Toward the Control of Non-Linear, Non-Minimum Phase Systems via Feedback Linearization and Reinforcement Learning

Michael Estrada

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-54
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-54.html

May 12, 2021

Acknowledgement

I would like to thank my adviser, S. Shankar Sastry for his guidance and mentorship in this work despite the unprecedented times in which it took place, and Ruzena Bajcsy, for her gracious support as both a mentor and a housemate. I would further like to thank Tyler Westebroek, a perpetual collaborator on this work and an overall excellent person. Finally, I would like to thank my parents, siblings, aunts, and uncles who have supported and encouraged me every step of the way.

Toward the Control of Non-Linear, Non-Minimum Phase Systems via Feedback Linearization and Reinforcement Learning

by

Michael Estrada

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor S. Shankar Sastry, Chair
Professor Ruzena Bajscy

Spring 2021

Toward the Control of Non-Linear, Non-Minimum Phase Systems via Feedback Linearization and Reinforcement Learning

Abstract

Toward the Control of Non-Linear, Non-Minimum Phase Systems via Feedback Linearization and Reinforcement Learning

by

Michael Estrada

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor S. Shankar Sastry, Chair

The control of non-linear systems has historically been a difficult task due primarily to the need for additional analysis of the specific system being controlled in absence of the standard techniques of linear systems theory. While Feedback Linearization can be employed to leverage the techniques of linear systems theory for non-linear systems, error in the dynamic model used to construct the controller can lead to erroneous behavior. Similarly, in the case of a non-minimum phase system, even an exactly accurate linearizaing controller can render the zero dynamics of the system unstable; A condition that must be actively considered and accounted for in the design of the controller and path to be followed. In this work, we present three example cases of non-linear, and later, non-minimum phase systems. The first, strictly non-linear example is that of racing a car. The technique of Learning Feedback Linearization is used to learn an exactly linearizing controller for a 2D racecar, represented by a dynamically extended bicycle model. A preprocessing network is further posited to convert the outputs of the linearizing controller to a seperate set of inputs for the racecar. The latter two examples, both demonstrating non-minimum phase behavior, are those of Planar Vertical Take-Off and Landing in aircraft, and the control of a two-wheeled bicycle. In both cases, these problems are formulated and posed as Reinforcement Learning problems. The formulation of our solution for the latter examples attempts to learn how to correct a nominal path, provided as input to the algorithm, in an effort to maintain the stability of the zero dynamics. Our progress towards these goals is reported; some demonstrative example cases of all three systems are shown, and the progress and results of these experiments are discussed in depth. Finally, possible avenues of future work are discussed.

Contents

List of Figures

Acknowledgments

I would like to thank my adviser, S. Shankar Sastry for his guidance and mentorship in this work despite the unprecedented times in which it took place, and Ruzena Bajcsy, for her gracious support as both a mentor and a housemate. I would further like to thank Tyler Westebroek, a perpetual collaborator on this work and an overall excellent person. Finally, I would like to thank my parents, siblings, aunts, and uncles who have supported and encouraged me every step of the way.

Chapter 1

Preliminaries

1.1 Introduction

The control of nonlinear systems is traditionally considered challenging because of the many, and varied classes of nonlinear systems that each display unique behaviors. typically this needs to be accounted for on a case-by-case basis. The field of geometric control seeks to address this by taking advantage of the structure of the system's dynamics to simplify related tasks like trajectory tracking and planning [1, 2]. The fundamental issue with this approach is the necessity for *accurate* knowledge of the system's dynamics, which can involve a great deal of system identification work or otherwise simplifying the model of the dynamics. This is in contrast to the techniques of model free reinforcement learning, [3–5] which can be used to automatically compute optimal control strategies with no prior knowledge of the system's dynamics. Recent work in this direction has shown some promise but remains difficult in many cases due to poor sample complexity in many robotics applications [6–8]. In this paper, we first seek to demonstrate the technique of Learning Feedback Linearization (LFBL), first proposed in [9]. LFBL is designed to exploit the advantages of both geometric control and model-free reinforcement learning to optimize a geometric controller for a plant with unknown dynamics and limited structural assumptions about those dynamics. Specifically, it seeks to learn an optimal implementation of a technique called *Feedback Linearization* (FBL). A successful feedback linearization scheme has the extremely useful property of rendering the input-output relationship of an otherwise *nonlinear* plant to be *linear*. This then enables the use of path planning and control techniques from the much more broadly applicable, and easily applied, field of linear systems theory [10–12]. While this technique has been used extensively in robotics, the fundamental concern with feedback linearization is the necessity for highly accurate models of the dynamics governing the robot [1, 2, 13–15]. The field of adaptive control has seen extensive work on the construction of exactly linearizing controllers [16–23], but this too has the limitation of needing highly structured representations of how and where the nonlinearities arise in the system's dynamics. The method of LFBL, in constrast, requires relatively limited knowledge about the structure of a nonlinear system to exactly linearize it.

Beyond briefly presenting the method of FBL and extending it to LFBL, this paper is aimed at applying the method for the task of controlling 4-wheeled motor vehicles. Specifically, we want to employ this method to exploit the benefits of linear control in the development of algorithms for *racing* cars. The nonlinear and nonholonomic nature of a car's dynamics, paired with their prevalence in our daily lives, makes the study of their efficient and effective control a tantalizing pursuit. A great deal of work has been done to accommodate the idiosyncrasies of path planning and control for vehicles, and autonomous racing has become a point of interest for profit, sport, and hobby alike [24–27]. There is a mature body of literature on the design of Model Predictive Control (MPC) algorithms that seeks to create optimal paths based on an explicitly stated sets of dynamics and system constraints. There has also been some recent work towards learning various elements of MPC [28–30], With this background in mind, we seek to use LFBL to learn the dynamics of a car. The resulting linear input-output relationship can be used to vastly simplify its planning and control. We demonstrate this by the construction of a simple path planner that uses a Linear Quadratic Regulator (LQR) structure paired with linear constraints to enforce the dynamics of the vehicle. In addition, we posit a novel extension to the method of LFBL; a preprocessing network that is designed to convert the values calculated by an exactly linearizing controller into an appropriate set of inputs for the system we wish to control. This conversion then allows for the application of a feedback linearizing controller to systems that do not share the same inputs as the linearizing controller.

Beyond racing cars, a particular interesting subset of non-linear systems are the class of non-minimum phase systems. The analysis and control of these systems is complicated by the synthesis of a set of dynamics that must be found through the construction of a diffeomorphism. This process of constructing the so-called "eta dynamics", or "left-over dynamics", renders some modes of the system unobservable, and worse-yet, if these modes are unstable with respect to the eigenvalues of the system, then those modes may act erroneously, resulting in non-minimum phase behavior [1, 31]. From a theoretical standpoint, some progress has been made towards adequately controlling this class of systems through stable inversion techniques [32, 33], however, these types of approaches can lead to issues in causality of solution, computational time, and being reliant on the addition of a feedback stabilizing controller. Thus, it is often the case that a particular non-minimum phase system must have special consideration given to it's particular intricacies.
Two particularly useful examples of non-minimum phase systems to study are Planar Vertical Take Off and Landing (PVTOL), and the control of a two wheeled bicycle. The former has been studied in depth for it's application to aircraft that must get into the air with little or no room for a traditional runway, such as in the case of the harrier jet. A variety of approaches have been used to solve this problem, each with it's own set of trade-offs [34–36]. The latter example has also been studied in depth, with a wide variety of approaches in simulation or even on custom-built hardware [37–40]. There has also been a respectable literature developing around the use of machine learning to solve the problem of bicycle control in some volume [41–43]. The difficulty with these approaches is that they often

simplify the dynamics, and as a consequence, the non-minimum phase behavior inherent in a bicycle is mitigated, or not considered at all. On the other hand, some approaches use multi-body dynamics simulations that don't admit any sort of explicit dynamic model for the system, making analysis of the proposed controller difficult beyond empirical success or failure [44–46]. To rectify these concerns, we propose a Reinforcement Learning approach to the control of both PVTOL and bicycle systems. Specifically, we opt to learn how to construct a path for non-minimum phase systems that will simultaneously adhere as closely to a nominal desired path while attempting to maintain the zero dynamics of the system within a predetermined desirable bound.

The remainder of this paper is structured as follows: Chapter 1: Preliminaries, will continue to develop necessary technical and intuitive background for the work here-in. Chapter 2: Non-Linear Vehicle Control will address the specifics of the vehicle dynamics under consideration, briefly develop the method of LFBL, and present our progress towards the efficacious control of a race car. Chapter 3: Non-Minimum Phase Control, will address the dynamics of both PVTOL and bicycle systems, formulate their control as a Reinforcement Learning problem, and report our progress towards the their effective control. Finally, Chapter 4: Concluding Remarks, will briefly summarize the progress of the methods and techniques as they are presented here, with an eye for how the current approach may be improved and some promising possible directions of future work.

1.2 Feedback Linearization

To begin, we consider a system in the control affine form:

$$\begin{aligned} \dot{x} &= f(x) + g(x)u \\ y &= h(x), \end{aligned} \tag{1.1}$$

The method presented here can address the "square" case of this form where the number of inputs matches the number of outputs. $x \in \mathbb{R}^n$ is the state, $u \in \mathbb{R}^q$ is the input, and $y \in \mathbb{R}^q$ is the output with the mappings $f : \mathbb{R}^n \to \mathbb{R}^n$, $g : \mathbb{R}^n \to \mathbb{R}^{nxq}$, and $h : \mathbb{R}^n \to \mathbb{R}^q$ are assumed smooth.

For the purposes of intuition, we will primarily address the single-input-single-output (SISO; q=1) case with the knowledge that the Multi-input Multi-output (MIMO) square case abides the same logic, albeit more mechanically obtuse. In the SISO case, we begin by taking time derivatives of $y = h(x)$ until the input u appears and invert this relationship so-as to enforce the desired linear input-output behavior. Consider the first derivative of the output posed in (1.1):

$$\begin{aligned} \dot{y} &= \frac{dh}{dx}(x) \cdot \Big(f(x) + g(x)u\Big) \\ &= \frac{dh}{dx}(x) \cdot f(x) + \frac{dh}{dx}(x) \cdot g(x)u \end{aligned}$$

where $L_f h(x) = \frac{dh}{dx}(x) \cdot f(x)$ and $L_g h(x) = \frac{dh}{dx}(x) \cdot g(x)$ are called Lie derivatives and serve to quantify the rate of change of $y = h(x)$ along vector fields defined by f, and g respectively. In the case where $L_g h(x) \neq 0$, u appears in the derivative and we can exactly control the outputs $y = h(x)$ with a controller of the form:

$$u(x, v) = \frac{1}{L_g h(x)}(-L_f h(x) + v) \tag{1.2}$$

where $v \in R^q$ is the set of virtual inputs. This transformation between u and v serves to "cancel out" the nonlinearities of the original system and enforce the relationship $\dot{y} = v$. If, however, $L_g h(x) \equiv 0$, then the input can't directly affect the first derivative as desired and the control law (1.2) is not well defined. In this case, we can continue taking derivatives of $y(x)$ until the input appears. When the input appears in the γ^{th} derivative, the derivative can be expressed as:

$$y^{(\gamma)} = L_f^{\gamma} h(x) + L_g L_f^{\gamma-1} h(x) u \tag{1.3}$$

where $L_f^{\gamma} h(x)$ and $L_g L_f^{\gamma-1} h(x)$ are higher order Lie Derivatives. As before, if $L_g L_f^{\gamma-1} h(x) \neq 0$ then the control law can be rewritten as:

$$u(x, v) = \frac{1}{L_g L_f^{\gamma-1} h(x)}(-L_f^{\gamma} h(x) + v) \tag{1.4}$$

As above, this choice of control law enforces the relationship $y^{\gamma} = v$, where gamma is called the "relative degree" of the nonlinear system.

This process can be extended readily to square, MIMO systems $[q > 1]$. By successively taking the derivatives of each output variable until at least one of the inputs appears, we can obtain an input-output relationship of the form:

$$[y_1^{\gamma_1}, ..., y_p^{\gamma_q}]^T = b(x) + A(x)u \tag{1.5}$$

where the square matrix $A(x) \in \mathbb{R}^{q \times q}$ is called the "decoupling matrix" and $b(x) \in \mathbb{R}^q$ is called the "drift term". If $A(x)$ is invertable, then the control law can again be written as:

$$u(x, v) = A^{-1}(x)(-b(x) + v) \tag{1.6}$$

with virtual inputs $v \in \mathbb{R}^q$ corresponding to the decoupled linear system in (1.6):

$$[y_1^{\gamma_1}, ..., y_q^{\gamma_q}]^T = [v_1, ..., v_q]^T \tag{1.7}$$

where the j^{th} entry of v is denoted v_j, $y_j^{\gamma_j}$ is the γ^{th} derivative of the j^{th} output, and $\gamma = (\gamma_1, ..., \gamma_q)$ is called the "relative degree" of the system. The system (1.7) is Linear Time invariant (LTI) and can be represented in the "normal form": with

$$\dot{\xi}_r = A'\xi_r + B'v_r \tag{1.8}$$

With state variables $\xi_r = (y_1, \dot{y_1}, ..., y_1^{\gamma_1 - 1}, ..., y_q, ..., y_q^{\gamma_q - 1})$ and $A' \in \mathbb{R}^{\gamma \times \gamma}$ and $B' \in \mathbb{R}^{\gamma \times q}$. This simplified normal form can be used to more easily create a desired trajectory for the output of the nonlinear system using the techniques of linear control theory. A key insight from this analysis is the manner in which the transformed LTI system is made to behave. Specifically, the normal form is formulated in such a way that the γ_j^{th} derivative of the j^{th} output is the value being directly controlled while the actual output, y_j is related to this value by a chain of integrators, e.g. we directly control the acceleration of a vehicle to achieve the desired position values.

1.3 Non-Minimum Phase Dynamics

Continuing our analysis specifically for the SISO case, we can address the special case of a feedback linearized system that displays non-minimum phase behavior. It is often the case that a feedback linearized system will have relative degree $\gamma \leq n$, resulting in a system of γ linearly independent equations at some point x^o. We would like to complete this system of equations to encompass n equations, the size of the nominal system state. In particular, we can generalize our previous understanding of the normal form to encompass these added equations:

$$\begin{aligned}
&\dot{\xi_1} = \xi_2 \\
&\vdots \\
&\dot{\xi_\gamma} = p(\xi, \eta) \\
&\dot{\eta}_1(x) = q_1(\xi, \eta) \\
&\vdots \\
&\dot{\eta}_{n-\gamma}(x) = q_{n-\gamma}(\xi, \eta)
\end{aligned} \tag{1.9}$$

Where the $\eta_1, \eta_2, ..., \eta_{n-\gamma}$ terms are the "eta dynamics" of the system. A, typically non-unique, set of eta dynamics can be found through the construction of a diffeomorphism. A map $\Phi : R^n \to R^n$ is called a diffeomorphism if it is a one-to-one smooth map with a smooth inverse. Further, the map $\Phi : U \to U$ is a diffeomorphism if its Jacobian $D\Phi$ has full rank. So, by the construction of a map with the appropriate Jacobian, a corresponding diffeomorphism can be found that outputs the correct change of coordinates. An unfortunate side-effect of this transformation is that it renders the eta dynamics unobservable, further complicating their analysis.

Next, we most consider the eta dynamics under a condition of output zeroing. As the name would suggest, we seek to find an input that regulates the outputs being controlled to zero. This task is in-and-of-itself trivial in light of 1.7, but can present some interesting residual complications in the case of non-minimum phase systems. The process of choosing u to zero the outputs results in the last $n - \gamma$ equations of 1.9 being only a function of the eta states:

$$\begin{aligned}
&\dot{\eta}_1(x) = q_1(0, \eta) \\
&\vdots \\
&\dot{\eta}_{n-\gamma}(x) = q_{n-\gamma}(0, \eta)
\end{aligned} \tag{1.10}$$

These are called the "zero dynamics" of the system. For ease, we will consider the linearization about an equilibrium point at the origin. If the eigenvalues of the system lie strictly in the left-half plane, then the system is said to be locally exponentially minimum phase; if one or more of the eigenvalues lies on the $j\omega$ axis, then the system may only be asymptotically minimum phase. Finally, if any of the eigenvalues of the linearization lie in the right-half plane, then the system is said to be non-minimum phase. Effectively, this analysis amounts to the question of stability in the eta dynamics. If, in the process of regulating the controlled outputs to zero, the residual eta dynamics are stable with respect to the systems eigenvalues, then the eta dynamics will pose no issues. If, however, the eta dynamics are not stable, then the system may act erroneously in a manner that is not conducive to safe operation. Some specific examples of non-minimum phase systems are discussed in chapter 3 to provide more concrete insight into the possible difficulties of non-minimum phase systems, and how we hope to address them.

1.4 Optimal Control and Planning

The transformation of the previously nonlinear dynamics into a linear system of the form (1.8) enables the use of a wide range of possible linear control and path planning techniques that present greater opportunity for guarantees in performance and safety. Central to the path tracking approach used in this work is the continuous-time LQR:

$$\min_{v} \quad \int_0^\infty (x^T Q x + v^T R v) \tag{1.11}$$

Where $x \in \mathbb{R}^\gamma$ is the system state, $v \in \mathbb{R}^q$ is the input and $Q \in \mathbb{R}^{\gamma \times \gamma}$ and $R \in \mathbb{R}^{q \times q}$ are weight matrices determined by the designer. The solution of this optimization problem can be readily found by solving the continuous-time algebraic ricatti equation:

$$A^T P + PA - PBR^{-1}B^T P + Q = 0 \tag{1.12}$$

Solving this equation for a unique, positive definite solution, P, provides sufficient information to construct an optimal feedback control law:

$$v^* = Fx = R^{-1}B^T Px \tag{1.13}$$

The ease of solution of equation (1.12), paired with the simple control law (1.13) and guarantees of optimality in control make this ideal as a feedback control strategy to regulate the car to the commands of a higher-level path-planner, which we construct as a discrete-time, constrained LQR problem:

$$\begin{aligned} \min_{v} \quad & \textstyle\sum_{k=1}^{N-1}(x^T Q x_k + v_k^T R v_k) \; + \mathrm{x}_N^T Q_f x_N \\ \text{s.t.} \quad & x_{\tau+1} = A' x_\tau + B' v_\tau \\ & x(T) = x_f \\ & x(0) = x_0 \\ & |v(k)|_\infty \leq v_{bnds} \end{aligned} \tag{1.14}$$

Here, N is the number of execution steps in the desired look-ahead time, T. τ represents the current time-step, x_f and x_0 represent the desired final and initial locations of the system's *linear* state respectively. x_τ and v_τ represent the linear system state and input, respectively, at time-step τ. Finally, $Q \in \mathbb{R}^{\gamma\times\gamma}$, $Q_f \in \mathbb{R}^{\gamma\times\gamma}$, and $R \in \mathbb{R}^{q\times q}$ are weight matrices determined by the designer. Further,

$$\begin{aligned} v &= [v_1, .., v_{N-1}] \\ x &= [x_1, ..., x_N] \end{aligned} \tag{1.15}$$

This configuration is ideal for multiple reasons. While the unconstrained version of (1.14) can be solved readily using a ricatti equation similar to (1.12), we choose to formulate this system with the additional constraints of a starting and ending location, Other constraints are added to enforce mechanical limitations like only having the ability to accelerate so quickly. This effectively forces the planner to account for the boundary conditions of the car's starting location, mechanical limitations, and where it should be at the end of its look-ahead period. The addition of constraints to enforce waypoint tracking between the start and finish locations can be used to further constrain the planner to a desired path. In the exactly linearizing case of chapter 2, both the state path and inputs that we calculate in this approach can be used limiting the need for the continuous time LQR solution (1.13), However, in the approximate controller case, we will need a more active form of feedback control to correct for the unknown true dynamics of the system. It is convenient to note that in both cases, (1.14) a linearly constrained quadratic programming problem that is easily solvable by most off-the-shelf solvers quickly and thus is ideal for a waypoint following-based path planning approach. Specifically, this formulation can be iteratively solved as the system evolves to update the planned path as the vehicle successively approaches and passes new waypoints, akin to an MPC approach.

1.5 Reinforcement Learning

While we construct our control approach in this work to be amenable to any common Reinforcement Learning algorithm, we have opted throughout this work to use an implementation of Soft Actor-Critic (SAC). a state-of-the-art model-free deep RL algorithm. SAC is an off-policy algorithm that optimizes over a stochastic policy with entropy regularization, which augments its objective to be

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t,a_t) + \alpha H(\pi(\cdot|s_t))] \tag{1.16}$$

where $H(\pi(\cdot|s_t))$ is the entropy term that measures the randomness of the policy under s_t and α is a temperature hyper-parameter that determines the relative importance of the entropy [47]. Detailed implementations and novelty designs [48] of SAC will not be mentioned in this work. We have opted to use SAC for a variety of reasons, including it's success in the related work posed by [9], it's general performance, and overall desirable sample complexity properties.

Chapter 2

Non-Linear Vehicle Control

2.1 Vehicle Dynamics and Feedback Linearization

For the purposes of this work, we address only the lateral dynamics of a vehicle, or, the dynamics from a "top down" view. The intricacies of a vehicle's suspension, the possibility of tire slip, and the dynamics of the car's drive train will ideally be learned by the preprocessing network posited in section 2.2. With this in mind, we consider a dynamically extended bicycle model of the vehicle dynamics:

$$\begin{aligned} \dot{x} &= V cos(\psi + \beta) \\ \dot{y} &= V sin(\psi + \beta) \\ \dot{\psi} &= \tfrac{V}{l_r} sin(\beta) \\ \dot{V} &= a \\ \dot{\beta} &= b \end{aligned} \tag{2.1}$$

Where x and y are the coordinates of the vehicle's center of mass in an inertial frame (X, Y), ψ is the inertial heading of the car, V is the velocity of the car, l_r is the distance from the center of mass to the vehicle's rear axle, and β is the angle of the velocity of the center of mass with respect to the longitudinal axis of the car. Here, we choose x and y as the outputs of the system that we seek to control and further choose a and b as the inputs through which we will do so. A visualization of this system is provided in figure 2.1. A first-principles analysis of this system reveals further:

$$\beta = tan^{-1}(\frac{l_r}{l_f + l_r}(tan(\delta_f)) \tag{2.2}$$

where δ_f is the angle of steering of the front tires and l_f is the distance from the center of mass to the vehicle's front axle. While δ_f is more intuitively meaningful as a choice of input, the complicated nature of its relationship to the vehicle's dynamics make it preferable for our formulation to control $\dot{\beta}$ directly and back-calculate δ_f uniquely in the case of a steering angle limited to at most ± 90 degrees from a neutral orientation. This easily covers the range of any standard vehicle's max steering angle.

With this form of the dynamics, we can start deriving the system's linearized normal form. We first take derivatives of the outputs with respect to time until the inputs appear:

$$\begin{aligned}
&x = x \\
&\dot{x} = V cos(\psi + \beta) \\
&\ddot{x} = \dot{V} cos(\psi + \beta) - \dot{\beta} V sin(\psi + \beta) - \tfrac{V}{l_r} sin(\psi + \beta) \\
&\ddot{x} = cos(\psi + \beta) a - V sin(\psi + \beta) b - \tfrac{V}{l_r} sin(\psi + \beta)
\end{aligned} \tag{2.3}$$

similarly, the derivatives of y can be found:

$$\begin{aligned}
&y = y \\
&\dot{y} = V sin(\psi + \beta) \\
&\ddot{y} = \dot{V} sin(\psi + \beta) + \dot{\beta} V cos(\psi + \beta) + \tfrac{V}{l_r} sin(\psi + \beta) \\
&\ddot{y} = sin(\psi + \beta) a + V cos(\psi + \beta) b + \tfrac{V}{l_r} sin(\psi + \beta)
\end{aligned} \tag{2.4}$$

With this, our vector relative degree is:

$$\begin{bmatrix} \gamma_1 \\ \gamma_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} \tag{2.5}$$

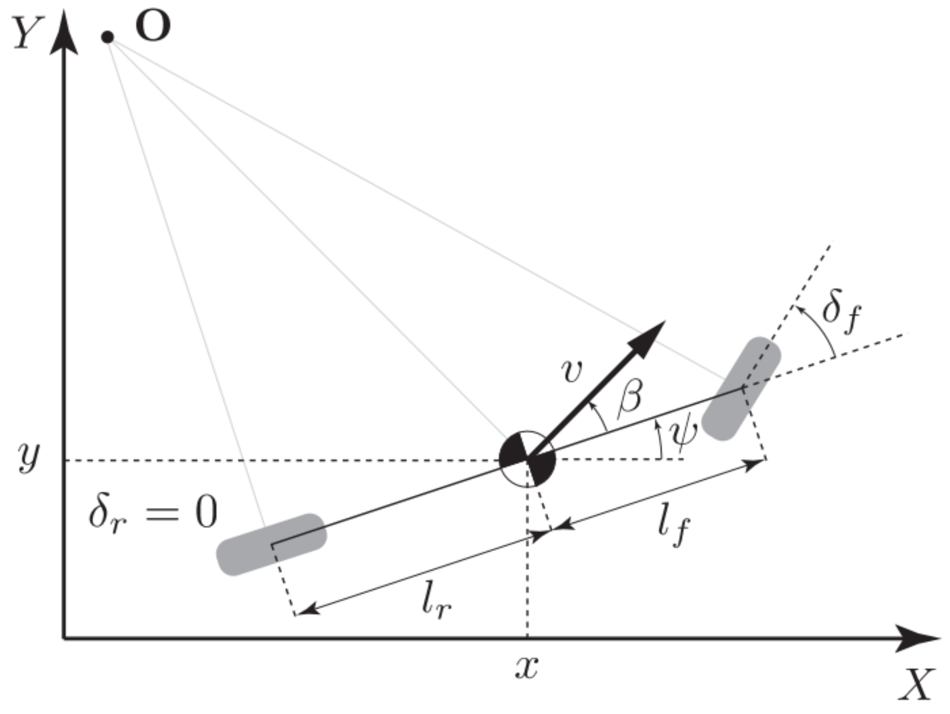

Figure 2.1: Visual representation of kinematic bicycle model

By applying the transformation $x = \xi_1^1$, and $y = \xi_2^1$, the normal form can be written as:

$$\begin{aligned}
\dot{\xi}_1^1 &= \xi_2^1 \\
\dot{\xi}_2^1 &= \xi_3^1 \\
\dot{\xi}_3^1 &= cos(\psi+\beta)a - Vsin(\psi+\beta)b - \tfrac{V}{l_r}sin(\psi+\beta) \\
\dot{\xi}_1^2 &= \xi_2^2 \\
\dot{\xi}_2^2 &= \xi_3^2 \\
\dot{\xi}_3^2 &= sin(\psi+\beta)a + Vcos(\psi+\beta)b + \tfrac{V}{l_r}sin(\psi+\beta)
\end{aligned} \tag{2.6}$$

with corresponding decoupling matrix:

$$A = \begin{bmatrix} cos(\psi+\beta) & -Vsin(\psi+\beta) \\ sin(\psi+\beta) & Vcos(\psi+\beta) \end{bmatrix} \tag{2.7}$$

and associated drift term:

$$b = \begin{bmatrix} -\tfrac{V}{l_r}sin(\psi+\beta) \\ \tfrac{V}{l_r}sin(\psi+\beta) \end{bmatrix} \tag{2.8}$$

This yields the equations for our linearizing inputs:

$$u = -A^{-1}b + -A^{-1}v \tag{2.9}$$

where v is the virtual input to the desired linear system and u is the corrected input that accounts for the nonlinearities of the system. Finally, we arrive at the linearized dynamics relating the virtual inputs to the outputs:

$$\begin{aligned} \ddot{x} &= v_1 \\ \ddot{y} &= v_2 \end{aligned} \tag{2.10}$$

It is clear from inspection that the appropriate choice of v_1 and v_2 allows us to exactly control the desired output, x and y. The resulting *linear* system as expressed in (1.8) is

$$A' = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, B' = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{2.11}$$

$$C' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.12}$$

2.2 Problem Formulation

Learning Feedback Linearization

The primary shortcoming of feedback linearization as a methodology is it's inherent dependence on *exact* knowledge of the dynamics governing the plant, or, the system that is being controlled. This naturally leads to two sets of dynamics that must be considered in practice, the plant dynamics that represent the true behavior of the system, and the model dynamics designed by the engineer attempting to control the system; consider:

$$\begin{aligned}\dot{x}_p &= f_p(x_p) + g_p(x)u_p \\ y_p &= h_p(x),\end{aligned} \tag{2.13}$$

$$\begin{aligned}\dot{x}_m &= f_m(x_m) + g_m(x)u_m \\ y_m &= h_m(x),\end{aligned} \tag{2.14}$$

Where a subscript p denotes that we are referring to the plant and a subscript m indicates that we are referring to the model. With the technique of Learning Feedback Linearization, we aim to rectify the aforementioned model mismatch by learning how to appropriately deviate the nonlinear input u_p to better match the output of the plant to what we would expect based on the model. In this way, we are not only learning an exactly linearizing controller, but also forcing the plant to behave according to the dynamics designated by our model. For learning feedback linearization to be applicable, we first require that the system(s) (2.13), (2.14) have the same, well-defined relative degree:

$$(\gamma_1^p, ..., \gamma_q^p) = (\gamma_1^m, ..., \gamma_q^m) \tag{2.15}$$

With this assumption, we can guarantee the existence of feedback control laws of the form:

$$u_p(x, v) = \beta_p(x) + \alpha_p(x)v \tag{2.16}$$

$$u_m(x, v) = \beta_m(x) + \alpha_m(x)v \tag{2.17}$$

With $\beta_p(x), \beta_m(x) \in \mathbb{R}^q$, and $\alpha_p(x), \alpha_m(x) \in \mathbb{R}^{q\times q}$. u_p is, as of yet, unknown, but owing to the structure and assumptions reviewed here, we can relate the dynamics of the plant and model as follows:

$$\beta_p(x) = \beta_m(x) + \Delta\beta(x) \tag{2.18}$$

$$\alpha_p(x) = \alpha_m(x) + \Delta\alpha(x) \tag{2.19}$$

Here, $\Delta\beta(x)$ and $\Delta\alpha(x)$ are two continuous functions with parameterized estimates:

$$\Delta\beta(x) \approx \beta_{\theta_1}(x), \Delta\alpha(x) \approx \alpha_{\theta_2}(x) \tag{2.20}$$

Where θ_1, θ_2 represent the parameters to be learned through experimentation of the reinforcement learning agent. It is through learning the approximations (2.20) that the model mismatch is rectified and exact feedback linearization of the dynamic system in question can be achieved with the control law:

$$\hat{u}_\theta(x, v) = [\beta_m(x) + \beta_{\theta_1}] + [\alpha_m(x) + \alpha_{\theta_2}]v \tag{2.21}$$

Formulating the Learning Problem

We know from section 1.2 that the dynamics relating the inputs to the outputs under control law (2.21) are of the form:

$$y^{(\gamma)} = \underbrace{b_p(x) + A_p(x)\hat{u}_\theta(x.v)}_{W_\theta(x,v)} \tag{2.22}$$

Where $b_p(x)$ and $A_p(x)$ are unknowns. We seek to find the parameters θ such that $W_{\theta^*}(x,v) \approx v$, leading to the choice of a point-wise loss $\ell : \mathbb{R}^n \times \mathbb{R}^q \times \mathbb{R}^{K_1+K_2} \to \mathbb{R}$:

$$\ell(x,v,\theta) = ||v - W_{\theta^*}(x,v)||_2^2 \tag{2.23}$$

This relatively simple 2-norm loss measures the learned controller's performance in linearizing the plant dynamics as a function of the control $\hat{u}_\theta$ and the state x when virtual input v is applied to the linear reference model whose behavior we seek to match. Next we can define a probablity distribution X over $\mathbb{R}^n$ with V, the uniform distribution over the set $\{v \in \mathbb{R}^q : ||v|| \leq 1\}$ and define the weighted loss:

$$\mathrm{L}(\theta) = \mathbb{E}_{x\sim X, v\sim V} l(x,v,\theta) \tag{2.24}$$

With this, we chose to formulate our learned controller to optimize over parameters, θ to solve the problem:

$$\min_\theta L(\theta) \tag{2.25}$$

The primary concern in formulating the learning problem like this is the unknown nature of the terms that constitute $W_{\theta^*}(x,v)$, however, this can be circumvented by measuring the outputs of the plant, $y^{(\gamma)}$ as there is an equivalence between the two as defined in (2.22). Further detail on the continuous time formulation of this problem, including the conditions under which global minima can be found, can be found in [9].

Preprocessing Network

The gym environment 'CarRacing-v0' takes in tuples of actions in the form $\{s, g, b\}$, corresponding to the steering, gas, and brake applied respectively. In an end-to-end reinforcement learning task, there exists no need to investigate the detailed effects and properties of individual actions, as the learning algorithm would implicitly learn some representations of the actions, such as a Q functions, through the training process [49].

In this task, however, the gym environment can be modified to return the inputs needed for the LFBL controller, but not to receive it's outputs. That is, the LFBL controller provides commands for acceleration and steering angle, while the racing environment requires inputs in terms of steering angle, gas and brake. This signal mismatch makes it necessary to develop a methodology for converting between the two types of inputs, and motivated the development and training of "preprocessing network" using traditional supervised learning techniques.

Since the relationship between linear acceleration and the required control inputs is reasonably simple, a shallow feed-forward neural network can approximate the relationship well. The input layer has size (1×200) followed by a leakyReLU activation, and the output has size (200×1) with identity output activation. The Adam optimizer is used with a learning rate of 0.05 over the MSE loss.

Overall Control Architecture

To briefly summarize the overall control architecture posited in this paper, we direct the reader to figure 2.2 which is a visual representation of how the path and corresponding control signals are created and fed to the environment. In general, the entire system is guided by a path planner that calculates an optimal path from a desired starting point to a desired end-point, as-in (1.14). Then the LFBL controller can be used to calculate an input to the nonlinear dynamics of the car that will behave as-in the desired linear relationship, (1.7). The output of this controller, acceleration, is then converted into gas and brake signals that are fed directly to the racing environment in tandem with the other output of the linearizing controller, steering angle. The effect of these inputs is observed and used to further train the LFBL controller. Note that, in the case of the exact learned controller desribed below, the preprocessing network is not needed, as the dynamics being learned are strictly of the form of the LFBL controller.

2.3 Experiments, Results, and Discussion

Exact Learned Controller

We first sought to learn a controller that starts with a linearizing controller based on a set of nominal dynamics that can be exactly linearized relative to the true dynamics being learned. The setup for running experiments to this end involves the instantiation of two sets of dynamic equations of the form (2.1). The key difference is the differing choice of l_r. This inherently affects how the inertial heading evolves and has downstream consequences for how the outputs we seek to control evolve. In particular, this mostly effects the drift term. Thus the problem is for the agent to learn a *corrective* drift term that accurately accounts for the difference in l_r and causes the true dynamics to behave *exactly* as the nominal dynamics would dictate. For the tests performed, the nominal dynamics were instantiated with an l_r value of .5, idicating a distance between the front wheels and the center of mass of .5 meters.

With this setup we began training. The LQR planner discussed in section 1.4 was configured to plan a path for the car from a starting location of $[x, y] = [0, 0]$ at $t = 0$ to a

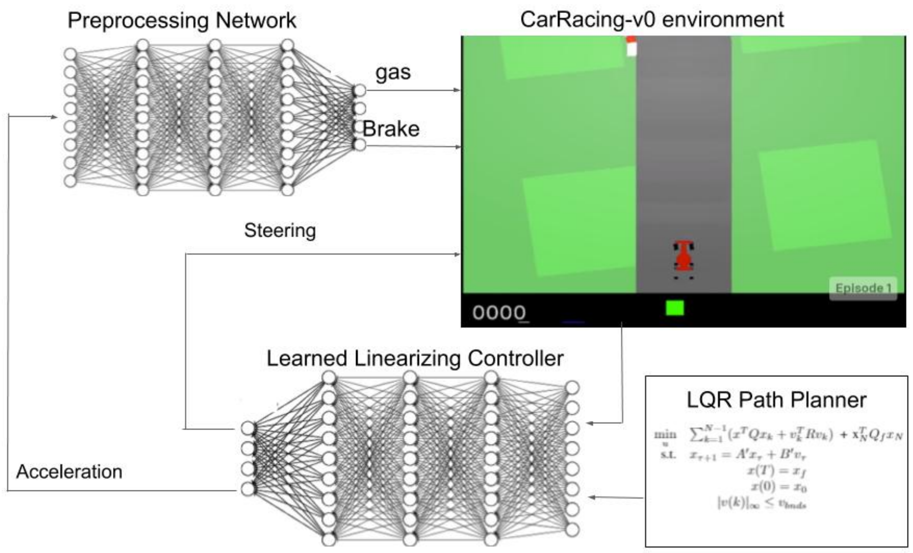
Preprocessing Network
CarRacing-v0 environment
gas
Brake
Steering
Episode 1
0000
Learned Linearizing Controller
LQR Path Planner
$\min_u \sum_{k=1}^{N-1}(x^TQx_k + v_k^TRv_k) + x_N^TQ_fx_N$
s.t. $x_{\tau+1} = A'x_\tau + B'v_\tau$
$x(T) = x_f$
$x(0) = x_0$
$|v(k)|_\infty \leq v_{bnds}$
Acceleration

Figure 2.2: Diagram depicting the overall flow of the proposed control scheme

final location of $[x, y] = [5, 5]$ at $t = 5$s with a time step of .02s, resulting in episodes of 249 steps of the environment. because the car begins aligned with the y axis, this involves turning, which will in turn require that the agent learn to correct for turning as well as acceleration. The step reward is calculated as the 2-norm distance of the vehicle's *linear* state from the desired state calculated by the planner at that time. the sum of these values over an episode is therefore a good indication of the ability of the controller to follow the desired path and is consequently a good proxy for the quality of the learned controller. To begin, we removed the ability of the agent to actively effect the controller in order to set a baseline that reflects the performance of the controller as it was nominally designed with $l_r = .5$. We found that in an episode of 249 steps the total loss for an episode was constant, as there was no way for the agent to interact with, and correct, the controller. The baseline average episode return value we observed was -1.4×10^3. We next tried to run the algorithm using the default parameters of the SAC implementation being used and received a max average episode return of -350 at epoch 56 of 200. This is slightly less than a four-fold reduction, indicating significant improvements in the performance of the learned controller, a plot for the learning curve of this agent, as well as a visualization of the path it followed vs the path it was intended to follow are in figures 2.3 and 2.4, respectively. Once the performance of a naively tuned agent was found, we began a hyperparameter sweep to determine the best performing set of parameters. We opted to run these sweeps over a single variable, keeping the rest of the hyperparameters as the default setting used in the previous experiment. We ran tests varying learning rate, the batch size of training updates, the discount factor, γ, as well as the polyak iterpolation factor, ρ, the result of most of these experiments was the same; within roughly 10 eopchs of starting the experiment, the agent would converge to an Average episode return fluctuating between -700 and -900, although it would occassionally show chaotic behavior where it either performed exceptionally well, as in the trial reported above, or blowing up and returning NaN values after several epochs. This seems to indicate that the performance of the agent is sensitive to the seed used in the training process, which was selected at random for each trial. We further note that these trials were run in roughly one hour per trial on both a personal laptop and desktop.

From inspection of 2.4, a few things are clear. While the desired path is relatively smooth, the path that is followed by the nominal controller with no learning performs quite poorly, diverging from the initial path very quickly, performing chaotic maneuvers and not arriving at the correct final location. This is to be expected as the controller is designed not to match the model that is actually being controlled. More interesting is the path followed by the learned controller, reported above to have an average episode return of -350. It is clear that this controller does not follow the path very closely, makes erroneous maneuvers, and fails to reach the specified endpoint. It does, however, reach roughly the correct y-coordinate, $y = 5$ and manages to stay much closer to the desired path than the the controller with no learning. There might be a few reasons for this lackluster performance. As previously mentioned, this trial appears to be the product of a good initial seed without any special hyperparameter tuning. Thus, performing a second hyperparameter sweep using this initial seed will likely improve performance significantly.

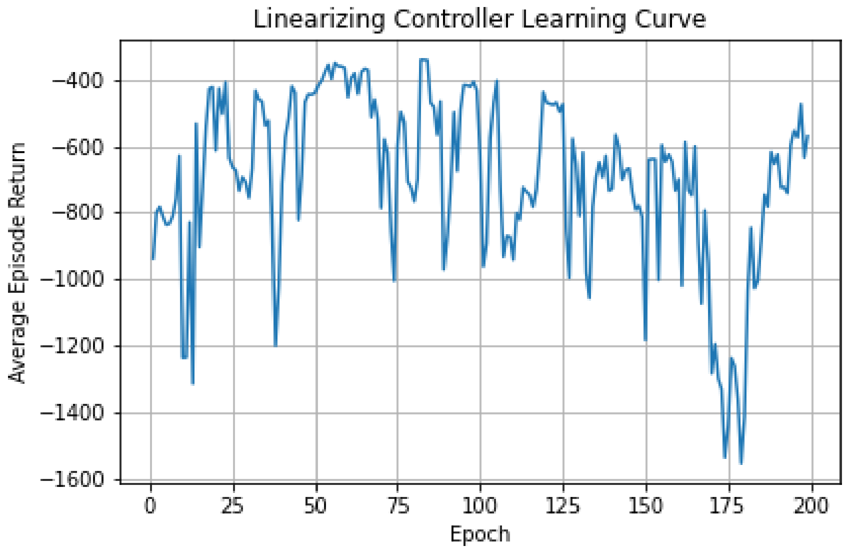

Figure 2.3: Plot of the learning curve for the exactly linearizing agent.

Approximate Learned Controller

The next case to consider is the approximate learned controller. here the true dynamics are modeled by the car racing environment discussed in section III. Beyond simply learning a corrective term for 2.1, this set of true dynamics also models more complicated behaviors relating to the application of brakes, steering, and gas as well as accounting for the delays in their effect on the acceleration and steering that we seek to control. This is where the preprocessing network is used to learn those more complicated relationships and relate them to the acceleration and steering angle that we command from our linearized controller. This involves not only a well-trained linearizing controller, but a well trained preprocessing network. As such, we though it best to decouple the problem of training the controller and preprocessing networks before we could begin experiments to this end in-earnest. While an initial version of the preprocessing network has been trained as per section 2.3, the fact that we are still in the process of training an optimally performant linearizing controller has limited our ability to test the performance of the controller and preprocessing network combined.

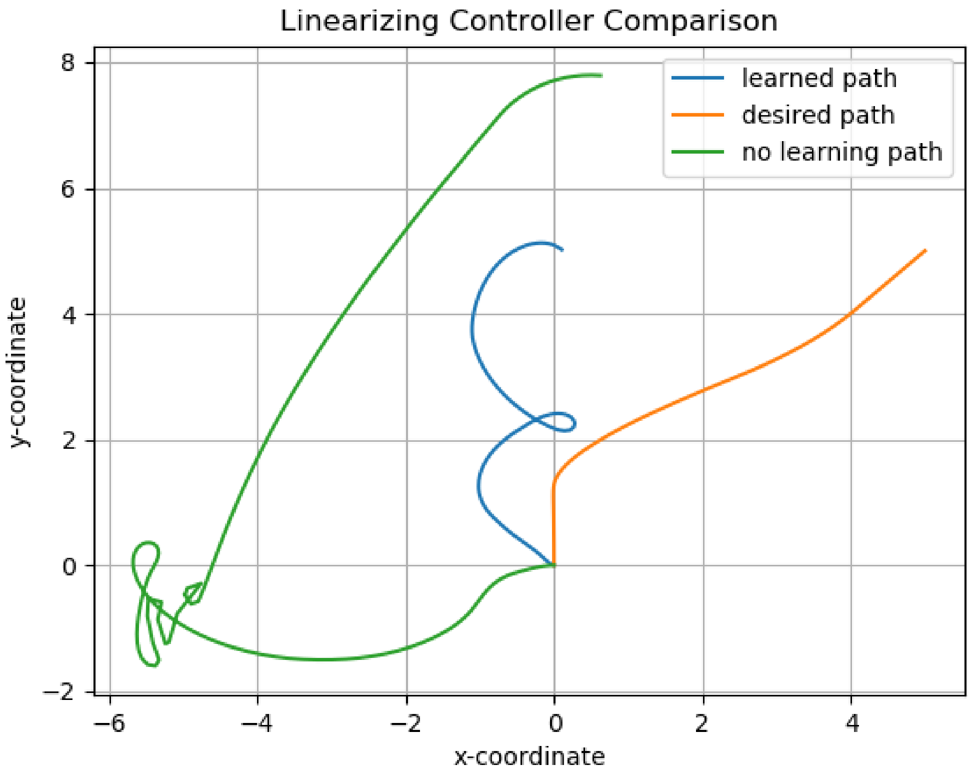

Figure 2.4: Plot of desired path versus learned path versus no learning path

Preprocessing Network Training

The aim of the preprocessing network is to take in the desirable linear acceleration that the feedback linearizing controller outputs, and return the tuple $\{gas, brake\}$ which can be directly sent as a control action to the gym environment.

The training data set is collected by sampling random actions from the ‘CarRacing-v0‘ environment with the steering angle fixed to be zero. Once an action vector is sampled, it is fixed and kept applied to the environment for certain number of steps (usually 10). The acceleration is calculated by taking the average of the differences in velocity between consecutive steps. Our final data set consists of 5000 $\{acceleration, gas, break\}$ observations, where the last two variables serve as the label in the supervised learning task. A plot of the learning curve for the initial preprocessing network implementation is shown in figure 2.5. While it appears to have converged by iteration 20 of training, we have not yet had the opportunity to test it in tandem with the linearizing controller.

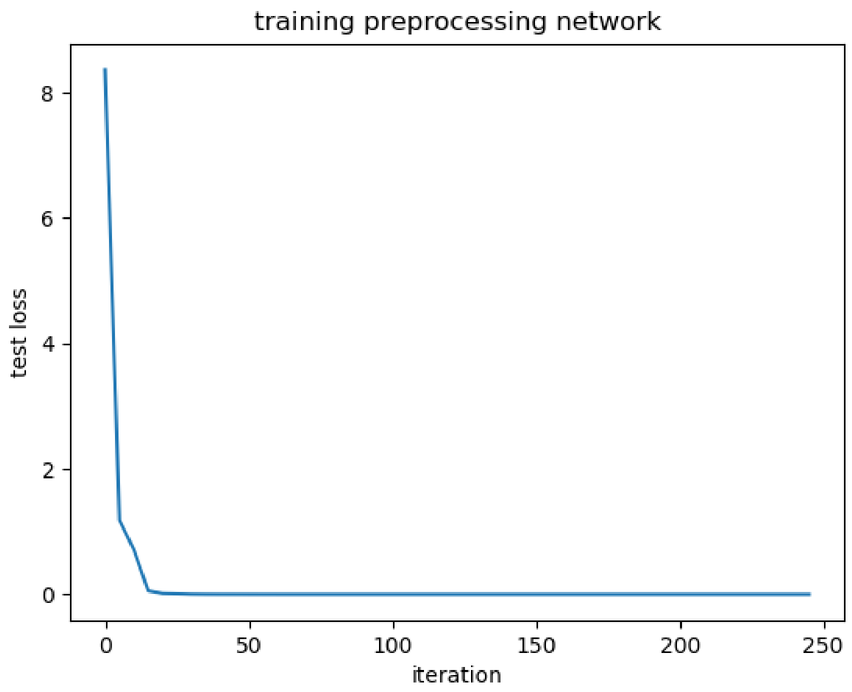

Figure 2.5: Plot depicting the learning curve of the preprocessing network.

Chapter 3

Non-Minimum Phase Control

3.1 Example Dynamics and Feedback Linearization

Planar Vertical Take Off and Landing

Planar Vertical Take Off and Landing (PVTOL) is an oft-studied example of non-minumum phase behavior and consists of an aircraft, assumed to be operating strictly in the 2D plane whose normal vector is defined by the roll axis of the aircraft at the beginning of the trial. The most common real-world example of this can be found in the harrier jet. In general, we choose the state of the plane to be the x and y coordinates of the center of mass, the roll angle, θ, as well as their derivatives, $\dot{x}, \dot{y}, \dot{\theta}$. A visualization of this problem is shown below in [**figure from textbook**]. The nominal nonlinear dynamics of this system can be described as:

$$\begin{aligned} \ddot{x} &= -sin(\theta)u_1 + \epsilon cos(\theta)u_2 \\ \ddot{y} &= cos(\theta)u_1 + \epsilon sin(\theta)u_2 - 1 \\ \ddot{\theta} &= u_2 \end{aligned} \tag{3.1}$$

Here, $\epsilon \geq 0$ intuitively represents the coupling term between the roll moment and lateral acceleration of the aircraft being studied, which is, in turn, determined based on the design of the specific aircraft being studied. The decoupling matrix of this system is:

$$A = \begin{bmatrix} -sin(\theta) & \epsilon cos(\theta) \\ cos(\theta) & \epsilon sin(\theta) \end{bmatrix} \tag{3.2}$$

with associated drift term:

$$b = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \tag{3.3}$$

It is important to note that $det(A) = -\epsilon$, implying that the decoupling matrix is non-singular so long as the strict inequality $\epsilon > 0$ is maintained. With this linearization scheme in mind, the resulting linear system can be described by the following system of equations:

$$\begin{aligned} \ddot{x} &= v_1 \\ \ddot{y} &= v_2 \\ \ddot{\theta} &= \tfrac{1}{\epsilon}(sin(\theta) + cos(\theta)v_1 + sin(\theta)v_2) \end{aligned} \tag{3.4}$$

The resulting *linear* input-output system is of the form we seek, with the residual effect of rendering the θ dynamics unobservable. To address the idiosyncrasies of this, we must analyze the zero dynamics of the system. By choosing $v_1 = v_2 = 0$, we find the resulting zero dynamics to be:

$$\ddot{\theta} = \frac{1}{\epsilon} sin(\theta) \tag{3.5}$$

This corresponds to the motion of an undamped pendulum and indicates that the PVTOL system is non-minimum phase. Intuitively, this indicates that the system, unperturbed by inputs, will naturally rotate back and forth or repeatedly roll over, depending on it's initial conditions. In the case of this work, we assume perfect model knowledge, resulting in only 3.4 being simulated for computational efficiency. Further we strictly abide the condition $\epsilon > 0$ to insure the efficacy of this approach. Further details of this derivation, as well as the intracacies of the $\epsilon = 0$ case can be found in [1]

Bicycle

The case of a two-wheeled bicycle is a particularly challenging example of a non-minimum phase system due primarily to the nature of it's non-holonomic dynamics. Intuitively, we seek to feedback to linearize the bicycle to control it's x and y location through choice of linear inputs v_1, and v_2. The difficulty arises in enforcing the no slip condition between the bicycle's tires and the road. Because the tire cannot slip, the bicycle is forced to move in such a way that the front tire is constantly moving forward along the plane of the tire.

The bicycle model chosen utilizes some key assumptions that simplify the construction of a dynamic model. First, the upright bicycle is assumed to have a fixed steering axis perpendicular to the ground. The introduction of tilt to the steering axis has multiple desireable residual effects on the dynamics of the bicycle including allowing the bicycle to be self-stabilizing in some situations[45], but the modelling of this condition would be needlessly complicating for our purposes. Second, a single inertia tensor, J, is associated with both the bicycle frame and any rigidly attached payload and is expressed in the center of mass frame of the vehicle. Further, J is assumed to be a scalar multiple of the 3 by 3 identity matrix. Finally, instead of modelling the driving force of the car as a common chain and sprocket mechanism, we place a force generator at a point opposite the forward direction of the bicycle, in the plane of the bicycle frame and at the height of the center of mass to avoid a net pitch moment on the bicycle. Diagrams of the bicycle configuration can be found in figures 3.1, and 3.2.

A Lagrangian formulation of the bicycle's dynamics results in the following equations:

$$\begin{gathered} \dot{s} = \omega \\ M(q_1)\ddot{q} = F(q_1, \dot{q}) + B(q_1)u + A^T(s, q_1)\lambda \\ A(s, q_1)\dot{q} = 0 \end{gathered} \tag{3.6}$$

Where $q_1 = [\alpha, \theta]^T$, $q_2 = [x, y]^T$ and q is the stacked vector of q_1 and q_2, and u is the force generated by the force generator. Further, x and y represent the location of the center of mass, α is the roll-angle, and θ is the yaw angle, or, the angle formed between the x-axis and the contact line formed by the point of contact of both wheels. ϕ is the steering-angle of the bicycle, and $s := tan(\phi)/b$ is the steering variable with derivative ω. M is symmmetric and positive definite, A is full-rank and $\lambda \in \mathbb{R}^2$ are the lagrange multipliers used to enforce the non-holonomic constraints of the system. The zero dynamics resulting from this formulation can be described as:

$$\ddot{\alpha} = \frac{gpsin(\alpha)}{j + p^2} \tag{3.7}$$

Where g is gravity, p and j are constants determined by the shape, size, and mass of the bicycle. This corresponds to the equation of an inverted pendulum that is unstable about $\alpha = 0$. Consequently, the system is non-minimum phase.

To ameliorate the issues surrounding the non-holonomic constraints, the system can further be transformed to be in terms of $\dot{\tilde{q}} = [\dot{\alpha}, v_r, \dot{\theta}, v_p]$, via the tranformation $\dot{q} = T\dot{\tilde{q}}$.

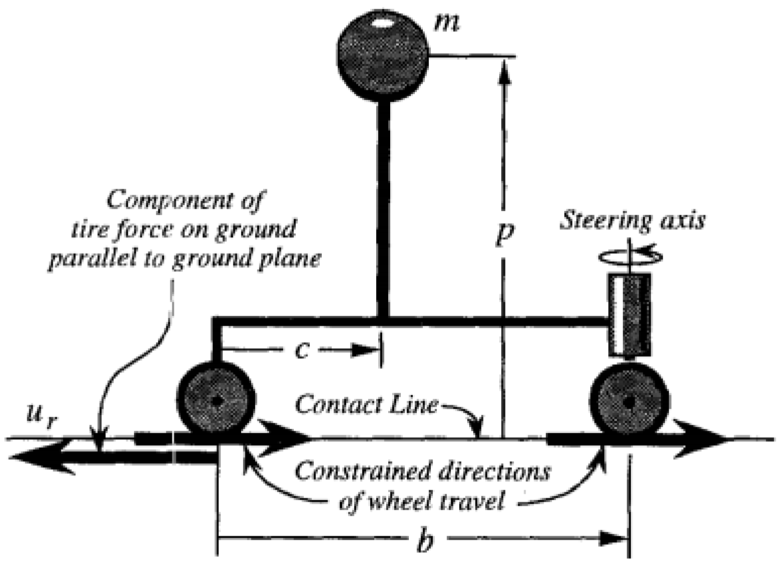
m
Component of tire force on ground parallel to ground plane
p
Steering axis
c
u_r
Contact Line
Constrained directions of wheel travel
b

Figure 3.1: Side view visualization of bicycle configuration [37]

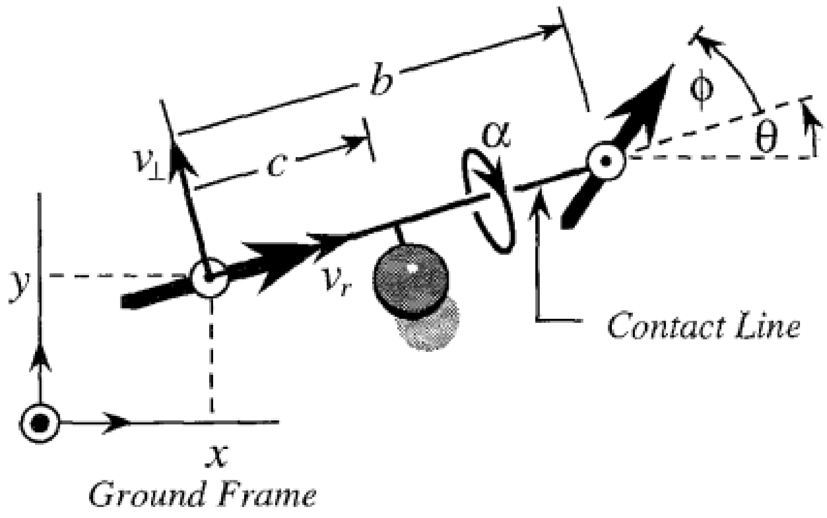

Figure 3.2: Top-down visualization of bicycle configuration [37]

As in q, $\dot{\tilde{q}}$ can be decomposed into $\dot{\tilde{q_1}} = [\dot{\alpha}, v_r]^T$ and $\dot{\tilde{q_2}} = [\dot{\theta}, v_p]^T$. The result of this transformation is a simplified set of dynamics that is not dependent on the A matrix:

$$\tilde{M}\ddot{\tilde{q_1}} = \tilde{F} + \tilde{B}\bar{u} \tag{3.8}$$

Here $\bar{u} := [w, u]^T$ is the vector of nonlinear inputs to the transformed system. Supposing $\tilde{B}$ is invertable, $v_r > 0$, and letting $\ddot{\tilde{q}} = v$, the nonlinear input can be found as:

$$\bar{u} = \tilde{B}^{-1}(-\tilde{F} + \tilde{M}v) \tag{3.9}$$

Further, letting $\alpha_d(t)$ and $v_{rd}(t)$ be the desired twice differential trajectories for the roll and rear-wheel velocity, with $v_{rd}(t) \neq 0$ for all times, t, and choosing real numbers c_{11}, c_{10}, c_{20} such that the polynomials $r^2 + c_{11}r + c_{10}$ and $r + c_{20}$ are stable, we can construct our new inputs as:

$$v = \begin{bmatrix} (\ddot{\alpha_d}) - c_{11}(\dot{\alpha} - \dot{\alpha_d}) - c_{10}(\alpha - \alpha_d) \\ \dot{v}_{rd} - c_{20}(v_r - v_{rd}) \end{bmatrix} \tag{3.10}$$

This choice of input will serve to stabilize the the roll angle and bicycle velocity to the desired values defined above, effectively accounting for the otherwise non-minimum phase behavior of the bicycle. One unfortunate side effect of this control technique is that it effectively sacrifices control of the bicycles x and y coordinates to insure the stability of the roll dynamics, causing this control approach to be severely limited in use. A more in depth explanation, as well as a complete derivation of the above dynamics can be found in [37].

The limited use of the controller posed above led to a reformulation of the feedback linearizing controller to directly control x and y, leaving the non-minimum phase zero dynamics to evolve unabated. We begin again, as in 2.1, taking derivatives of the output values we seek to control:

$$\begin{aligned} x &= x \\ \dot{x} &= cos(\theta)v_r \\ \ddot{x} &= -sin(\theta)\dot{\theta}v_r + cos(\theta)\dot{v_r} \end{aligned} \tag{3.11}$$

As well as:

$$\begin{aligned} y &= y \\ \dot{y} &= sin(\theta)v_r \\ \ddot{y} &= cos(\theta)\dot{\theta}v_r + sin(\theta)\dot{v_r} \end{aligned} \tag{3.12}$$

This, in tandem with the relationship, $\dot{\theta} = s\dot{v}$, leads us to our final form of the derivatives:

$$\ddot{x} = -sin(\theta)sv_r^2 + cos(\theta)\dot{v_r}, \quad \ddot{y} = cos(\theta)sv_r^2 + sin(\theta)\dot{v_r} \tag{3.13}$$

It can further be shown that $\dot{v_r}$ can be expressed as:

$$\dot{v_r} = (\hat{M}^{-1}\hat{F})_{2,1} + (\hat{M}^{-1}\hat{B})_{2,1}u + (\hat{M}^{-1}\hat{B})_{2,2}\omega \tag{3.14}$$

Where the notation $(A)_{n,m}$ denotes the (n,m) entry of matrix A. It is important to observe that 3.14 provides a relationship between $\dot{v_r}$ and ω, which would provide an avenue for controlling the system using ω. Upon closer inspection, it can be shown that the decoupling matrix for this formulation of the controller would be singular with respect to u, and ω. So we instead seek to formulate this problem, without substitution, in terms of s and $\dot{v}$:

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = \underbrace{\begin{bmatrix} -sin(\theta)v_r^2 & cos(\theta) \\ cos(\theta)v_r^2 & sin(\theta) \end{bmatrix}}_{A} \begin{bmatrix} s_d \\ \dot{v}_{r,d} \end{bmatrix} \tag{3.15}$$

Where A is the decoupling matrix of this new formulation and the desired values can be calculated as:

$$\begin{bmatrix} s_d \\ \dot{v}_{r,d} \end{bmatrix} = A^{-1}v \tag{3.16}$$

Here, v is the input calculated by the linear controller. This formulation does have some idocycracies; foremost, while we calculate control in terms of s, the dynamics explicitly require an input in terms of ω, hence we calculate:

$$\omega = -\gamma(s - s_d) \tag{3.17}$$

Where $\gamma > 0$ is chosen so that s tracks s_d as close as reasonably possible without causing instability in the dynamics or other erroneous behavior. rearranging 3.14 further provides a way to calculate u:

$$u = \frac{1}{(\hat{M}^{-1}\hat{B})_{2,1}}(\dot{v}_{r,d} - (\hat{M}^{-1}\hat{F})_{2,1} - (\hat{M}^{-1}\hat{B})_{2,2}\omega) \tag{3.18}$$

This choice of u will implicitly cause the relationship $\dot{v}_r = \dot{v}_{r,d}$ to hold, as desired. This controller formulation allows a specific sequence of x and y coordinates to be tracked while sacrificing any guarantees of stability of the roll angle of the bicycle. While this is normally not ideal, this is a natural formulation for the learning approach that will be developed in section 3.2

3.2 Formulating the Learning Problem

As discussed in section 1.3, the major difficulty of non-minimum phase systems rests in the eta dynamics, which are, by definition, not asymptotically stable, and some times even unstable when the ξ states are set to 0. these two types of behavior are demonstrated in the PVTOL and bicycle examples discussed in section 3.1, respectively. To rectify this, we have opted to learn how to account for the zero dynamics using reinforcement learning. Specifically, we aim to train an agent that can accept a desired nominal path for the system to follow and subsequently outputs a corrected path that deviates nominally from the desired path in an effort to keep the eta dynamics within a desired "safe" range. To formalize this approach, lets consider a set of dynamics in the standard, feedback linearized form:

$$\begin{aligned} \dot{\xi} &= A\xi + Bv \\ \dot{\eta} &= q(\xi,\eta) + p(\xi,\eta)v \end{aligned} \tag{3.19}$$

Where $\xi \in \mathbb{R}^P$, and $\eta \in \mathbb{R}^l$, implying $(\xi,\eta) \in \mathbb{R}^n$. Then we can formulate this problem as the following optimization problem:

$$\begin{aligned} \inf_{\theta\in\Theta} \mathbb{E}_{(\xi_0,\eta_0)\sim X,\phi\sim\Phi} \int_0^T & \lambda_P\|\xi_d(t,\phi)-\xi(t)\| + \lambda_{ZD}H(\eta(t)) + \lambda_I\|v(t)\| + \lambda_T\|\xi_d(T,\phi)-\xi(T)\|dt \\ & \dot{\xi}(t) = A\xi(t) + Bv(t) \\ & \dot{\eta}(t) = q(\xi(t),\eta(t)) + p(\xi(t),\eta(t))v(t) \\ & v(t) = \tilde{y}_d^{(\gamma)}(t,\alpha) + K(\tilde{\xi}_d(t,\alpha) - \xi(t)) \\ & (\xi(0),\eta(0)) = (\xi_0,\eta_0) \\ & \alpha = M_\theta(\xi_0,\eta_0,\phi) \end{aligned} \tag{3.20}$$

Where $\xi_d(\cdot,\phi) : [0,T] \to \mathbb{R}^p$ is the nominal trajectory that we seek to track, $\tilde{\xi}_d(\cdot,\alpha) : [0,T] \to \mathbb{R}^p$ is the corrected trajectory to be followed in actuality, M_θ is the neural network parameterized by θ, and K is the fixed feedback gain matrix calculated as the solution of the continuous time LQR problem described in section 1.4. Further Φ is a distribution over some subset of $\mathbb{R}^k$, with K, the number of parameters for the trajectories to be tracked. The cost function is constituted by four terms; $\|\xi_d(t,\phi) - \xi(t)\|$ seeks to penalize the neural network for deviation from the desired path, with weight $\lambda_P \geq 0$. $H(\eta(t))$ is a function that serves to penalize the growth of the eta dynamics beyond the pre-defined safe range with weight $\lambda_{ZD} \geq 0$ and $\|v(t)\|$ is intended to place a cost on the use of the inputs with weight $\lambda_I \geq 0$.

Finally, $\|\xi_d(T,\phi) - \xi(T)\|$ penalizes the difference between the desired and actual terminal states at time T with weight term $\lambda_T \geq 0$. Turning now to the constraints of this problem, the first two constraints serve to enforce the feasibility of the dynamics. The third constraint enforces that the linear input is calculated as expected. The fourth constraint enforces the initial conditions of the system and the final constraint simply indicates that the α value used to calculated our desired values is the output of the trained neural network. In practice, the desired path is posed as a sequence of way points that are then used to interpolate an appropriate spline; The spline is then symbolically differentiated and the spline as well as its derivatives are quarried to determine the overall path to be followed. Similarly, the network outputs a set of way points and a similar process is used to determine the corrected path.

Of particular interest in this formulation is the cost on the zero-dynamics, $H(\eta(t)) > 0$. We began with a few basic conditions; First, the function must be piece-wise continuous on $\mathbb{R}$. Second, the function must be convex in order to insure the existence of one or more minima in that the algorithm can seek through experimentation. With this in mind, a piecewise continuous function was chosen:

$$H(\eta(t)) = \begin{cases} |(\eta(t) - B_U)^{c_h}| & \eta(t) > B_U \\ 0 & B_L < \eta(t) < B_U \\ |(\eta(t) + B_L)^{c_h}| & \eta(t) < B_L \end{cases} \tag{3.21}$$

Here, $B_U > 0$ and $B_L < 0$ are upper and lower bounds chosen by the engineer to demarcate "soft" boundaries for a safe range of operation for the eta dynamics. c_h is a constant coefficient tuned in order to control how quickly the cost on the eta dynamics begins to increase from zero outside of the "soft" safe zone. This cost scheme is convenient for multiple reasons; First, it is piece wise continuous and forms a convex function for which the set $[B_L, B_U]$ are all minimizing solutions. An additional benefit of this approach is the ability to tune the cost with only two coefficients. by increasing λ_{ZD} and c_h, the plot can be made arbitrarily steep, allowing for a gradual increase of cost where a wider range of $\eta(t)$ values is acceptable and an almost directly vertical, "psuedo-hard" constraint when a certain range must be strictly adhered to.

3.3 Experiments, Results, and Discussion

Planar Vertical Take-Off and Landing

As the simpler of the two examples, we sought to demonstrate the overall performance of our approach on the PVTOL case. We assume that the system starts at an origin in the air, allowing it to maneuver in any direction the learned agent sees fit. For this experiment, we chose to use $\epsilon = .3$, which, in accordance with 3.4, corresponds to a relatively large constant multiplier on the zero dynamics and providing an approachable challenge to learn a correcting path-planning agent. The agent was allowed to train on 300,000 simulated

PVTOL scenarios where, in each case, it was tasked with achieving x, and y coordinates selected randomly from $[-5, 5]$ within a 3 second window of simulation while maintaining the stability of the eta dynamics. In this case, the soft constraint we aimed for was keeping the eta dynamics in the range $[-1, 1]$, with coefficients $\lambda_{ZD} = 300$ and $c_h = 2$. B_U and B_L were chosen so that $H(1) = H(-1) = 1$. Finally, the other coefficients were chosen to be: $\lambda_P = 1$, $\lambda_I = 0$, $\lambda_T = 1000$. This configuration was constructed to give the agent the most flexibility and incentive to reach the terminal condition while keeping the eta dynamics as in check.

To test the trained agent, we tasked it with accomplishing an x,y coordinate of [3,3] in the same 3 second period. A plot of the resulting path it created and followed can be found in 3.4. For comparison, a plot of the nominal path, a straight line from $[0, 0]$ to $[3, 3]$ without any learned path correction is shown in fig 3.3. We observe that the uncorrected path results in exact tracking of the straight line, but further causes the zero dynamics to become relatively large compared to our "soft" safety bounds, reaching a roll angle of over 4 rads, which would correspond to more than one complete rotation of the system. The learned agent, however, decreases the max roll angle achieved significantly. By deviating from the nominal path, it can actively account for the eta dynamics. It performs well to keep the system at a roll angle just below 1 rad for the first 2 seconds of the simulation, and begins turning the opposite direction achieving a minimum roll angle of roughly -2 rads. It is also important to

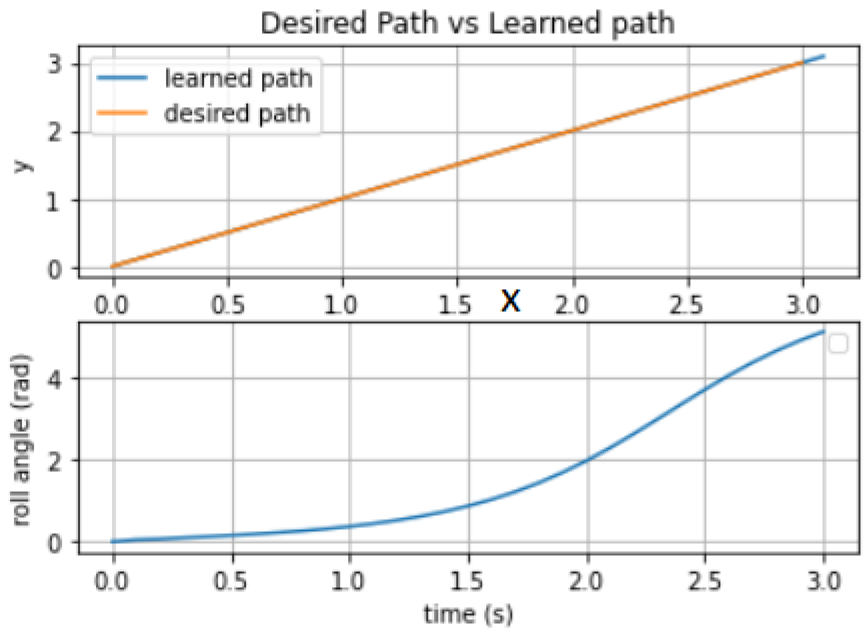

Figure 3.3: Plot demonstrating performance of default controller

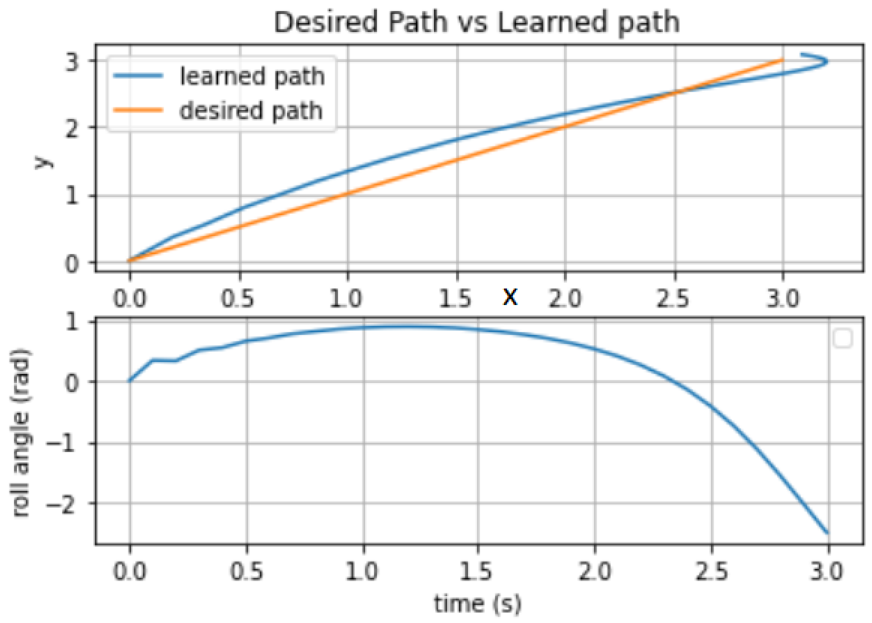

Figure 3.4: Plot demonstrating performance of agent on PVTOL problem

note that the agent achieves a final x,y, coordinate of $[3.08, 3.04]$.

While this is a marked improvement over following the nominal path, even these smaller rotations may be uncomfortable for an operator on a real physical system. So, even tighter constraints should be explored moving forward, making the problem even more challenging. In total, while we have demonstrated significant progress on this problem, there are still improvements to be made, the costs of the λ values need to continue being tuned to better meet the desired *eta* conditions and terminal state. Further, the parameters of the SAC algorithm being used to train the agent need to be tuned to insure more consistant, reproducible performance.

Bicycle

For the bicycle example, We first sought to reproduce the controller of [37], defined by 3.10, in order to verify that the simulated dynamics were correct. Specifically, we aimed to reproduce experiment 1, which involved starting the bicycle at an initial roll angle $\alpha(0) = -\pi/4$, $v_r(0)$=2, $x(0) = y(0) = 0$, and neutral steering angle with controller parameters $c_{11} = 2, c_{10} = 6, c_{20} = 1$. Full details of the experiment can be seen in [37]. A comparison of our simulated results to those of Getz can be seen below in 3.5. This figure demonstrates an exact reproduction of the path traveled by Getz' simulation and confirmed that the dynamics were implemented correctly.

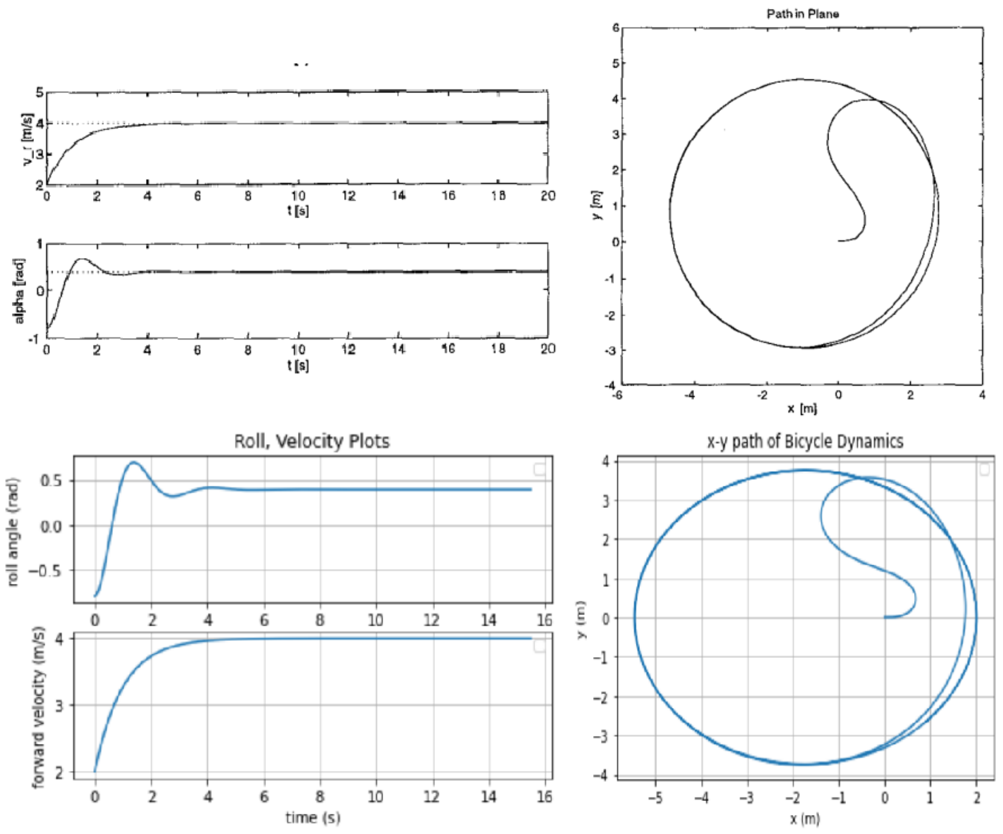

Figure 3.5: Comparison of plots in reproduction of experiment 1 [37].

Next, we sought to implement the posited linearizing controller of 3.16. We opted to start the bicycle at [0,0] with initial conditions $\alpha(0) = 0$ and $v_r(0) = 2$ and $\beta = 2$ and other parameters as in experiment 1. The controller was asked to follow a straight-line path of y(t)=x(t)=t for 30 seconds and the results of this experiment are shown below in figures 3.6 and 3.7. Because the bicycle is made to start oriented along the x axis, it must start going straight and slowly turn to better align itself with the desired path, after overshooting the path it must correct by turning again, continuing on in this fashion for the majority of the trial, but overshooting less each time. Finally by the end of the trial at 30 seconds, the controller has nearly converged to the desired path. The calculated roll angle, as expected is left unchecked and diverges to unreasonably large numbers, nearing 200 rads of tilt on the bicycle. This is, of course, not possible due to the nature of a bike and it's roll angle, but acts as a good indication that there may be use for our posited algorithm in the control of a bicycle. Finally, the forward velocity cycles higher and lower to keep the bicycle near it's desired waypoint while ultimately settling at approximately 1.4 m/s.

Finally, we had hoped to begin running trials of the full learning problem as in the

PVTOL case, but a few key issues arose. First, the lagrangian formulation of the dynamics proved significantly more complicated than that of PVTOL, requiring a greater number of nontrivial matrix operations per time step simulated. Second, we found that this formulation required much smaller time steps in simulation to insure the convergence of the differential equations being simulated. These two facts together made the run time of a single trial of the simulation roughly 2-3 seconds, as opposed to the fraction of a second that a single PVTOL simulation takes. The result of this is that the paradigm of running hundreds of thousands of simulations to train an agent is currently intractable on a consumer grade personal computer. It is possible that this may be rectified through one or more steps moving forward. First, we may consider attempting to learn paths that are relatively short, requiring less simulation time to execute and fewer times steps to simulate overall. Further, there may be performance gains to be had in optimizing the code for the particular case of a bicycle.

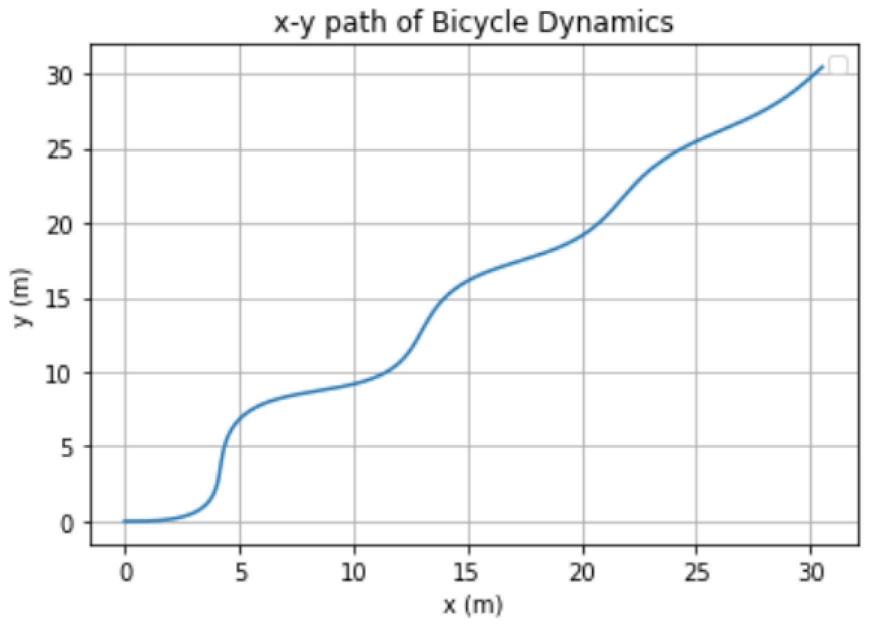

Figure 3.6: Path of bicycle in feedback linearization experiment

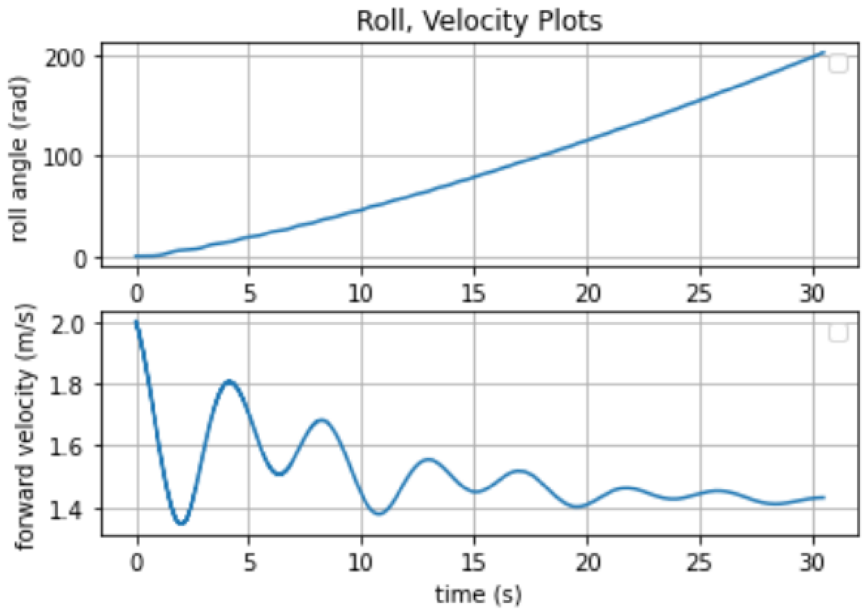

Figure 3.7: Roll angle and Velocity of bicycle in feedback linearization experiment

Chapter 4

Concluding Remarks

4.1 Summary

We have presented two techniques for the control of non-linear, and later, non-minimum phase systems. The technique of Learning Feedback Linearization, first posed by [9] was applied to the case of a race car that was feedback linearized based on a dynamically extended bicycle model. an exactly feedback linearizing controller was learned via LFBL in the case where exact linearization was possible; that is, when the error between the initial estimate of the model and the true dynamics was defined only by a difference in the coefficients of the same overall dynamic model. We then attempted to apply this to the case of a more complicated dynamic model, simulated using the CarRacing-v0 gym environment. This environment serves to model additional behaviors not considered in a dynamically extended bicycle model, including things like tire slip and delays in steering and acceleration. To accomplish this, a preprocessing network was posited to convert the nominal inputs calculated by the LFBL into the inputs for the race car environment. Data of the race car's response to randomly selected inputs to the racing environment was collected to train the network. As this line of work currently stands, initial experiments in the exact linearization case have yielded promising early results, but do not, as of yet, seem to work as expected. This could be due to a variety of reasons, but is most likely due to the need for further experimentation with the parameters of the reinforcement learning algorithm being used. Because The more complex task of controlling a race car with additional dynamic considerations implicitly requires fitting an approximate linearizing model, we have yet to begin experimentation in earnest. We have instead opted to reserve those experiements until the exact linearizing controller can be fine-tuned and demonstrated to perform as well as reasonably possible.

next, we developed a Reinforcement Learning methodology for the control of non-minimum phase systems based intuitively on the notion of correcting a desired path to better account for the zero dynamics of the system in question. The formulation was made to learn this corrected path by the choice of cost coefficients that implicitly encode the relative importance of several key factors in the systems evolution, including, how close it is to the nominal desired path, consistently keeping the zero dynamics of the system within a desired envelope, minimizing the use of input signals, and insuring that the desired terminal state is met. This is accomplished while explicitly enforcing the dynamics of the system being studied. This technique was applied to the case of an aircraft attempting Planar Vertical Take Off and Landing. Further, a dynamic simulation of a two-wheeled bicycle based on [37] was implemented, debugged, and a new linearizing controller was designed to better suit the needs of our project. At the time of writing, this line of work is still mid-development. As such, we have begun initial experiments on the PVTOL case with mixed results. While the agent appears to improve it's performance over many iterations of attempting to learn a corrected path, it still does not consistently converge to a solution that can routinely find a corrected path that abides the desired bounds on the zero dynamics. We suspect that this is, again, due to the need for further experimentation and tuning of the Reinforcement Learning algorithm being used. The bicycle example entails much more complicated dynamics than the PVTOL system, and has consequently introduced some practical difficulties to the project. Specifically, the time it takes to simulate the bicycle in one trial with the agent is roughly an order of magnitude greater than that of the PVTOL system, primarily owing to the lagrangian formulation used. This significantly increases the time to train an agent beyond what is practical on a consumer grade PC. We do expect that there are performance gains to be made through code optimizations and tweaking of experimental parameters, but it may ultimately require computational resources beyond what is currently available to us. This may include something like a dedicated server in the lab or outsourcing the computation to a cloud computing provider like Amazon or Google.

4.2 Future Work

Due in part to the current state of these two lines of work, as discussed above, there are some obvious next steps to be taken in the continuation of this project. In terms of controlling a race car with LFBL, there is further work to be done in experimenting with the exactly linearizing controller in order to refine its ability to learn an effective corrective controller and follow a desired path. This would involve performing multiple sweeps across various parameters of the Reinforcement Learning algorithm being used. Once the LFBL agent can be trained suitably well on the exactly linearizing case, we can begin implementing the controller for the full use in the approximately linearizing controller. This would involve training a new agent to learn the dynamics of the more complicated racing environment as an approximated bicycle model, this would again require an extensive cycle of experiementation to determine the best set of hyperparameters for the learning agent. After that, the proposed preprocessing network can be retrained and tested to verify it's ability to faithfully convert the inputs provided by the controller into the inputs needed for the racing environment. In terms of learning for non-minimum phase control, the first step would be continuing to fine tune the learning algorithm's hyperparameters to demonstrate more consistant performance on the task of achieving a (x,y) terminal condition while respecting the constraints placed on the zero dynamics and the path we want it to follow. Once that has been accomplished and the overall approach demonstrated in earnest, we intend to address the looming issue with the bicycle dynamics. That being, it's relatively slow performance in completing a single run. This will be done by seeking to restructure and optimize the code that was written to perform the simulation with the hope that major performance gains can be achieved. If not, then an outside source of computational resources will be sought. It may alternatively be the case that a different, simpler dynamic model demonstrating non-minimum phase behavior will be chosen and implemented. With a practical and functional set of bicycle dynamics, we will begin experiments in earnest. Specifically we hope to construct an example of a bicycle trying to make a turn. This maneuver would excite the zero dynamics in a way that a successfully trained agent could correct the desired path while still arriving at the appropriate end state.

Beyond further refinement of the approaches and methods discussed here, there are a number of exciting possible avenues to take this work. Once the LFBL controller can be demonstrated to perform well in the racing environment, then we can consider expanding the method to work on hardware such as the ROAR cars, or even full scale vehicles. While the current methodology would work in principle for a physical car on a flat surface, the development of a similar controller that can account for changes in height (e.g. ascending a hill) would motivate the search for a way to reformulate the dynamics for the 3D case.
The non-minimum phase control work also has some natural ways in which we hope to extend it's performance. First, we would seek to develop the learning agent to iteratively plan longer paths than the single time horizon over which it currently plans. We would accomplish this by generalizing the training and overall formulation of the agent to work with arbitrary initial conditions, instead of the nominal zero state that all of our experiments are currently initialized at. With this, the agent could then be used to continually replan as it progresses along a predefined path that would take longer than the planner could reasonably expect to travel over a single look-ahead period. An example of a scenario where this may be useful is the execution of a slalom with the bicycle. A nominal path can be designated as a sin wave going around the markers that define the course. While the agent as we envision it here might only be able to plan once through a small portion of the course due to its finite time horizon, it can be iteratively applied to follow the slalom from whatever starting conditions it may be in when a new plan is formulated. This example also poses the interesting question of how the planner can be formulated to avoid obstacles like the markers, motivating the study of how this formulation can be developed to account for obstacles that it cannot or should not interact with.

Bibliography

[1] S. Sastry. *Nonlinear systems: analysis, stability, and control.* Vol. 10. Springer Science & Business Media, 1999.

[2] A. Isidori. *Nonlinear control systems.* Springer Science & Business Media, 2013.

[3] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic programming.* Athena Scientific, 1996.

[4] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction.* 2018.

[5] R. S. Sutton et al. "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in neural information processing systems.* 2000, pp. 1057–1063.

[6] J. Schulman et al. "Trust region policy optimization". In: *International conference on machine learning.* 2015, pp. 1889–1897.

[7] T. P. Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[8] J. Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[9] T. Westenbroek et al. "Feedback Linearization for Unknown Systems via Reinforcement Learning". In: *arXiv preprint arXiv:1910.13272* (2019).

[10] P. Martin, R. M. Murray, and P. Rouchon. "Flat systems, equivalence and trajectory generation". In: (2003).

[11] R. E. Kalman et al. "Contributions to the theory of optimal control". In: *Bol. soc. mat. mexicana* 5.2 (1960), pp. 102–119.

[12] F. Borrelli, A. Bemporad, and M. Morari. *Predictive control for linear and hybrid systems.* Cambridge University Press, 2017.

[13] J. W. Grizzle, G. Abba, and F. Plestan. "Asymptotically stable walking for biped robots: Analysis via systems with impulse effects". In: *IEEE Transactions on automatic control* 46.1 (2001), pp. 51–64.

[14] A. D. Ames et al. "Rapidly exponentially stabilizing control lyapunov functions and hybrid zero dynamics". In: *IEEE Transactions on Automatic Control* 59.4 (2014), pp. 876–891.

[15] D. Mellinger and V. Kumar. "Minimum snap trajectory generation and control for quadrotors". In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 2520–2525.

[16] S. Sastry and M. Bodson. *Adaptive control: stability, convergence and robustness*. Courier Corporation, 1989.

[17] J. J. Craig, P. Hsu, and S. S. Sastry. "Adaptive control of mechanical manipulators". In: *The International Journal of Robotics Research* 6.2 (1987), pp. 16–28.

[18] S. S. Sastry and A. Isidori. "Adaptive control of linearizable systems". In: *IEEE Transactions on Automatic Control* 34.11 (1989), pp. 1123–1131.

[19] K. Nam and A. Araposthathis. "A model reference adaptive control scheme for pure-feedback nonlinear systems". In: *IEEE Transactions on Automatic Control* 33.9 (1988), pp. 803–811.

[20] I. Kanellakopoulos, P. V. Kokotovic, and A. S. Morse. "Systematic design of adaptive controllers for feedback linearizable systems". In: *1991 American Control Conference*. IEEE. 1991, pp. 649–654.

[21] J. Umlauft et al. "Feedback linearization using Gaussian processes". In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE. 2017, pp. 5249–5255.

[22] G. Chowdhary et al. "Bayesian nonparametric adaptive control using gaussian processes". In: *IEEE Transactions on Neural Networks and Learning Systems* 26.3 (2014), pp. 537–550.

[23] G. Chowdhary et al. "Bayesian nonparametric adaptive control of time-varying systems using Gaussian processes". In: *2013 American Control Conference*. IEEE. 2013, pp. 2655–2661.

[24] *Autonmous Racing Lueagues/Competitions*. URL: `https://diyrobocars.com/autonomous-racing-leaguescompetitions/` (visited on 12/14/2020).

[25] *AWS DeepRacer*. 2020. URL: `https://aws.amazon.com/deepracer/` (visited on 12/14/2020).

[26] *F1tenth*. 2020. URL: `https://f1tenth.org/index.html` (visited on 12/14/2020).

[27] *RoboRace*. 2020. URL: `https://roborace.com/` (visited on 12/14/2020).

[28] G. Williams et al. "Aggressive driving with model predictive path integral control". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 1433–1440.

[29] U. Rosolia and F. Borrelli. "Learning Model Predictive Control for Iterative Tasks. A Data-Driven Control Framework". In: *IEEE Transactions on Automatic Control* 63.7 (2018), pp. 1883–1896.

[30] M. N. Z. Lukas Hewing Juraj Kabzan. "Cautious Model Predictive Control using Gaussian Process Regression". In: *arXiv preprint arXiv:1705.10702* (2019).

[31] A. Isidori. “The Zero Dynamics of a Nonlinear System: From The Origin To the Latest Progresses of a long successful story”. In: *Proceedings of the 30th Chinese Control Conference*. 2011, pp. 18–25.

[32] D. CHEN and B. PADEN. “Stable inversion of nonlinear non-minimum phase systems”. In: *International Journal of Control* 64.1 (1996), pp. 81–97. URL: `https://doi.org/10.1080/00207179608921618`.

[33] A. Isidori and C. Byrnes. “Output regulation of nonlinear systems”. In: *IEEE Transactions on Automatic Control* 35.2 (1990), pp. 131–140.

[34] H. Ye et al. “New stabilization design for planar vertical take-off and landing aircrafts”. In: *Journal of Control Theory and Applications* 9 (May 2011), pp. 195–202.

[35] L. Mao-Qing. “Control design for planar vertical takeoff-and-landing aircraft based on controlled Lagrangians”. In: 27 (June 2010), pp. 688–694.

[36] S. J. Petkar et al. “Robust Model Predictive Control of PVTOL AircraftThis work was supported by Center of Excellence in Complex and Nonlinear Dynamical Systems (CoE-CNDS), established under TEQIP-II, Subcomponent 1.2.1.” In: *IFAC-PapersOnLine* 49.1 (2016). 4th IFAC Conference on Advances in Control and Optimization of Dynamical Systems ACODS 2016, pp. 760–765. URL: `https://www.sciencedirect.com/science/article/pii/S2405896316301483`.

[37] N. Getz and J. Marsden. “Control for an autonomous bicycle”. In: *Proceedings of 1995 IEEE International Conference on Robotics and Automation*. Vol. 2. 1995, 1397–1402 vol.2.

[38] M. Yamakita, A. Utano, and K. Sekiguchi. “Experimental Study of Automatic Control of Bicycle with Balancer”. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2006, pp. 5606–5611.

[39] L. Guo et al. “A kind of bicycle robot dynamic modeling and nonlinear control”. In: *The 2010 IEEE International Conference on Information and Automation*. 2010, pp. 1613–1617.

[40] E. X. Wang et al. “Development of Efficient Nonlinear Benchmark Bicycle Dynamics for Control Applications”. In: *IEEE Transactions on Intelligent Transportation Systems* 16.4 (2015), pp. 2236–2246.

[41] M. Cook. “It takes two neurons to ride a bicycle”. In: (Jan. 2004).

[42] T. P. Le et al. “Learning a Self-driving Bicycle Using Deep Deterministic Policy Gradient”. In: *2018 18th International Conference on Control, Automation and Systems (ICCAS)*. 2018, pp. 231–236.

[43] T. Omata. “Learning with assistance based on evolutionary computation”. In: *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*. Vol. 3. 1998, 2180–2185 vol.3.

[44] J. Escalona and A. Recuero. “A bicycle model for education in multibody dynamics and real-time interactive simulation”. In: *Multibody System Dynamics* 27 (Mar. 2012).

[45] K. Åström, R. Klein, and A. Lennartsson. “Bicycle dynamics and control: Adapted bicycles for education and research”. In: *Control Systems, IEEE* 25 (Sept. 2005), pp. 26–47.

[46] J. Tan et al. “Learning bicycle stunts”. In: *ACM Transactions on Graphics (TOG)* 33 (2014), pp. 1–12.

[47] T. Haarnoja et al. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”. In: (2018).

[48] J. Achiam. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. 2018. URL: `https://spinningup.openai.com/en/latest/algorithms/sac.html`.

[49] V. Mnih et al. “Playing Atari with Deep Reinforcement Learning”. In: (2013).