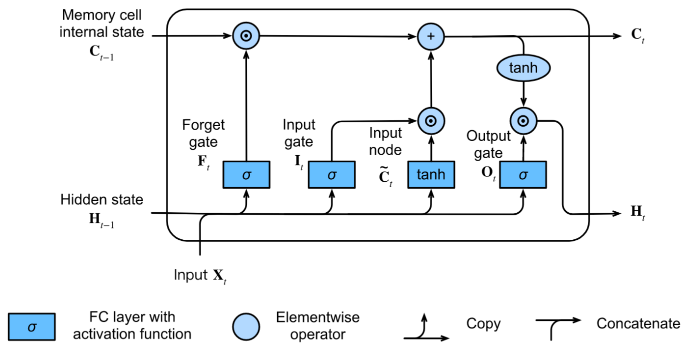

Memory cell internal state $\mathbf{C}_{t-1}$
$\mathbf{C}_t$
tanh
Forget gate $\mathbf{F}_t$
Input gate $\mathbf{I}_t$
Input node $\tilde{\mathbf{C}}_t$
Output gate $\mathbf{O}_t$
σ
tanh
Hidden state $\mathbf{H}_{t-1}$
$\mathbf{H}_t$
Input $\mathbf{X}_t$
FC layer with activation function
Elementwise operator
Copy
Concatenate

Figure 2.4: Architecture of a Long Short-Term Memory (LSTM) [2]

RNNs differ from the common Multi-Layer Perceptron (MLP) in that RNNs contain connections that allow them to reason about previous inputs. In time-series data, due to its connections RNNs are theoretically able to reason about all previous inputs $X_{0:t-1}$ as well as the current input X_t. This connection can be expressed as:

$$\mathbf{H}_t = \phi_h(\mathbf{X}_t\mathbf{W}_{xh} + \mathbf{H}_{t-1}\mathbf{W}_{hh} + \mathbf{b}_h) \tag{2.60}$$

RNNs learn through Backpropagation through time (BPTT). BPTT simply unfolds a RNN to create a large feedforward neural network in which standard backpropagation techniques may be applied. RNNs suffer from vanishing or exploding gradients due to BPTT. BPTT involves matrix multiplication over all time sequences. If there are small or large values, the gradient can easily vanish or explode as time progresses, leading to instability or forgetting.

2.5.2 Long Short-Term Memory

Long Short-Term Memory was introduced specifically to combat the vanishing or exploding gradient problem in RNNs [2, 31]. LSTMs achieve this through incorporating gates that control information flow through the network over time.

An LSTM maintains two internal states: a cell state and a hidden state. The cell state and hidden states are updated every timestep through the input gate, forget gate, and output gate, allowing the network to selectively retain or discard information from previous timesteps. Given an input sequence $\{\mathbf{X}_t\}_{t=1}^{T}$, the LSTM update equations are as follows:

$$\mathbf{F}_t = \sigma(\mathbf{W}_{xf}\mathbf{X}_t + \mathbf{W}_{hf}\mathbf{H}_{t-1} + \mathbf{b}_f) \tag{2.61}$$

$$\mathbf{I}_t = \sigma(\mathbf{W}_{xi}\mathbf{X}_t + \mathbf{W}_{hi}\mathbf{H}_{t-1} + \mathbf{b}_i) \tag{2.62}$$

$$\mathbf{O}_t = \sigma(\mathbf{W}_{xo}\mathbf{X}_t + \mathbf{W}_{ho}\mathbf{H}_{t-1} + \mathbf{b}_o) \tag{2.63}$$

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{W}_{xc}\mathbf{X}_t + \mathbf{W}_{hc}\mathbf{H}_{t-1} + \mathbf{b}_c) \tag{2.64}$$

$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t \tag{2.65}$$

$$\mathbf{H}_t = \mathbf{O}_t \odot \tanh(\mathbf{C}_t) \tag{2.66}$$

σ denotes the sigmoid activation function, and $\odot$ represents element-wise multiplication.

By circumventing the exploding or vanishing gradient problem, LSTMs can effectively capture patterns within time-series data over long horizons. In this thesis, LSTMs were chosen over simpler architecture to compensate for potential inaccuracies in the simplified choice of state in the kinematic model.

Chapter 3

Related Work

There has been increasing interest in incorporating data-driven methods with nominal models for better planning and control. A few common themes in incorporating data-driven methods with dynamics include predicting a correction to nominal dynamics models, predicting dynamics directly without incorporating model information, and utilizing nominal dynamics models during training within physics-based neural networks to encode information from nominal dynamics models.

Holzmann et. al predict the road-tire friction coefficient for use with a dynamic vehicle model [42]. Salzmann proposes using a data-driven method that takes the quadrotor's current instantaneous state (including position, orientation and velocity), control input, and an optional context vector to predict a 6-component acceleration residual vector used in the dynamics model [22]. McKinnon and Schoellig introduce probabilistic corrections to actuator dynamics with both Gaussian Processes and Bayesian Linear Regression, used with Stochastic MPC in unknown environments [21]. Datar et. al propose modelling a car-like robot in $SE(3)$ for rough-terrain navigation [19]. They train a data-driven model that takes the current 6 DoF chassis state obtained through an on-board estimator, the current action, and a local terrain patch, and outputs roll and pitch of the next timestep. The outputs are used alongside an Ackermann kinematic model in $SE(2)$.

A wide body of work directly predict state transitions and dynamics without relying on nominal dynamics models. Wang et. al ([16, 43]) proposes training a System Identification Transformer capable of condensing a window of observations into a context vector. The context vector is then fed into an adaptive dynamics model which provides a probabilistic forward model of state transitions. The output of the probabilistic forward model can be used with Risk-Aware MPPI Controller, or be used as a criteria determining validity of certain states and actions, guiding planning and controls [16, 43, 44]. Datar et. al builds upon their previous work and introduce a end-to-end model that takes an elevation map and encodes it into a latent state space, encodes state action pairs into a latent space, and feeds it into a kinodynamic model that predicts the state transitions of future states [18]. Tremblay et. al trains a model, inspired by Hafner et. al's work in latent spaces, that is capable of fusing multimodal sensor inputs into a latent space capable of predicting state transitions [20, 45]. Nagabandi et. al trains a dynamics model comprising of two hidden layer with 500 units that learns system dynamics from data, and controls using a variant of MPC that samples K random actions, rolls them forward with learned model, and keeps the action with highest reward [26]. Yu et. al learns a universal policy through pre-computing a large number of trajectories and situations a robot may encounter, then using a System-ID model to make control predictions [46].

Physics-based neural networks that are capable of predicting physically feasible state transitions have also gained in popularity in recent years. Engin and Isler represent the controller and the value function with neural networks [47], first learning state transitions, then training them using loss functions adapted from Hamilton-Jacobi-Bellman equations, ensuring that state transitions are physically feasible.

Chapter 4

Method

4.1 Simulation Environment

In order to test the proposed algorithms, a simulator capable of simulating custom robots through URDF and that captures rolling contact well is required. For this thesis, PyBullet, a Python wrapper to the popular Bullet engine, is chosen to simulate and verify experimental results.

4.1.1 PyBullet

Bullet is a widely adopted physics engine in robotics research due to its comprehensive support for forward and inverse dynamics, forward and inverse kinematics, and its dedicated collision detection framework [48]. PyBullet is a Python wrapper for Bullet that provides access to these capabilities through a Python API, making it highly convenient for prototyping and research in machine learning, widely adopted by researchers in both industry and academia [49].

This thesis explores learning the adaptive dynamics of a custom, double-steer car-like robot. PyBullet was selected as the primary simulator over other simulators like MuJoCo due to two key advantages: its accurate modeling of rolling frictional contact and its integration with robot models defined in the Universal Robot Description Format (URDF).

Among various physics engines commonly used in robotics including ODE, DART, MuJoCo, and RaiSim, PyBullet was benchmarked to have greatest accuracy in simulating rolling contact, while coming second in efficiency among the five different simulators [50].

PyBullet supports direct URDF import and automatically computes link inertial properties based on the shape, volume and mass of the collision geometry. It also supports collision detection with custom shapes on custom robots by approximating the geometry of the robot with a convex hull.

4.1.2 Robot Modeling and Control in Simulation

The robot model used in simulation was constructed through converting the CAD files of the physical platform into a URDF. This URDF was then loaded into PyBullet to define the robot's kinematic and dynamic structure.

By default, PyBullet treats the URDF as a collection of open kinematic chains. The physical robot contains closed kinematic loops which are not natively specified in URDF. Additional joint constraints were manually specified to close the loop and ensure correct relative motion between connected parts within the simulator.

Motor actuation for the robot's drivetrain is simulated using PyBullet's velocity control mode on the joints connecting the wheel to the tire. Velocity control allows users to set a desired velocity of the joint, and uses a proportional feedback control using the error between the desired velocity and actual velocity of the form:

$$\tau = K_v(v_{\text{des}} - v_{\text{actual;}}) \tag{4.1}$$

Here, the gain K_v is user-specified and can be tuned experimentally for stability and performance.

For steering, position control is applied to the front and rear wheel joints, corresponding with how a car-like robot is steered. PyBullet's position control mode combines both proportional position and velocity terms to compute the command applied to the joint:

$$\tau = K_p(p_{\text{des}} - p_{\text{actual}}) + K_v(v_{des} - v_{\text{actual}}) \tag{4.2}$$

The error is defined as $K_p(p_{\text{des}} - p_{\text{actual}}) + K_v(v_{\text{des}} - v_{\text{actual}})$.

The position controller accounts for both displacement and rate of change. As with drivetrain control, K_p and K_v are user-defined gains that require tuning for stability and performance.

4.2 Learning Dynamics

This thesis explores a residual learning approach to model the dynamics of a robot. Instead of learning the full system dynamics, the proposed method seeks to learn just a correction to a simple approximation of a model.

Let $x \in \mathbb{R}^n$ be the state vector and $u \in \mathbb{R}^m$ be the control input. The true system dynamics is unknown to us and can be expressed as:

$$\dot{x} = f_{true}(x, u) \tag{4.3}$$

Residuals are the difference between the state derivative predicted by the nominal dynamics model and the actual state derivative experienced by the robot. Formally, if f_{true} denotes the actual state derivative, and $f_{nominal}$ is the prediction from the nominal model, the residual is defined as:

$$r(x, u) = f_{true}(x, u) - f_{nominal}(x, u) \tag{4.4}$$

A learned model $\hat{r}_\theta(x, u)$ is parameterized by θ:

$$\dot{x} \approx f_{nominal}(x, u) + \hat{r}_\theta(x, u) \tag{4.5}$$

Residual learning provides several advantages. As the LSTM only has to capture any differences between the nominal and actual dynamics, learning could potentially be more sample efficient. Learning the discrepancies allows the model to leverage structure from the nominal model, and the discrepancies are typically of smaller magnitude and lower complexity [51].

Leveraging a nominal model preserves prior knowledge about the system's dynamics, allowing the LSTM to focus on modeling unmodeled effects.

Although not a focus of this thesis, the corrective model used in this work is probabilistic. In practice, this allows uncertainty-aware predictions, enabling the system to use the nominal model when learned corrections are highly uncertain. This allows for robustness during deployment, as the baseline model remains somewhat valid in regions of the state-space where corrective predictions are unreliable.

4.2.1 Model-to-Sim Residuals

To motivate the need for a residual correction model, the state derivatives predicted by the nominal model, in this case DSKBM, are compared against those obtained through simulation. The differences highlight mismatch between the nominal model and the ground truth which can lead to performance degradation or failure during trajectory tracking.

In this experiment, DSKBM is paired with Convex Quadratic MPC with a real-time iteration scheme to track a randomly generated track. During execution, the state derivatives predicted by DSKBM and the actual state derivatives experienced by the robot in the simulator are logged for comparison. The procedure for path tracking is shown in Algorithm 1 [52].

Algorithm 1 MPC-Based Path Tracker

```
Require: User-defined positive-semi-definite cost matrices $Q \succeq 0,\ Q_f \succeq 0,\ R \succ 0,\ R_\Delta \succeq 0$
Require: Initial state $x_0$, reference speed $v_{\text{ref}}$, sampling time $dt$, MPC horizon $T$, simulation length $T_{\text{sim}}$, track
1: $x \leftarrow x_0$
2: for $t = 0$ to $T_{\text{sim}} - 1$ do
3:     $x_{\text{ref}} \leftarrow$ GET_REFERENCE_TRAJECTORY$(x,\ \mathit{track},\ v_{\text{ref}},\ T,\ dt)$
4:     $u_k^\star,\ x_k^\star \leftarrow$ MPC.PREDICT$\big(x_k,\ x_{\text{ref}},\ Q, Q_f, R, R_\Delta\big)$
5:     $u_k \leftarrow u_k^\star[0]$
6:     send $u$ to low-level controller
7:     $x_{k+1} \leftarrow$ SIM.CONTROLLOOPSTEP$(x_k, u_k, dt)$
8: end for
```

PyBullet offers an API call to obtain the instantaneous linear and angular velocity experienced by the robot. Acceleration experienced by the robot is calculated through finite differences of the velocity.

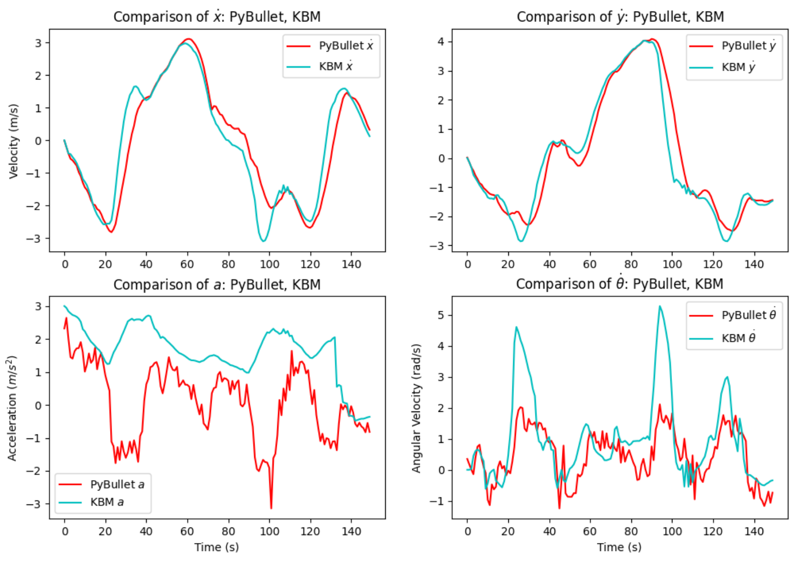


Figure 4.1: $\dot{x}$ computed by KBM vs. ground truth obtained from PyBullet

Figure 4.1 illustrates the discrepancy between the predicted and true state derivatives.

Despite a small mismatch, the nominal model handles predictions of $\dot{x}$ and $\dot{y}$ on flat terrain well. That is expected primary because DSKBM predicts $\dot{x}$ and $\dot{y}$ directly through using the robot's current velocity and its current yaw ψ.

DSKBM struggles with predicting angular velocity and acceleration. This discrepancy arises since kinematic models do not reason about forces and mass. In this model, COM acceleration is a control input, and any applied acceleration is directly translated into an increase in velocity. In contrast, the true system dynamics are influenced by friction, slip among other factors, and not all commanded acceleration results in a proportional velocity change.

Similarly, angular velocity is predicted in the DSKBM through the robot's current velocity and COM sideslip, computed through the robot's front and rear steering angles and the wheelbase through the robot's geometry, without accounting for the dynamic response of the vehicle. In reality, angular velocity evolves over time due to inertial effects and cannot change instantaneously as predicted by the kinematic model.

These model mismatches, especially in the acceleration and angular velocity of the

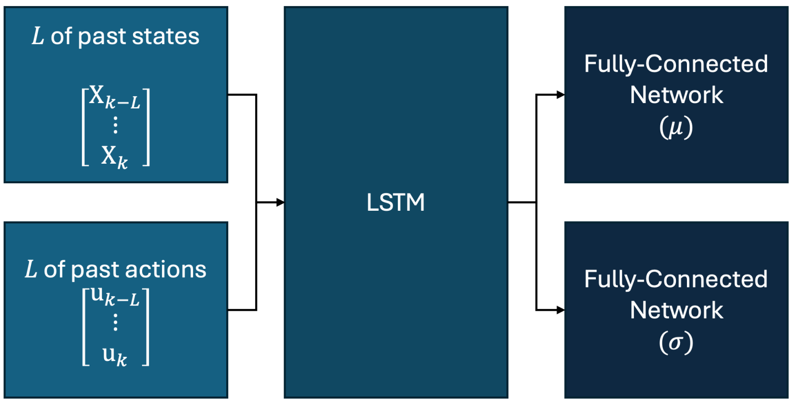

L of past states
$\begin{bmatrix} x_{k-L} \\ \vdots \\ x_k \end{bmatrix}$
L of past actions
$\begin{bmatrix} u_{k-L} \\ \vdots \\ u_k \end{bmatrix}$
LSTM
Fully-Connected Network (μ)
Fully-Connected Network (σ)

Figure 4.2: Architecture of Probabilistic CRN

vehicle, can accumulate over time, leading to drift or suboptimal behavior when used in conjunction with MPC. This motivates the corrective residual network introduced within this thesis to correct nominal predictions.

4.2.2 Corrective Residual Network (CRN)

To learn a correction to the nominal model, the method uses LSTM networks, an architecture that excels at reasoning over time-series data. In the rest of the thesis, the model is referred to as the Corrective Residual Network (CRN).

The choice of state used with the DSKBM is intentionally simplistic for computational purposes, and does not encapsulate all physical variables needed for accurate prediction. Consequently, instantaneous corrective predictions based on the simplistic state is insufficient. By incorporating a temporal window of past states and actions, the CRN can capture dynamic trends over time not explicitly modeled by the nominal system.

The input to the CRN consist of a sequence of robot states and actions over a fixed horizon. In this thesis, a sequence length of 8 is chosen, providing the CRN a window of 8 past actions and states, including the current state and action. The CRN makes a corrective prediction to the current state derivative using this window.

The output from the LSTM is fed into two fully connected (FC) units, both with 1 hidden layer and 256 units. One branch outputs the mean of the predicted residual, and the other outputs the covariance, forming a probabilistic residual model.

To ensure the predicted covariance matrix is well-defined and positive semi-definite,

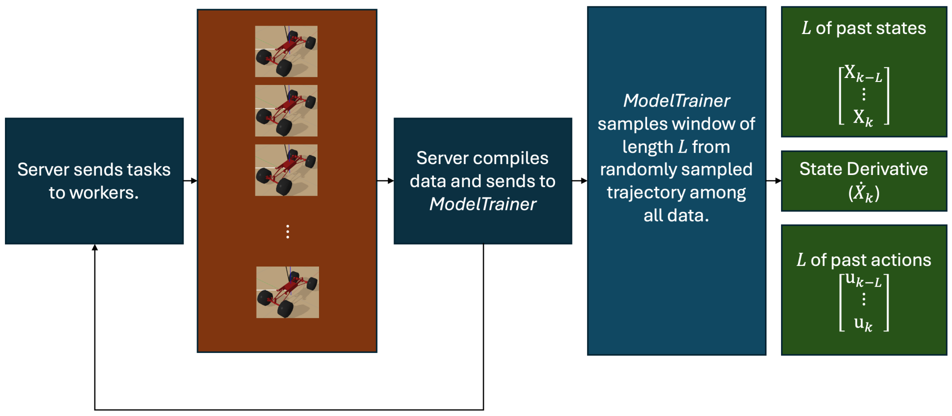

Server sends tasks to workers.
Server compiles data and sends to *ModelTrainer*
ModelTrainer samples window of length L from randomly sampled trajectory among all data.
L of past states
$\begin{bmatrix} x_{k-L} \\ \vdots \\ x_k \end{bmatrix}$
State Derivative ($\dot{X}_k$)
L of past actions
$\begin{bmatrix} u_{k-L} \\ \vdots \\ u_k \end{bmatrix}$

Figure 4.3: Overview of asynchronous training architecture used for data collection and model training. A central simulation server controls the tasks workers have to execute. Multiple simulation clients run one iteration of SLQ-MPC with DSKBM over a randomized track to collect varied trajectory data. The server compiles data collected by the client and sends it to the ModelTrainer. The ModelTrainer samples a window of sequence length L from all collected data and packages it into a window of past states and actions, and the ground truth state derivative.

the predictions from the covariance branch is manipulated to fit the lower-triangular elements of the Cholesky factor. An element-wise softplus is applied to the diagonal elements to enforce positivity. Softplus is a smooth approximation to ReLU and can be used to constrain the output of a machine to always be positive and is given by:

$$softplus(x) = \frac{1}{\beta} \log(1 + \exp(\beta * x)) \tag{4.6}$$

Entries above the diagonal are zeroed out, and entries including the diagonal and below are assembled to be the lower-triangular component of a Cholesky factorization with a strictly positive diagonal.

While the exploration of uncertainty-aware corrections is beyond the primary scope of this thesis, the probabilistic nature of the CRN sets up numerous possibilities in future work. These possibilities are discussed in the concluding chapter.

4.2.3 Training Architecture

A distributed training architecture inspired by the approach used in Wang et. al was implemented for efficient data collection and training of the CRN [16, 43, 44].

The training server manages client connections and sends tasks to idle connected clients. The server repeats this process until a user-defined number of trajectories have been collected. Each simulation client independently samples a random track by selecting five points uniformly within the bounds of the experiment and initializes the robot. The points are connected sequentially in order of sampling.

Each client's robot uses the MPC Path Tracker detailed in algorithm 1 to track the path. At each timestep, the ground truth of the robot's current state, its state derivatives, as well as its actions are recorded. These are stored in a TensorDict sturcture for subsequent training.

Concurrently, the ModelTrainer commences training with training data already collected. During each training iteration, the ModelTrainer randomly samples a trajectory out of all collected trajectories. Within the randomly sampled trajectory, the ModelTrainer samples a valid window corresponding to the sequence length of the LSTM. To obtain the residual the model has to learn from, the ModelTrainer uses the final state action pair within the sequence length as the current state and queries the DSKBM for its predictions. The residual is, as previously alluded in Equation 4.4, defined as:

$$r(x, u) = \dot{x}_{sim} - \dot{x}_{model} \tag{4.7}$$

As the goal was to train a probabilistic corrective network, training is performed using a probabilistic objective. The two branches of the CRN form the a predicted mean and a covariance matrix over the residuals. The log-likelihood of the observed residual under this distribution is computed, and the negative log-likelihood is minimized using Adam [53].

To accelerate and stabilize learning, learning rate scheduling like that in [54] was implemented. The learning rate scheduler has an initial warm-up phase and controlled decay to facilitate convergence initially while preventing instability in given more data.

4.3 Corrective Sequential Linear-Quadratic MPC

4.3.1 Sequential Linear-Quadratic MPC (SLQ-MPC)

Algorithm 2 Sequential Linear–Quadratic MPC (SLQ–MPC)

```
Require: Cost matrices $Q \succeq 0,\ Q_f \succeq 0,\ R \succ 0,\ R_\Delta \succeq 0$
Require: Current state $x_k$, initial control guess $\{\bar{u}_i\}_{i=0}^{T-1}$
Require: Reference trajectory $\{x_{\text{ref},k+i}\}_{i=0}^{T}$, horizon $T$, time step $dt$, inner iterations $N_{\text{iter}}$
 1: for $\ell \leftarrow 1$ to $N_{\text{iter}}$ do
 2:     $\bar{x}_0 \leftarrow x_k$
 3:     $l\_dynamics \leftarrow [\,]$
 4:     for $i \leftarrow 0$ to $T-1$ do
 5:         $(A_i, B_i, C_i) \leftarrow \text{LINEARIZE}(\bar{x}_i, \bar{u}_i)$
 6:         Append $A_i, B_i, C_i$ to $l\_dynamics$.
 7:         $\bar{x}_{i+1} \leftarrow A_i\bar{x}_i + B_i\bar{u}_i + C_i$
 8:     end for
 9:     QP $\leftarrow \text{FORMQP}(\bar{x}_0, l\_dynamics, Q, Q_f, R, R_\Delta, x_{\text{ref}})$
10:     Warm-start solver with $\{\bar{u}_i\}_{i=0}^{T-1}$
11:     $\{u_i^\star\}_{i=0}^{T-1} \leftarrow \text{SOLVEQP}(\text{QP})$
12:     if $\max_i \|u_i^\star - \bar{u}_i\| < \varepsilon$ then
13:         break                                        ▷ Check for convergence
14:     end if
15:     $\bar{u} \leftarrow u^\star$                     ▷ Use optimal controls as new nominal
16: end for
17: return $u_0^\star$
```

Sequential Linear-Quadratic Model Predictive Control (SLQ-MPC) is a first-order collocation method that solves a nonlinear optimal control problem by iteratively approximating it as a sequence of convex quadratic subproblems. It has parallels to Sequential Quadratic Programming (SQP), though SQP is typically formulated with direct collocation, and has stricter convergence checks corresponding to satisfaction of the nonlinear system's KKT conditions after linearization. The core algorithm is presented in Algorithm 2.

At each iteration, the algorithm optimizes for the optimal control sequence $\{u_t\}_{t=0}^{T-1}$. During each time step of the forward rollout, the nonlinear dynamics is linearized into the state-transition matrix, A_k, the input matrix, B_k, and the affine offset to shift the system back to the origin, C_k. These matrices are cached for building the QP subproblem. The corresponding state sequence is directly obtained from a forward rollout through the linearized dynamics around the current operating point.

During the sequential linear-quadratic refinement of the solution, a quadratic program is assembled according to 2.53.

During the very first iteration, a set of dummy actions with zero steering and a user-defined forward acceleration is given to the solver as an operating point.

$$\bar{u}_{\text{init}} = \begin{bmatrix} a_0 & \cdots & a_H \\ 0 & \cdots & 0 \\ 0 & \cdots & 0 \end{bmatrix} \tag{4.8}$$

Although the corresponding states and control inputs are unlikely to lie in the neighborhood of the true optimal solution, they allow formation of a locally valid linear-quadratic approximation of the system. The solution to the locally valid QP yields a control sequence that is closer to the optimal solution, which is used to warm-start the next iteration. Each successive iteration produces a trajectory with control sequences approaching the optimal controls, and with constraints that better reflect the true non-linear dynamics [7].

The process repeats until the change in control inputs between iterations falls below a threshold, or for a fixed number of iterations, following that of a RTI scheme for MPC. This provides a balance between computational efficiency and solution accuracy. SLQ-MPC allows the subproblems to stay convex and well defined, allowing solutions to remain computationally tractable while yielding control sequences that approach the optimal solution of the original nonlinear problem.

4.3.2 Corrective Sequential Linear-Quadratic MPC

Algorithm 3 Corrective Sequential Linear–Quadratic MPC (CSLQ–MPC)

```
Require: Cost matrices $Q \succeq 0,\ Q_f \succeq 0,\ R \succ 0,\ R_\Delta \succeq 0$
Require: Current state $x_k$, initial control guess $\{\bar{u}_i\}_{i=0}^{T-1}$
Require: Reference trajectory $\{x_{\text{ref},k+i}\}_{i=0}^{T}$, horizon $T$, timestep $dt$, inner iterations $N_{\text{iter}}$
Require: Corrective Residual Network CRN (LSTM) with window size $L$
 1: $\text{CRN}_k \leftarrow \text{DeepCopy}(\text{CRN})$
 2: for $\ell \leftarrow 1$ to $N_{\text{iter}}$ do
 3:     $\text{CRN}_\ell \leftarrow \text{CRN}_k$
 4:     $\bar{x}_0 \leftarrow x_k$
 5:     $l\_dynamics \leftarrow [\,]$
 6:     for $i \leftarrow 0$ to $T-1$ do
 7:         $(A_i, B_i, C_i) \leftarrow \text{Linearize}(\bar{x}_i, \bar{u}_i)$
 8:         $C_{\text{crn}} \leftarrow \text{CRN}_\ell(\text{History}_{i-L:i-1})$
 9:         Append $(A_i, B_i, C_i)$ to l_dynamics
10:         $\bar{x}_{i+1} \leftarrow A_i\bar{x}_i + B_i\bar{u}_i + C_i$
11:     end for
12:     $\text{QP} \leftarrow \text{FormQP}(\bar{x}_0, l\_dynamics, Q, Q_f, R, R_\Delta, x_{\text{ref}})$
13:     Warm-start solver with $\{\bar{u}_i\}_{i=0}^{T-1}$
14:     $\{u_i^\star\}_{i=0}^{T-1} \leftarrow \text{SolveQP}(\text{QP})$
15:     if $\max_i \|u_i^\star - \bar{u}_i\| < \varepsilon$ then
16:         break                                  ▷ Check for convergence
17:     end if
18:     $\bar{u} \leftarrow u^\star$                ▷ Use optimal controls as new nominal
19: end for
20: return $u_0^\star$
```

This thesis proposes Corrective Sequential Linear-Quadratic MPC (CSLQ-MPC), a method that integrates the learned dynamics corrections from the CRN.

The CRN is used to compute corrections to the dynamics model. The predicted mean from the CRN is used directly as an affine offset to the linearized dynamics computed by the model. The proposed method therefore amends the previous affine offset matrix with:

$$C_{eff} = C_{linearized} + \hat{r}_\theta(x, u) \tag{4.9}$$

This modification corrects the local linear approximation by explicitly accounting for unmodeled effects that the CRN has managed to learn not captured by the nominal model.

The LSTM used in the Corrective Residual Network (CRN) is stateful and maintains an internal hidden state across time steps to encode temporal information from past inputs. This introduces an additional layer of complexity where the CRN has to be handled carefully to when it is used to compute dynamics corrections within each iteration of CSLQ-MPC.

To address this, CSLQ-MPC adopts the following procedure:

- Before the start of each SLQ-MPC iteration from $\ell = 1 : N_{iter}$, a deep copy of the CRN is created. This ensures that correction computations for the current state and action sequence are isolated and do not modify the original CRN's internal state.
- After solving the QP subproblem and optaining a suboptimal control sequence, CRN is reset to the deep-copied instance.
- After N_{iter} or convergence, whichever comes earlier, CRN is reset once again to the deep-copied instance, and updated with the current state and optimized action the robot will execute, ensuring the CRN only maintains a hidden state of the actual state and action being used.

Chapter 5

Results

To evaluate the proposed method, 9 distinct evaluation tracks were created using a sampling method from that used during training. This ensures that the robot is evaluated on tracks it has never seen before, testing the model's ability to generalize.

Each evaluation track is generated using the Poisson sampling disk algorithm [55]. The algorithm randomly samples five points in space using a radius of 5m. These points are sorted in order according to their angle from the origin in order to create a more conventionally shaped track, as opposed to the random angular tracks faced during training. The resulting tracks are shown in Figure 5.1.

The robot platform used is a custom built, double-steer, car-like robot. The robot is introduced to the PyBullet simulator through a URDF file exported through CAD. The URDF file contains the mass and geometry of each of the components of the robot. The robot uses the DSKBM as defined in earlier chapters. The only parameters required by the DSKBM from the robot is the distance from the COM to its front and rear axles, shown in Table 5.1.

Three different models were trained, each using an LSTM architecture but with varying assumptions and objectives. The first two models trained were CRNs that correct

Table 5.1: KBM Parameters of the car-like robot

Parameter	Actual Params	Sys-ID Mismatch Params
l_f	0.342	0.484
l_r	0.342	0.200
wheelbase	0.684	0.684

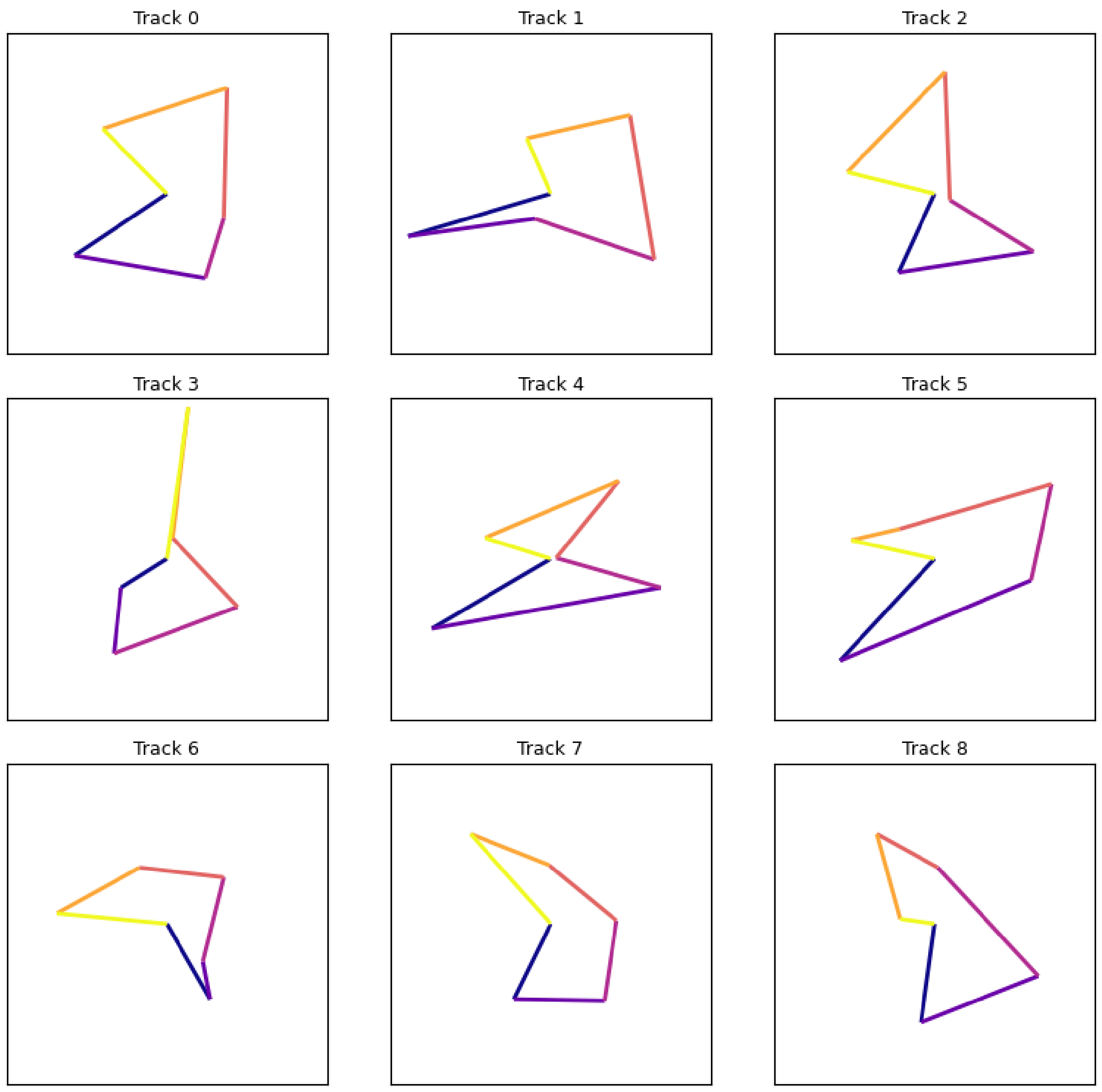


Figure 5.1: Evaluation tracks for experiments generated using Poisson disk sampling

Table 5.2: Summary of different models trained for experiments

Model Name	Nominal Model Used?	Model Purpose
CRN	Yes	Tests ability to compensate for modeling errors.
Mismatched CRN	Yes (with wrong parameters, $l_f = 0.484$, $l_r = 0.200$)	Tests robustness to poor system identification.
Predictive Dynamics Network	No	Evaluate performance without correcting a nominal model.

the nominal dynamics model they were trained alongside. The third model was trained to directly predict system dynamics without correcting a nominal model. A summary of the different models are shown in Table 5.2.

The CRN and Sys-ID Mismatch CRN are both corrective models. The CRN was trained to correct predictions from a nominal model using accurate system parameters. Experiments with this system demonstrate the CRN's ability to correct dynamics arising from nominal model mismatch with the actual dynamics. The Sys-ID Mismatch CRN was trained to correct predictions from a model that has incorrect system parameters. Experiments with this system demonstrate resilience to inaccurate identified system parameters. This network will be referred to as the Mismatched CRN. The CRN of the third system learns the dynamics directly instead of learning a residual. This network will be referred to as the Predictive Dynamics Network (PDN).

The third model, the predictive dynamics network, was trained to predict the full dynamics instead of a correction to a nominal model. This model will be used to evaluate the efficacy of residual models.

In all experiments, the robot attempts to follow the evaluation tracks using the CSLQ-MPC controller introduced in earlier chapters. For comparison, a baseline controller that uses SLQ-MPC with the DSKBM is run on the same tracks. During each run, the following data is logged:

- The true state derivatives queried from the simulator
- The predicted state derivatives from the learned model (with or without correction)
- The robot's trajectory

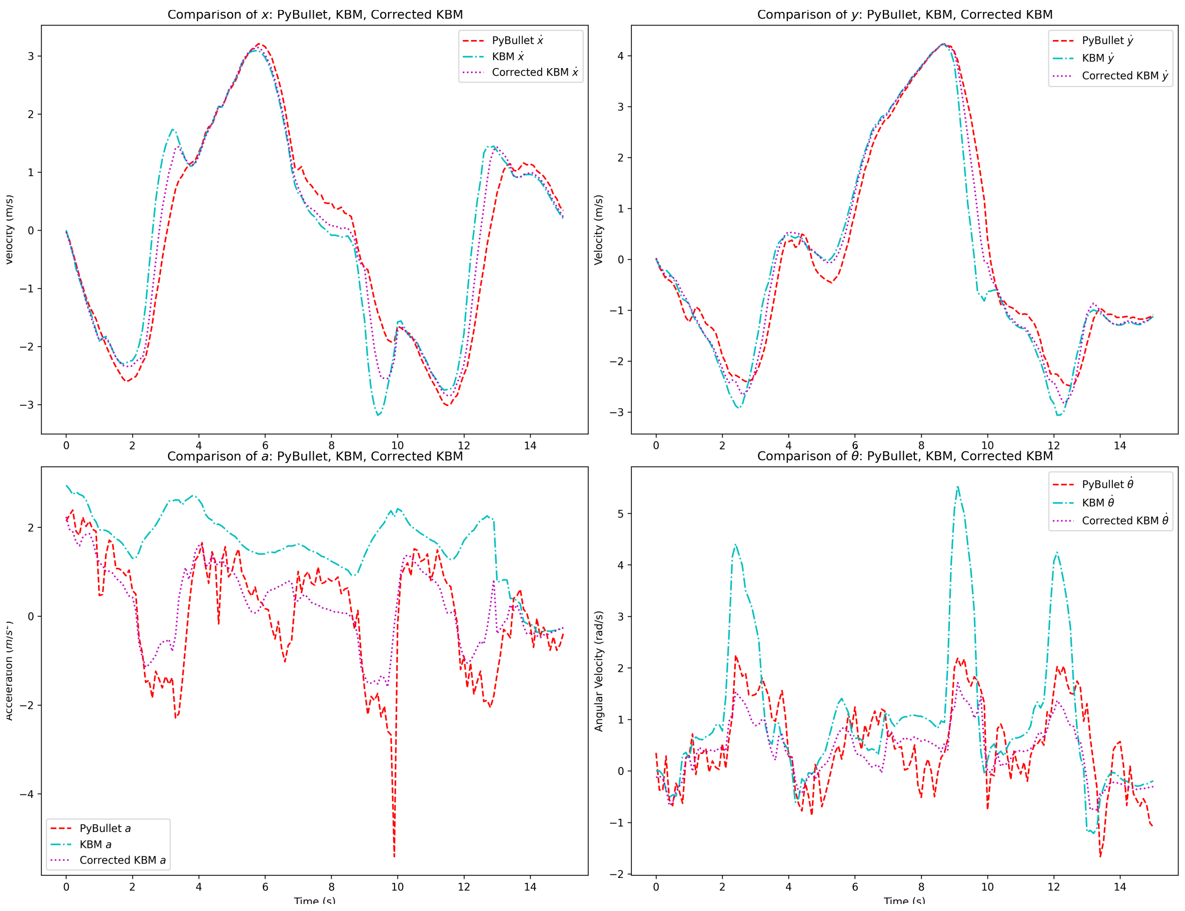


Figure 5.2: Predicted and Corrected State Derivatives vs. Simulation Ground Truth (Track 0)

5.1 Learned Dynamics

To evaluate the accuracy of the learned dynamics models, the predicted state derivatives from each model is compared against the ground truth values obtained from the PyBullet simulator.

Figure 5.2 shows an example from one evaluation track (Track 0), where we compare:

- Predictions from the baseline DSKBM
- Predictions from DSKBM corrected by the CRN
- The ground truth state derivatives from simulation.

In Figure 5.2, the nominal model predictions for $\dot{x}$ and $\dot{y}$ are close to the ground truth.

However, the CRN significantly improves accuracy across all predicted states. The CRN makes large corrections especially to both predicted acceleration and angular velocity, visibly reducing the mismatch between the nominal model and the simulator.

5.1.1 Corrective Residual Network (CRN)

The CRN model is evaluated across all 9 evaluation tracks. Figure 5.3 shows the average root mean square error (RMSE) for each state derivative, averaged across all 9 evaluation tracks, for both predictions from the nominal DSKBM and predictions from the CRN compared against the ground truth from the simulator. Figure 5.4 shows the reduction in average RMSE for predictions made using the CRN.

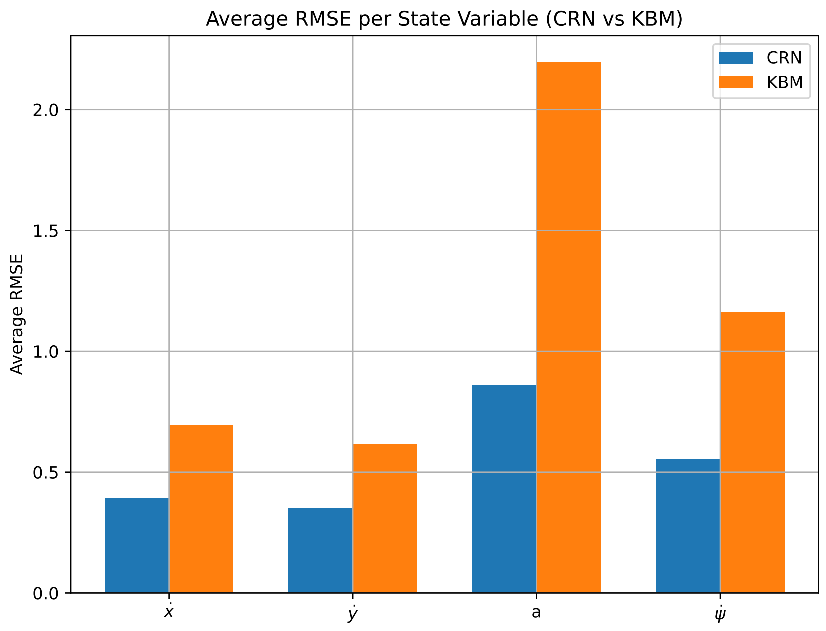


Figure 5.3: Average RMSE per Predicted State Derivative (CRN, KBM vs. Ground Truth) over all Evaluation Tracks

5.1.2 Predictive Dynamics Network (PDN)

The PDN shows an overall reduction in average RMSE when compared to the baseline DSKBM. As shown in Figure 5.5, the model achieves lower prediction error for most state derivatives.

Figure 5.6, which shows RMSE reduction for a run on each debug track, shows that in certain evaluation tracks, average RMSE of some state derivatives are actually slightly

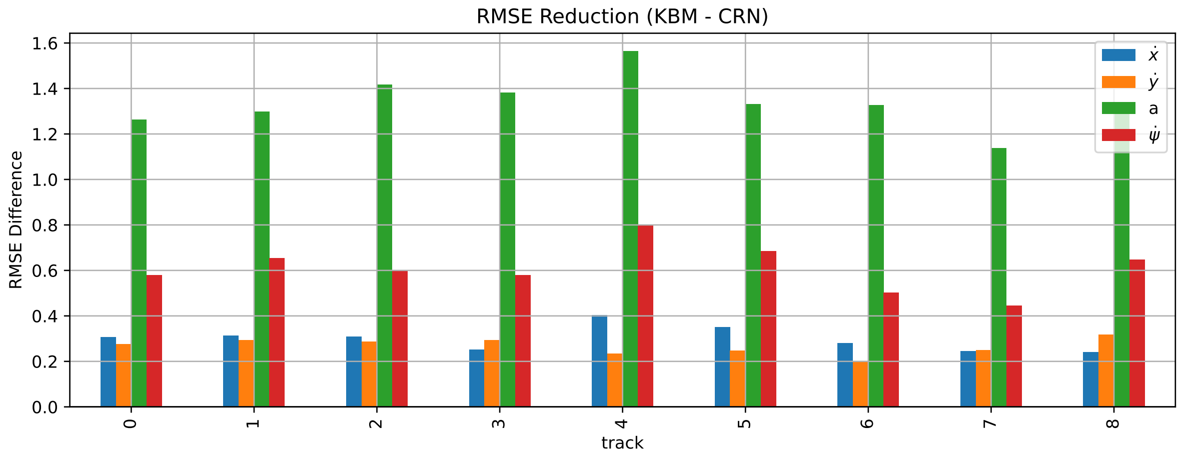


Figure 5.4: RMSE Reduction per State Derivative (DSKBM - CRN) over all Evaluation Tracks

worse than that of predictions from the nominal model. These inconsistencies suggest that the model is struggling to generalize for certain states across varying track geometries.

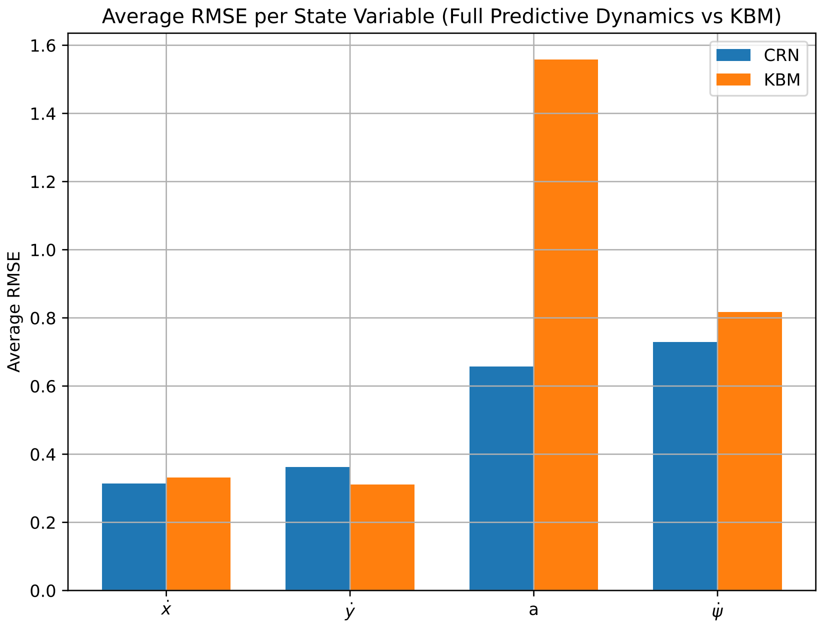


Figure 5.5: Average RMSE per State Derivative

Figure 5.7 compares the average RMSE across all state derivatives for both the full predictive model and the CRN. The full predictive model outperforms CRN in predicting longitudinal acceleration, a, and slightly outperforms it in predicting forward velocity $\dot{x}$. However, CRN performs marginally better in predicting lateral velocity ,$\dot{y}$, and more consistently outperforms the full model in predicting yaw rate ($\dot{\psi}$).

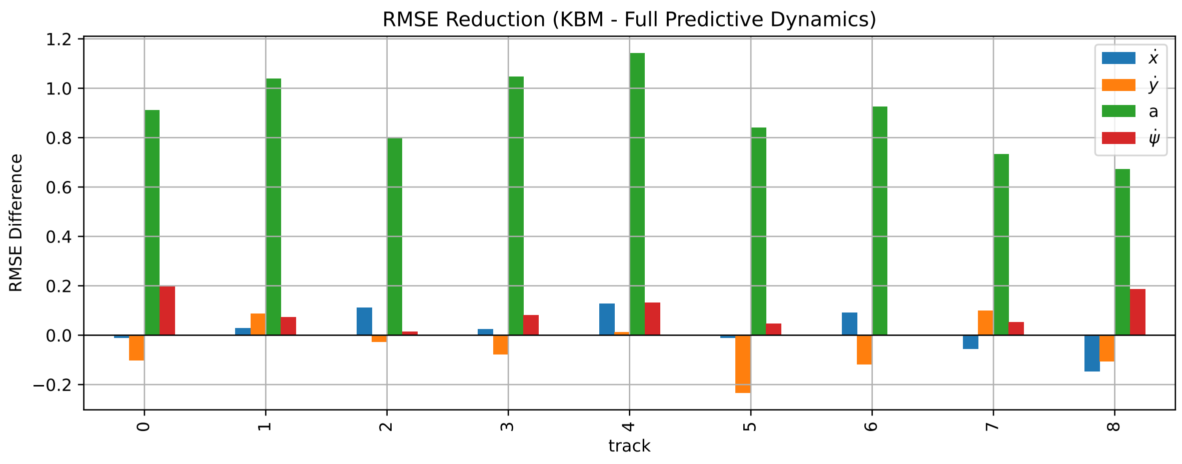


Figure 5.6: RMSE Reduction per State Derivative (DSKBM - CRN)

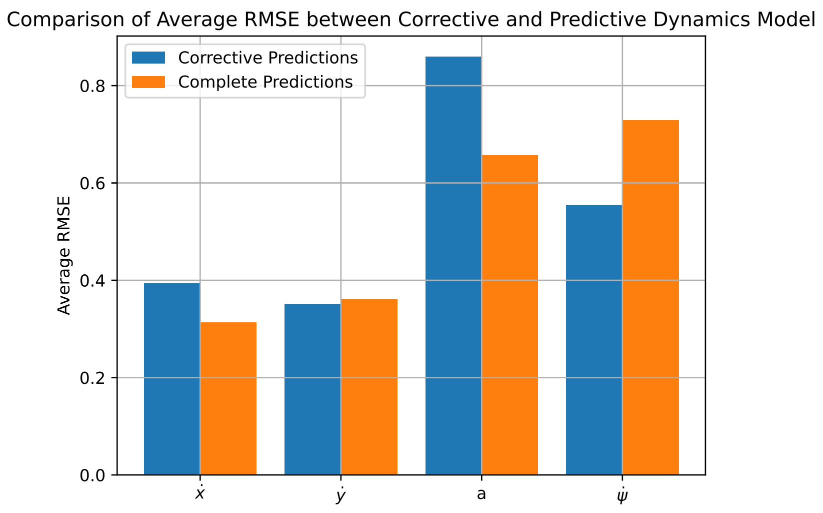


Figure 5.7: RMSE Comparison between Corrective and Full Predictive Models

5.2 Path Tracking MPC

This section evaluates the path-tracking ability of CSLQ-MPC (Algorithm 2) equipped with different variants of CRNs specified in Table 5.2 under different scenarios. Performance is measured using two metrics:

- Cross-track error (CTE): the average lateral deviation from the closest point on the reference path.
- Time: the duration required to complete a full lap of each evaluation track.

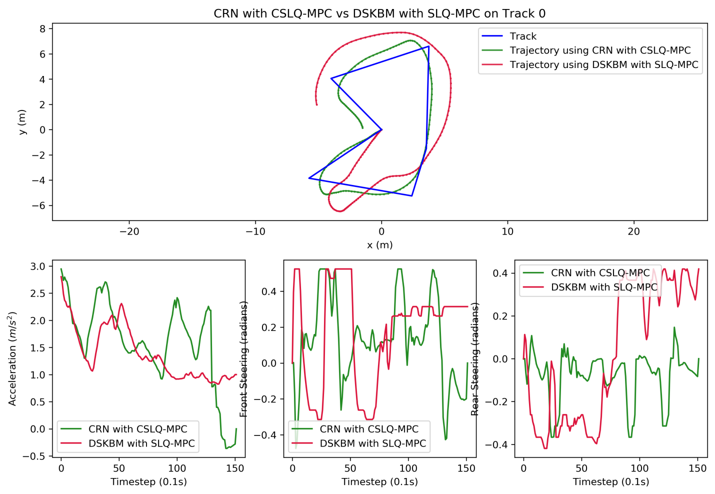
Table 5.3 summarizes the mean and standard deviation of both metrics across all 9 evaluation tracks. While the quantitative results appear to suggest that the CSLQ-MPC with the PDN has lowest cross-track error, runs using PDN take significantly more time than runs with the CRN. Additionally, the qualitative results show areas in which CSLQ-MPC with the PDN struggle.

Path Tracker	Cross-Track Error	Time Taken (μ)
CRN with CSLQ-MPC	0.57 ± 0.11	**15.6 ± 2.5**
DSKBM with SLQ-MPC	1.24 ± 0.35	18.2 ± 2.3
Mismatched CRN with CSLQ-MPC	0.63 ± 0.18	16.1 ± 2.3
Mismatched DSKBM with SLQ-MPC	2.05 ± 0.63	19.2 ± 2.3
PDN with CSLQ-MPC	**0.52 ± 0.05**	18.2 ± 2.2

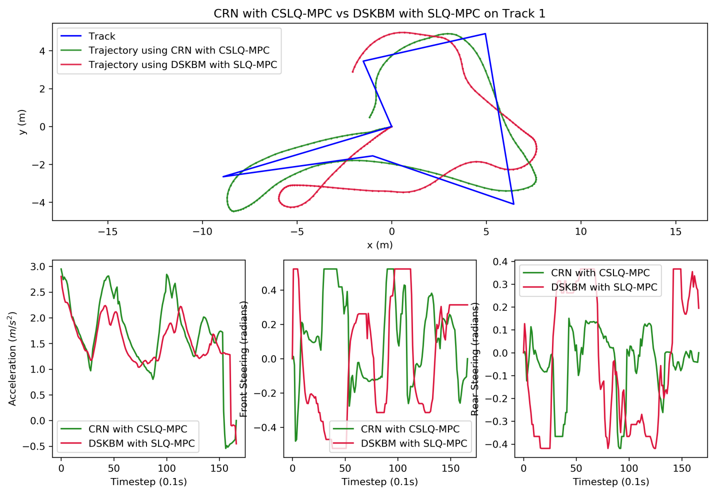
Table 5.3: Summary of metrics averaged across all evaluation runs using CRN, mismatched CRN, and PDN.

5.2.1 CSLQ-MPC (CRN) vs. SLQ-MPC (DSKBM)

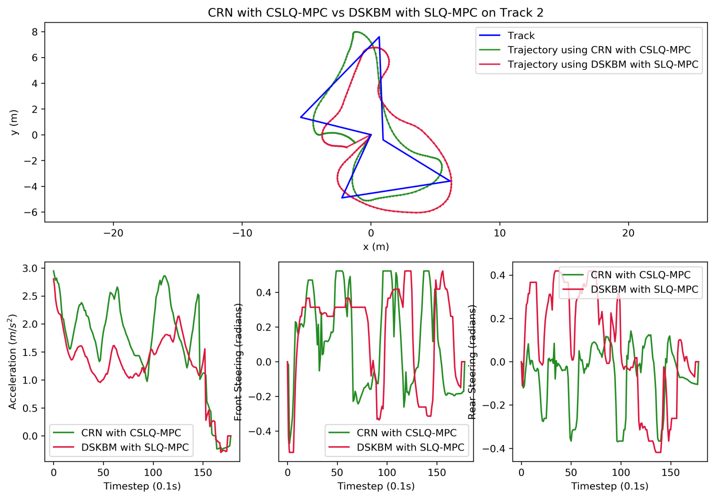
Figures 5.8, 5.9, 5.10 shows qualitative results of the proposed CSLQ-MPC path tracker with a CRN, compared alongside results of the SLQ-MPC path tracker using the DSKBM. The results demonstrate accurate tracking even in extreme angular evaluation tracks such as Track 1, 3, 4 shown in Figure 5.1 with minimal fluctuation in the control commands. The CSLQ-MPC with CRN is able to follow even difficult tracks reasonably well with data-driven corrections to a simple nominal model.



(a)

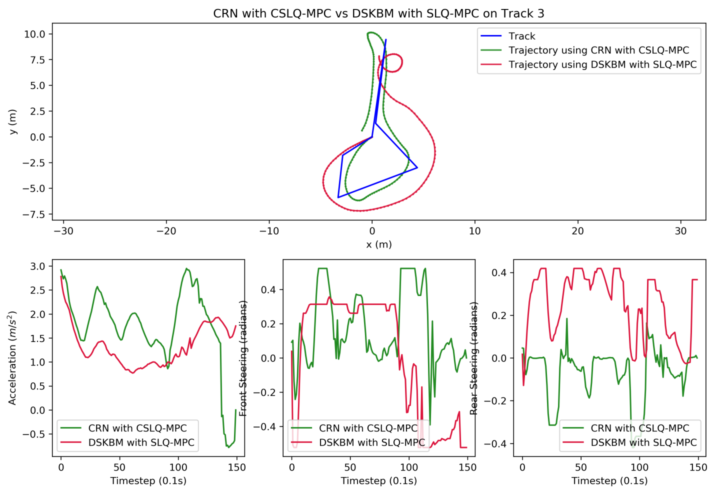


(b)

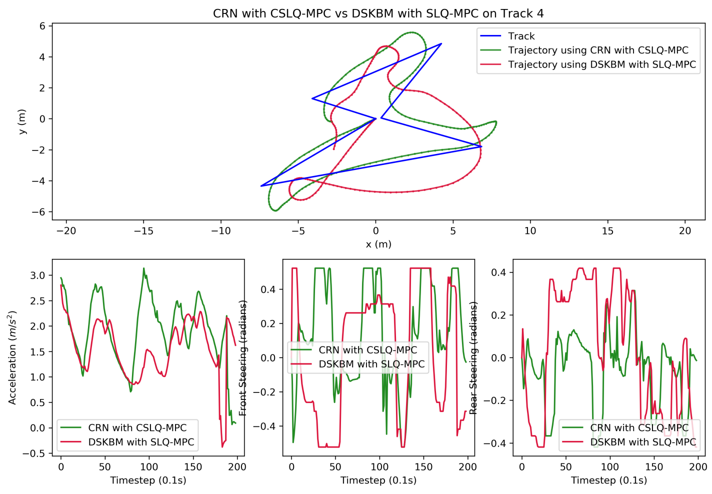


(c)

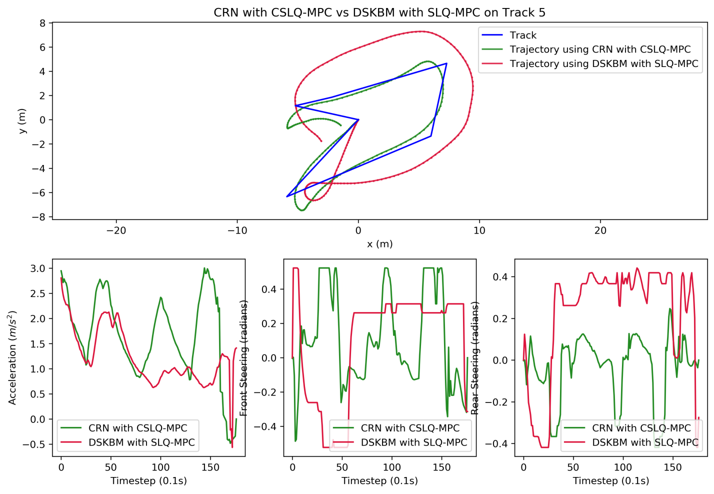
Figure 5.8: CRN with CSLQ-MPC vs DSKBM with SLQ-MPC on Tracks 0–2



(a)

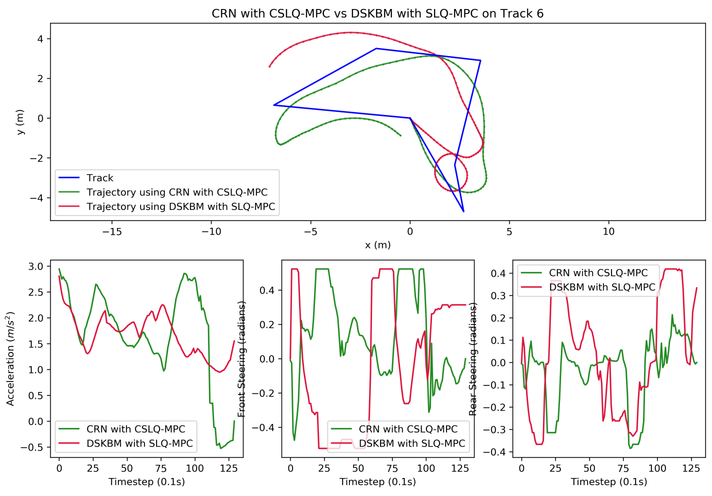


(b)

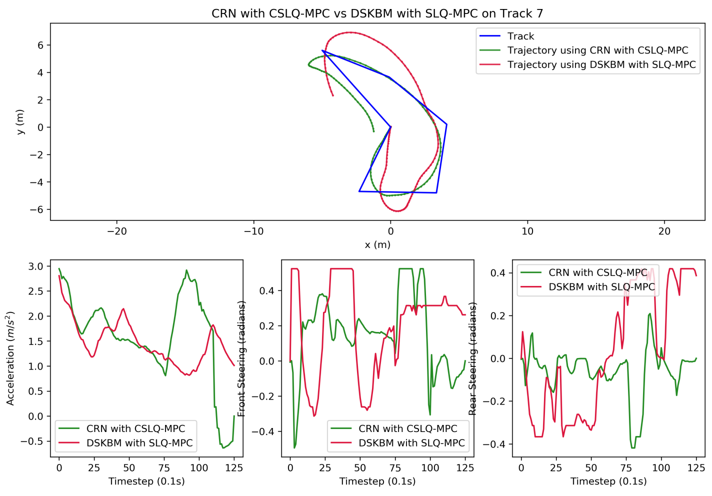


(c)

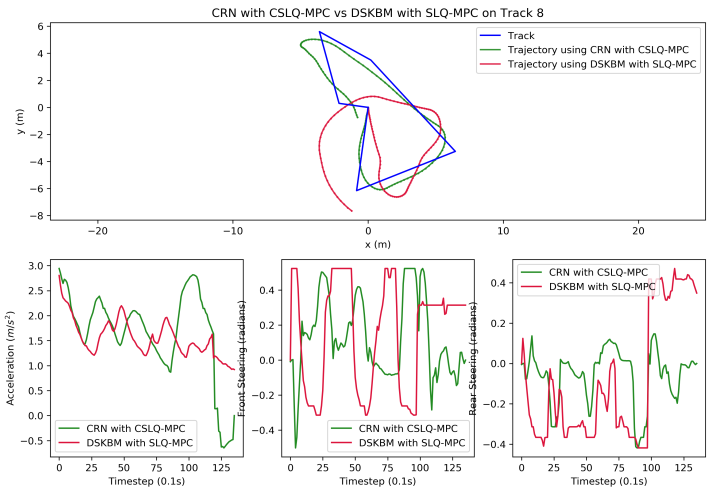
Figure 5.9: CRN with CSLQ-MPC vs DSKBM with SLQ-MPC on Tracks 3–5



(a)



(b)



(c)

Figure 5.10: CRN with CSLQ-MPC vs DSKBM with SLQ-MPC on Tracks 6–8

Due to the simplicity of the DSKBM model being unable to capture the dynamics of the robot at the given velocity, the path tracker struggles to follow even simple tracks well, as seen in the red trajectories in figures 5.8, 5.9, 5.10.

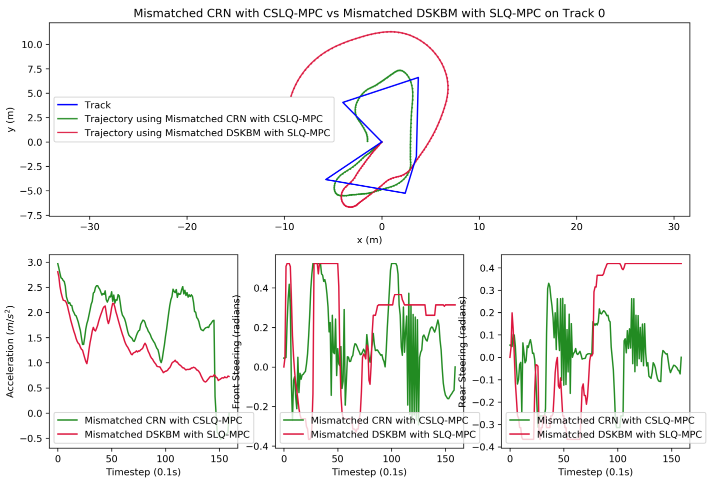
In Figure 5.8b, the robot is unable to track the first tight bend, which leads the robot to drive in a loop. In Figure 5.9a, the robot once again exhibits circular motions as it struggles to track the extremely sharp bend. Across all runs, the robot demonstrates drift, underscoring the models' inability to accurately represent the system dynamics at the given velocity. This is also reflected in the higher average CTE in Table 5.3.

Comparing the results of the CSLQ-MPC path tracker with a CRN to the results of SLQ-MPC path tracker with DSKBM highlight the ability of the CSLQ-MPC algorithm to learn the true underlying dynamics of the system. While the SLQ-MPC path tracker fails to track reference trajectories even in simple tracks, given data-driven corrections, CSLQ-MPC manages to track difficult, angular tracks such as that shown in Figure 5.9a.

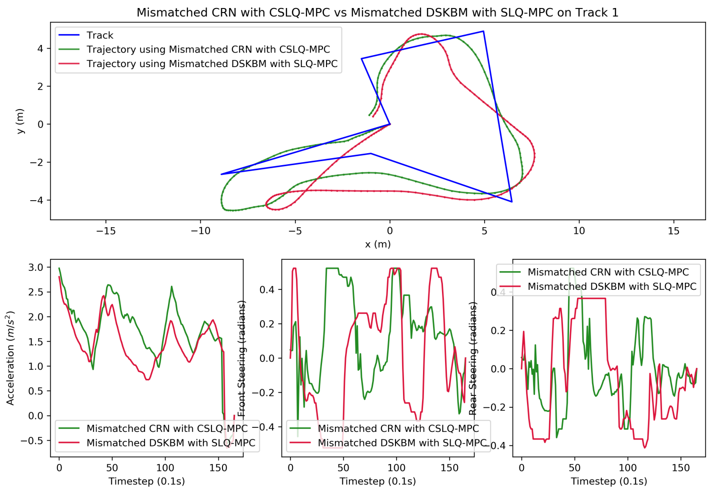
5.2.2 CSLQ-MPC (CRN) vs. SLQ-MPC (DSKBM), both with model parameters mismatch

The next runs demonstrate the ability of the CRN to adapt to large model parameter mismatch, while demonstrate the fragility of MPC schemes to inaccurate system identification. Figures 5.11, 5.12, 5.13 show the results of the CSLQ-MPC path tracker with a CRN running on all evaluation tracks. The nominal model CRN is trained to correct has wrong parameters detailed in Table 5.2.

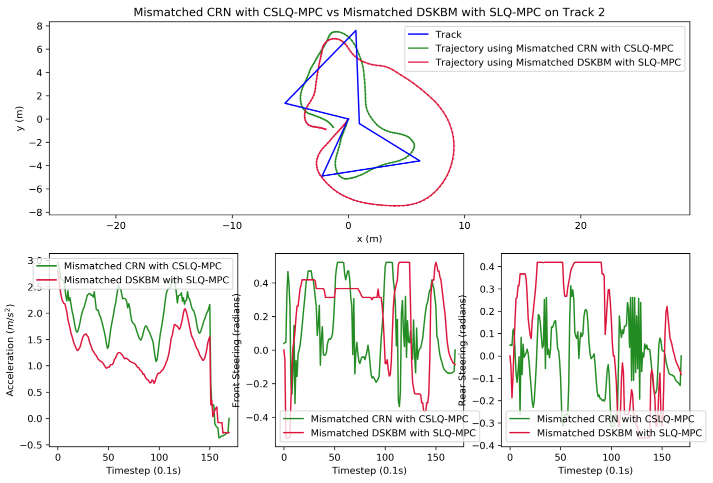
Despite having a nominal model with wrong system parameters, the CRN is capable of learning a correction to the dynamics that is representative of the true dynamics. CSLQ-MPC with mismatched system parameters outperforms SLQ-MPC with a nominal model using correct system parameters. Note that the control sequences solved by CSLQ-MPC with model parameter mismatch is noticeably noisier than CSLQ-MPC without, and the model struggles, while still managing to complete, tougher evaluation tracks like that in Figure 5.12a.



(a)

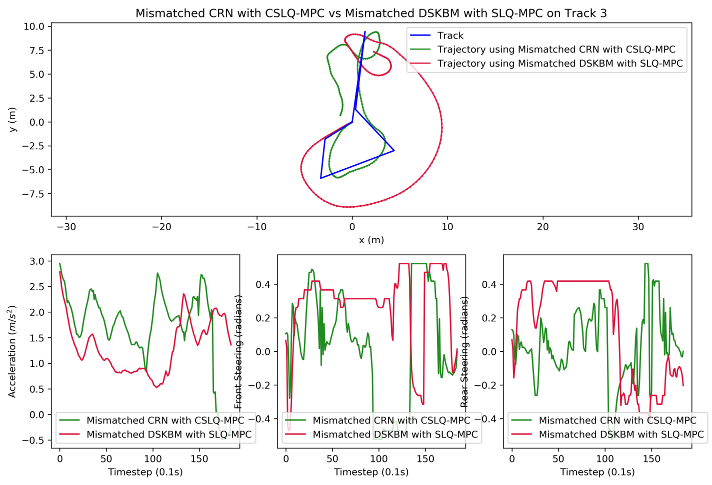


(b)

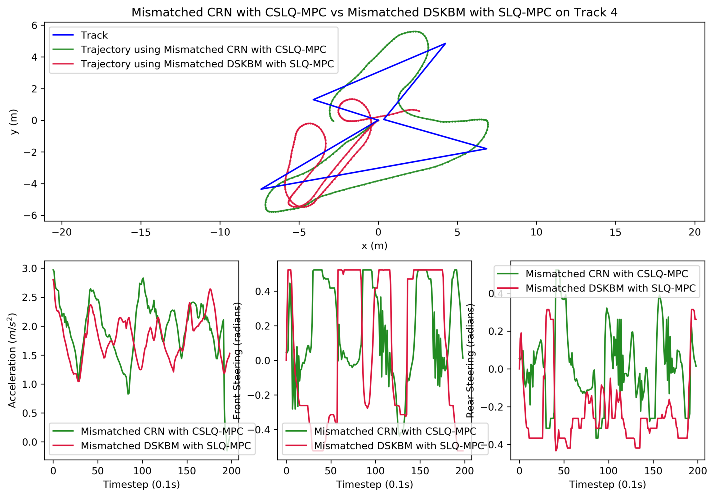


(c)

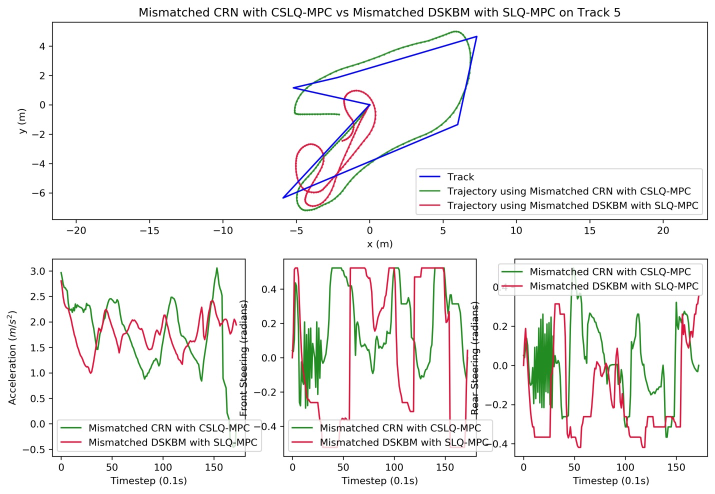
Figure 5.11: Mismatched CRN with CSLQ-MPC vs Mismatched DSKBM with SLQ-MPC evaluation on tracks 0–2



(a)

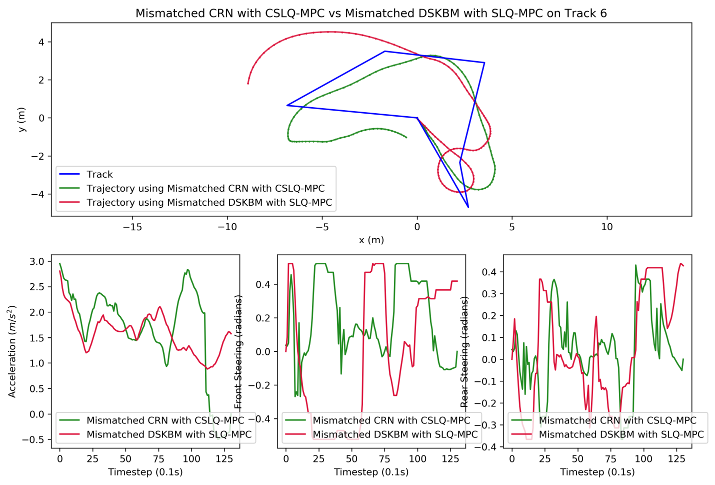


(b)

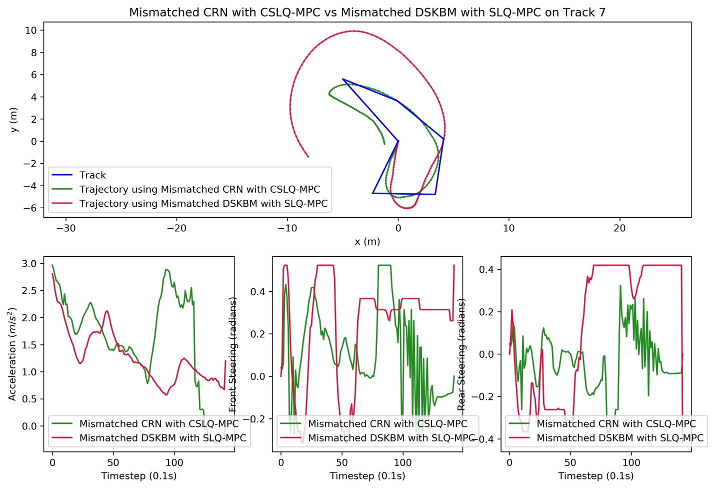


(c)

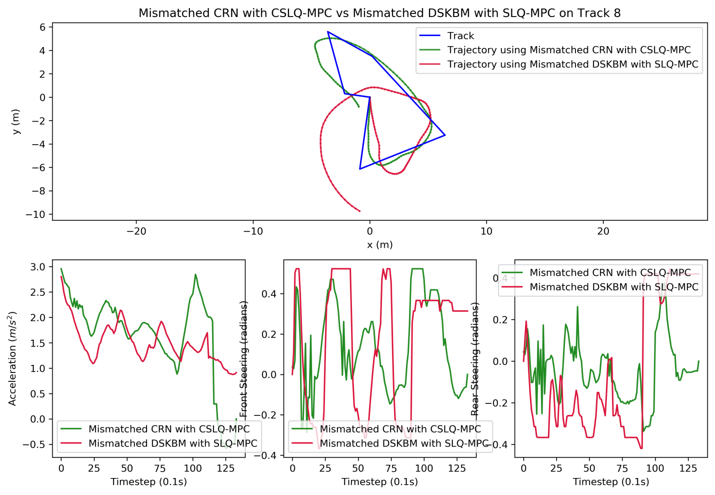
Figure 5.12: Mismatched CRN with CSLQ-MPC vs Mismatched DSKBM with SLQ-MPC evaluation on tracks 3–5



(a)



(b)



(c)

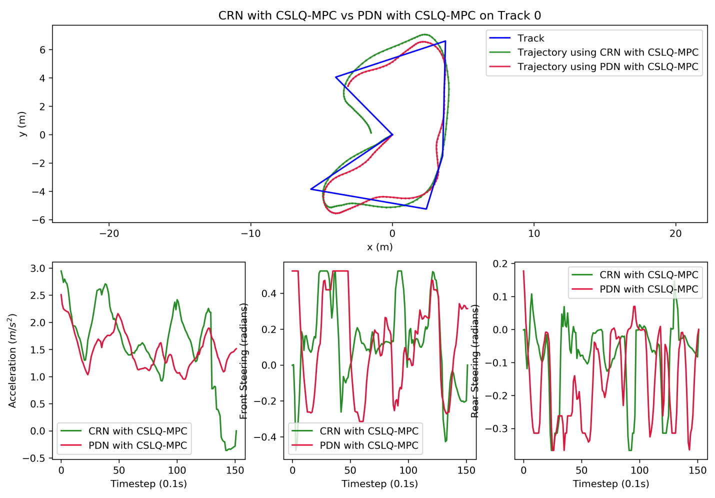
Figure 5.13: Mismatched CRN with CSLQ-MPC vs Mismatched DSKBM with SLQ-MPC evaluation on tracks 6–8

On the other hand, SLQ-MPC equiped with a DSKBM with model parameter mismatch performs poorly over the evaluation tracks. Figures **??**, 5.12, 5.13 show the results of evaluation runs on all test tracks. Given the results with the DSKBM with correct system parameters, unsurprisingly, SLQ-MPC path tracker with parameter mismatch performs worse. In Figure 5.11a, 5.11c, 5.13a, 5.13b, 5.13c, we see large drift as the algorithm is unable to correct for turns well. In Figures **??**, **??**, we see the algorithm struggle to follow the sharp angular turns of the evaluation track.

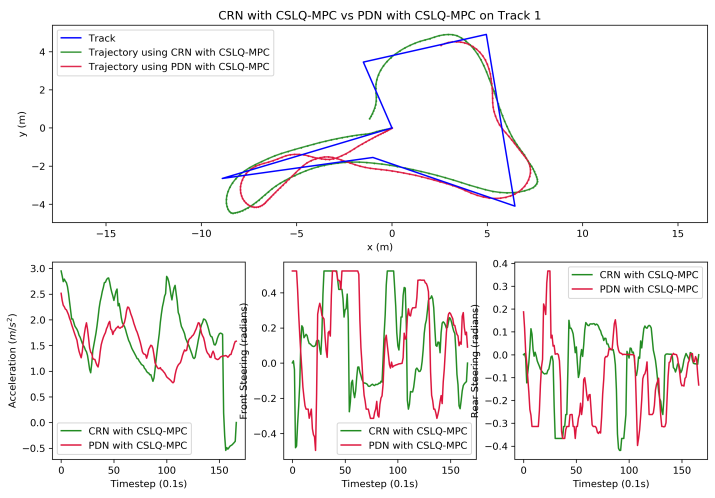
These results clearly demonstrate the ability of CSLQ-MPC to compensate for errors in system identification of model parameters. With just two mismatched parameters in the DSKBM, SLQ-MPC suffers from severe tracking issues, highlighting the sensitivity of model-based control to incorrect dynamics. CRN learns a residual correction that enables CSLQ-MPC to maintain accurate tracking performance, even when the underlying nominal model is flawed.

5.2.3 CSLQ-MPC (PDN)

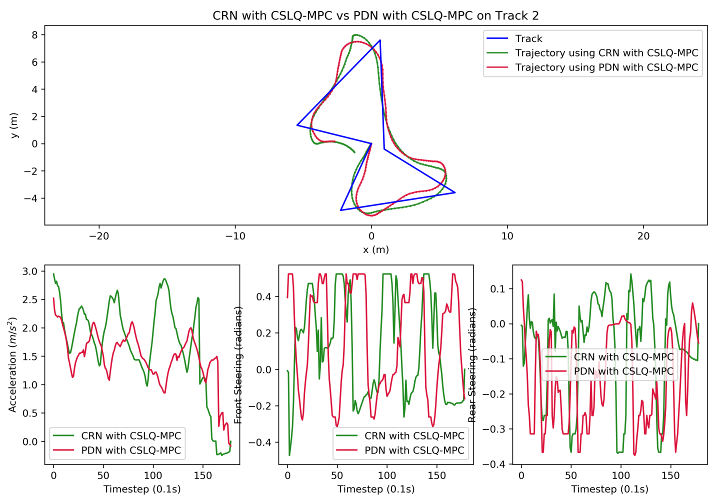
The PDN is evaluated with a modified version of CSLQ-MPC. As the PDN predicts the full dynamics of the system, no nominal model is used within this version of CSLQ-MPC, with the PDN instead providing an affine offset to the dynamics for each of the optimization subproblems. Figures 5.14, 5.15, 5.16 show the results of path tracking using the PDN.



(a)

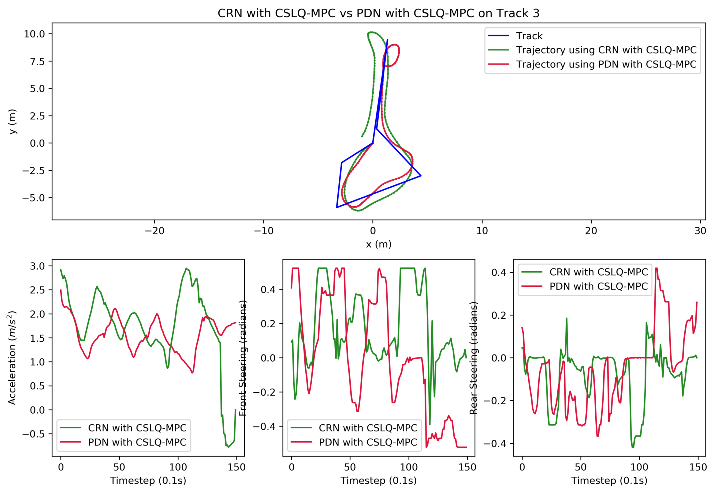


(b)

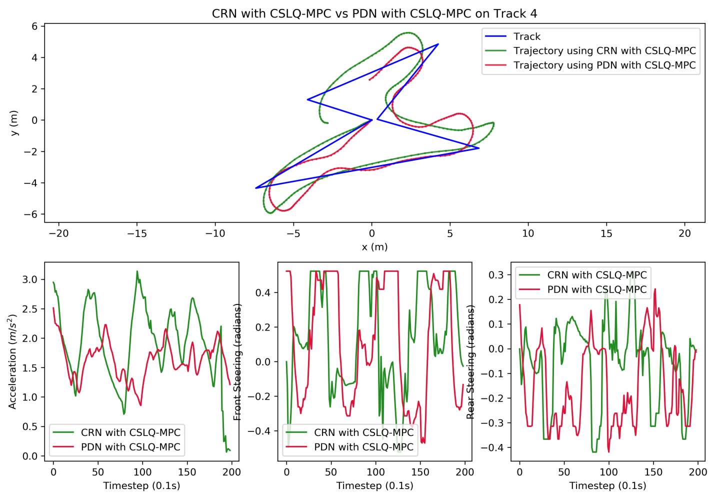


(c)

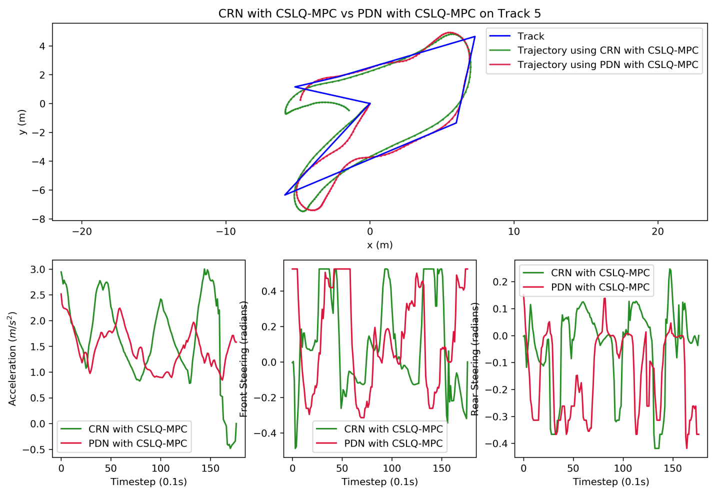
Figure 5.14: CRN vs. PDN evaluation on tracks 0–2



(a)

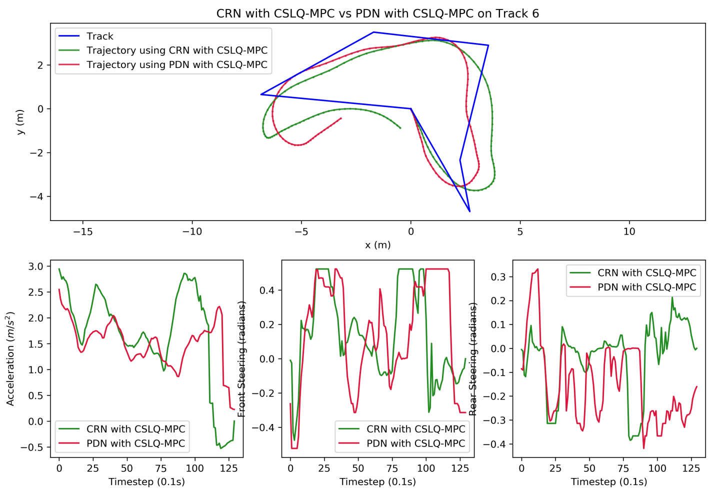


(b)

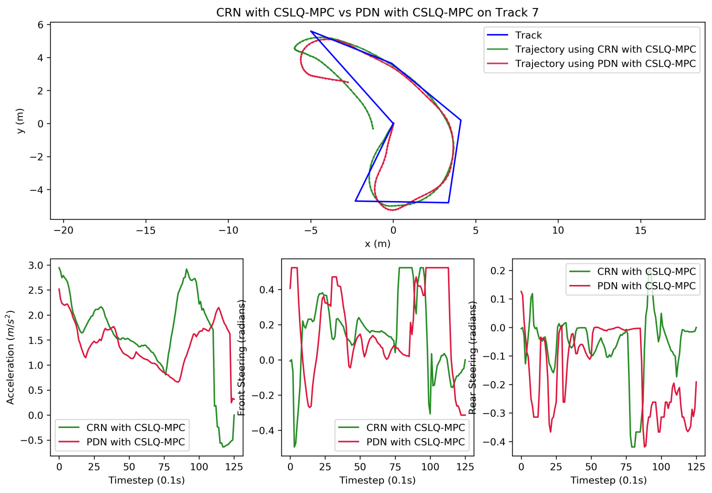


(c)

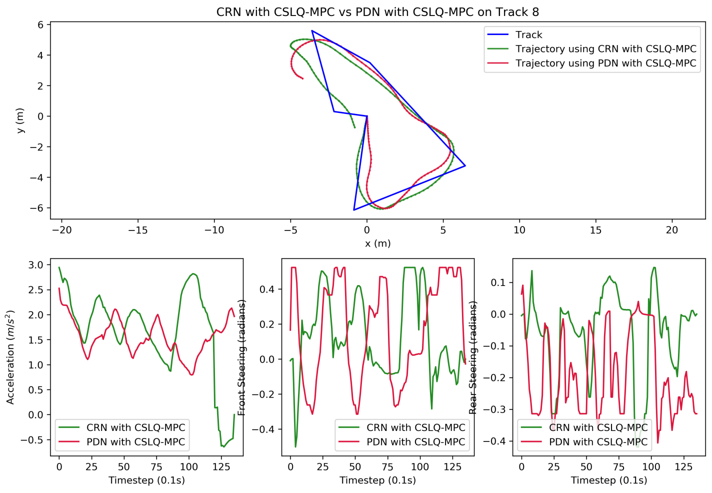
Figure 5.15: CRN vs. PDN evaluation on tracks 3–5



(a)



(b)



(c)

Figure 5.16: CRN vs. PDN evaluation on tracks 6–8

While CSLQ-MPC with the PDN is able to track paths somewhat accurately, in Figure 5.15a the algorithm fails to handle the sharp bend that CSLQ-MPC with CRN handles accurately. Overall, the robot moves at a much lower speed as well, and the robot fails to complete track 4, shown in Figure 5.15b. The results demonstrate that CSLQ-MPC with CRN still works more reliably and is able to control the robot at higher speeds, without adding too much additional cross track error, yielding better performance.

Chapter 6

Conclusion

6.1 Conclusion

This thesis addresses the challenge of achieving robust and efficient control for car-like robots under model mismatch, specifically when simplified dynamics models are used with potentially incorrect system parameters. The Double-Steer Kinematic Bicyle Model (DSKBM) was chosen as a simple nominal model due to its simplicity and real-time feasibility. However, it omits dynamical factors such as mass, friction, and tire forces.

To compensate for these limitations, the Corrective Residual Network (CRN) is proposed. The CRN uses an LSTM architecture to effectively correct the simple nominal model. A common real-time iteration MPC scheme, Sequential Linear-Quadratic Programming MPC (SLQ-MPC), is modified to include the CRN. The proposed algorithm, Corrective SLQ-MPC (CSLQ-MPC), is a computationally efficient method that significantly improves path-tracking performance. Notably, the CRN was able to compensate for errors when the underlying DSKBM used incorrect system parameters, enabling the robot to follow reference trajectories with CSLQ-MPC despite being given the wrong model. CSLQ-MPC with the CRN leads to improvement in cross-track error of 54.3% and an improvement in time taken of 14.3% over SLQ-MPC with the DSKBM.

This hybrid approach enables improved control performance without requiring complex or high-fidelity models, making it well-suited for low-cost custom-built robots where full system identification is impossible or impractical.

6.2 Future Works

Several promising directions can further extend the capabilities of the proposed algorithms within this thesis. Future work include uncertainty-aware control, multi-modal sensor integration, and integration of the CRN with planning.

The CRN outputs probabilistic predictions, which can be used to quantify uncertainty over the system's dynamics. In future work, this uncertainty can guide fallback strategies. When confidence in the learned correction is low, the CSLQ-MPC can return to the nominal model. This allows the controller to trade off optimality for robustness and interpretability during distribution shift or in uncertain scenarios.

The CRN architecture may be easily extended to incorporate additional sensor modalities and semantic information. By conditioning on additional sensor modalities and semantic information, the model may learn to reason about terrain-dependent dynamics, potentially improving performance in unstructured environments.

Beyond control, the CRN can also inform planning by serving as a learned forward dynamics model. This would allow a system to evaluate and select kinodynamically feasible trajectories and make decisions through unstructured environments, accounting for terrain, robot limitations, and complex interactions that might be difficult to capture analytically.

Bibliography

[1] Bill Triggs.
Motion Planning for Nonholonomic Vehicles: An Introduction.
June 1993.

[2] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola.
Dive into Deep Learning, August 2023.
arXiv:2106.11342 [cs].

[3] Yugang Liu and Goldie Nejat.
Robotic Urban Search and Rescue: A Survey from the Control Perspective.
Journal of Intelligent & Robotic Systems, 72(2):147–165, November 2013.
Publisher: Springer Science and Business Media LLC.

[4] Lingli Yu, Hanzhao Wu, Chongliang Liu, and Hao Jiao.
An Optimization-Based Motion Planner for Car-like Logistics Robots on Narrow Roads.
Sensors, 22(22):8948, November 2022.
Publisher: MDPI AG.

[5] Roland Siegwart and Illah Reza Nourbakhsh.
Introduction to autonomous mobile robots.
Intelligent robots and autonomous agents. MIT Press, Cambridge, Mass, 2004.

[6] Spyros G. Tzafestas.
Mobile Robot Control and Navigation: A Global Overview.
Journal of Intelligent & Robotic Systems, 91(1):35–58, July 2018.

[7] Julian Berberich, Johannes Köhler, Matthias A. Müller, and Frank Allgöwer.
Linear tracking MPC for nonlinear systems Part I: The model-based case.

IEEE Transactions on Automatic Control, 67(9):4390–4405, September 2022. arXiv:2105.08560 [math].

[8] Jason Kong, Mark Pfeiffer, Georg Schildbach, and Francesco Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 1094–1099, Seoul, South Korea, June 2015. IEEE.

[9] Shuping Chen, Huiyan Chen, and Dan Negrut. Implementation of MPC-Based Path Tracking for Autonomous Vehicles Considering Three Vehicle Dynamics Models with Different Fidelities. *Automotive Innovation*, 3(4):386–399, December 2020.

[10] Edward N. Hartley. A tutorial on model predictive control for spacecraft rendezvous. In *2015 European Control Conference (ECC)*, pages 1355–1361, Linz, Austria, July 2015. IEEE.

[11] James Dallas, Michael P. Cole, Paramsothy Jayakumar, and Tulga Ersal. Terrain Adaptive Trajectory Planning and Tracking on Deformable Terrains. *IEEE Transactions on Vehicular Technology*, 70(11):11255–11268, November 2021.

[12] Shuzhi Sam Ge and Yun J Cui. Dynamic motion planning for mobile robots using potential field method. *Autonomous robots*, 13(3):207–222, 2002.

[13] K. N. Spentzas, I. Alkhazali, and M. Demic. Kinematics of four-wheel-steering vehicles. *Forschung im Ingenieurwesen*, 66(5):0211–0216, May 2001.

[14] R. Pepy, A. Lambert, and H. Mounier. Path Planning using a Dynamic Vehicle Model. In *2006 2nd International Conference on Information & Communication Technologies*, volume 1, pages 781–786, Damascus, Syria, 2006. IEEE.

[15] Samuel Triest, Mateo Guaman Castro, Parv Maheshwari, Matthew Sivaprakasam, Wenshan Wang, and Sebastian Scherer. Learning risk-aware costmaps via inverse reinforcement learning for off-road navigation. *arXiv preprint arXiv:2302.00134*, 2023.

[16] Sean J Wang, Samuel Triest, Wenshan Wang, Sebastian Scherer, and Aaron M Johnson.
Rough Terrain Navigation Using Divergence Constrained Model-Based Reinforcement Learning.

[17] Marc Peter Deisenroth and Carl Edward Rasmussen.
PILCO: A Model-Based and Data-Efficient Approach to Policy Search.

[18] Aniket Datar, Chenhui Pan, Mohammad Nazeri, Anuj Pokhrel, and Xuesu Xiao.
Terrain-Attentive Learning for Efficient 6-DoF Kinodynamic Modeling on Vertically Challenging Terrain, March 2024.
arXiv:2403.16419 [cs].

[19] Aniket Datar, Chenhui Pan, and Xuesu Xiao.
Learning to Model and Plan for Wheeled Mobility on Vertically Challenging Terrain, September 2023.
arXiv:2306.11611 [cs].

[20] Jean-Francois Tremblay, Travis Manderson, Aurelio Noca, Gregory Dudek, and David Meger.
Multimodal dynamics modeling for off-road autonomous vehicles.
In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1796–1802, Xi'an, China, May 2021. IEEE.

[21] Christopher D. McKinnon and Angela P. Schoellig.
Learning Probabilistic Models for Safe Predictive Control in Unknown Environments.
In *2019 18th European Control Conference (ECC)*, pages 2472–2479, Naples, Italy, June 2019. IEEE.

[22] 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, 8(4):2397–2404, April 2023.
arXiv:2203.07747 [cs].

[23] Ananye Agarwal, Ashish Kumar, Jitendra Malik, and Deepak Pathak.
Legged locomotion in challenging terrains using egocentric vision.
In *Conference on robot learning*, pages 403–415. PMLR, 2023.

[24] Mrinal Kalakrishnan, Jonas Buchli, Peter Pastor, Michael Mistry, and Stefan Schaal.
Learning, planning, and control for quadruped locomotion over challenging terrain.
The International Journal of Robotics Research, 30(2):236–258, 2011.
_eprint: https://doi.org/10.1177/0278364910388677.

[25] Ruihan Yang, Minghao Zhang, Nicklas Hansen, Huazhe Xu, and Xiaolong Wang.
Learning Vision-Guided Quadrupedal Locomotion End-to-End with Cross-Modal Transformers.
October 2021.

[26] Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine.
Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning, December 2017.
arXiv:1708.02596 [cs].

[27] Kevin M. Lynch and Frank C. Park.
Modern Robotics: Mechanics, Planning, and Control.
Cambridge University Press, 1 edition, May 2017.

[28] Rajesh Rajamani.
Vehicle Dynamics and Control.
Mechanical Engineering Series. Springer US, Boston, MA, 2012.

[29] Altay Zhakatayev, Bexultan Rakhim, Olzhas Adiyatov, Almaskhan Baimyshev, and Huseyin Atakan Varol.
Successive linearization based model predictive control of variable stiffness actuated robots.
In *2017 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, pages 1774–1779, Munich, Germany, July 2017. IEEE.

[30] Robin M. Schmidt.
Recurrent Neural Networks (RNNs): A gentle Introduction and Overview, November 2019.
arXiv:1912.05911 [cs].

[31] Sepp Hochreiter and Jürgen Schmidhuber.
Long Short-Term Memory.
Neural Computation, 9(8):1735–1780, November 1997.
Publisher: MIT Press.

[32] Karl J. Åström and Richard M. Murray.
Feedback systems: an introduction for scientists and engineers.
Princeton University Press, Princeton, 2008.

[33] Michael Lemmon.
Nonlinear Control Systems (EE60580), 2017.

[34] A. De Luca, G. Oriolo, and C. Samson.
Feedback control of a nonholonomic car-like robot.
In M. Thoma and J. P. Laumond, editors, *Robot Motion Planning and Control*, volume 229, pages 171–253. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.
Series Title: Lecture Notes in Control and Information Sciences.

[35] Qiang Ge, Shengbo Eben Li, Qi Sun, and Sifa Zheng.
Numerically Stable Dynamic Bicycle Model for Discrete-time Control, November 2020.
arXiv:2011.09612 [eess].

[36] Philip Polack, Florent Altche, Brigitte d'Andrea Novel, and Arnaud De La Fortelle.
The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?
In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 812–818, Los Angeles, CA, USA, June 2017. IEEE.

[37] Raphael Dyrska and Martin Monnigmann.
Properties of Nonlinear MPC Solutions Illustrated with a Simple Example.
IFAC-PapersOnLine, 58(18):41–46, 2024.

[38] J.C. Butcher.
A history of Runge-Kutta methods.
Applied Numerical Mathematics, 20(3):247–260, March 1996.

[39] Manfred Morari and Jay H. Lee.
Model predictive control: past, present and future.
Computers & Chemical Engineering, 23(4-5):667–682, May 1999.

[40] James Blake Rawlings, David Q. Mayne, and Moritz Diehl.
Model predictive control: theory, computation, and design.
Nob Hill Publishing, Madison, Wisconsin, 2nd edition edition, 2017.

[41] Huibert Kwakernaak and Raphael Sivan.
Linear optimal control systems.
Wiley-Interscience, New York, 1972.

[42] Frederic Holzmann, Mario Bellino, Roland Siegwart, and Heiner Bubb.
Predictive estimation of the road-tire friction coefficient.
In *2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control*, pages 885–890, Munich, Germany, October 2006. IEEE.

[43] Sean J Wang, Honghao Zhu, and Aaron M Johnson.
Pay attention to how you drive: Safe and adaptive model-based reinforcement learning for off-road driving.
In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 16954–16960. IEEE, 2024.

[44] Sean J. Wang, Si Heng Teng, and Aaron M. Johnson.
Robust and adaptive rough terrain navigation through training in varied simulated dynamics.
In *ICRA Workshop on Resilient Off-road Autonomy*, 2024.

[45] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson.
Learning Latent Dynamics for Planning from Pixels, June 2019.
arXiv:1811.04551 [cs, stat].

[46] Wenhao Yu, Jie Tan, C. Karen Liu, and Greg Turk.
Preparing for the Unknown: Learning a Universal Policy with Online System Identification, May 2017.
arXiv:1702.02453 [cs].

[47] Selim Engin and Volkan Isler.
Neural Optimal Control using Learned System Dynamics, February 2023.
arXiv:2302.09846 [cs].

[48] Michael Kaup, Cornelius Wolff, Hyerim Hwang, Julius Mayer, and Elia Bruni.
A Review of Nine Physics Engines for Reinforcement Learning Research, August 2024.

arXiv:2407.08590 [cs].

[49] Erwin Coumans.
Bullet 2.83 Physics SDK Manual.
2012.

[50] Dongho Kang Jemin Hwangbo.
SimBenchmark, 2018.

[51] Guillem Torrente, Elia Kaufmann, Philipp Foehn, and Davide Scaramuzza.
Data-Driven MPC for Quadrotors, March 2021.
arXiv:2102.05773 [cs].

[52] Fast Direct Multiple Shooting Algorithms for Optimal Robot Control.
In *Lecture Notes in Control and Information Sciences*, pages 65–93. Springer Berlin Heidelberg, Berlin, Heidelberg, June 2009.

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

[54] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin.
Attention is all you need, 2023.

[55] Robert Bridson.
Fast poisson disk sampling in arbitrary dimensions.
SIGGRAPH sketches, 10(1):1, 2007.