UNIVERSITY OF CALIFORNIA
SANTA CRUZ

METRICS-ONLY TRAINING OF A NEURAL NETWORK FOR SWITCHING AMONG AN ARRAY OF FEEDBACK CONTROLLERS

A dissertation submitted in partial satisfaction of the
requirements for the degree of

MASTERS OF SCIENCE

in

COMPUTER ENGINEERING

by

Marco Carmona

June 2022

The Dissertation of Marco Carmona
is approved:

Professor Dejan Milutinović, Chair

Professor Michael Wehner

Professor Yu Zhang

Peter Biehl
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

Abstract

Metrics-only Training of a Neural Network for Switching Among an Array of Feedback Controllers

by

Marco Carmona

We propose a novel training approach for neural networks based on switching among an array of feedback controllers (FC). Traditionally, the neural network training implemented in reinforcement learning problems can be achieved with observable kinematic variables (KV). In this work, our training approach takes a step further by using the metrics-only (MO) or LTM inputs into the networks where each metric and FC fulfills a design specification. Alongside, the reinforcement learning algorithm is a hierarchical control architecture for switching among multiple FCs and its modes. With the designed FCs based on Lyapunov functions or stochastic optimal control, we show that the MO training has a faster convergence with less variations than the one that is based on KV. These results are important for applications requiring large number of controllers. We provide results for a pendulum control problem, a bicycle, and a tricycle navigation problem. In the case of the tricycle problem, we also show that the trained neural network can be applied beyond numerically simulated control problems. The results of this work are illustrated by numerical and virtual reality simulations.

To God Almighty

and

my family

Acknowledgments

Thank you to my advisor Professor Dejan Milutinović for initially presenting the problem investigated in this thesis. His incites were invaluable and even more so is his mentorship. Furthermore, I would like to thank the U.C. Santa Cruz Robotics & Control lab for all the help and support.

Chapter 1

Introduction

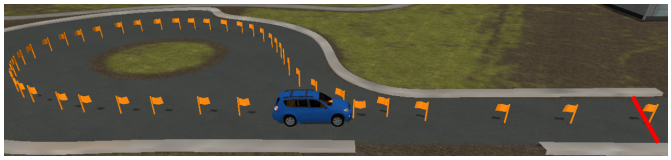


Figure 1.1: Stochastic optimal steering of a car: the figure is a snapshot from the high-fidelity simulator Anvel 3.0, in which we implemented the steering.

Typically, the design of a feedback controller is based on a certain set of assumptions. These assumptions are in the form of models, training data, parameters, etc. However, real world applications may or may not correspond to a single or a combination of these assumptions. The topic of this thesis work is on how to combine feedback controllers (FC), each designed under specific assumptions in a synergistic way to obtain a controller that is under a certain metrics better than a sum of its individual parts. We specifically investigate the synergistic composition by using a neural network that implements a policy for switching among the controllers. As depicted in Fig. 1.2,

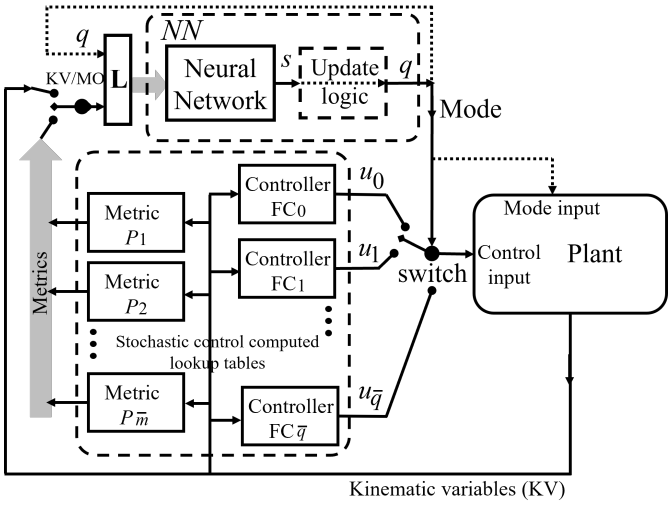


Figure 1.2: The hierarchical control architecture: the neural network is trained for switching among an array of feedback controllers (FC). The dotted lines indicate connections that are omitted for certain plants.

the neural network decides which controller should be active and plays the role of a supervisory controller.

The scenario which motivates this work is depicted in Fig. 1.1. The figure shows a car that navigates a circular road. It is obvious that for the navigation of the road, the driver does not need to know the exact radius of the circular road and the car could be to the right or to the left of the circular path and still traverse the road. The driver must trade off between adjusting the car velocity and keep the center of the road, otherwise the vehicle could veer off the road. Our approach to the trade-off is to design an array of controllers, in which each controller can navigate the road at a single fixed velocity. The neural network is trained using only the kinematic variables (KV) describing the motion of vehicle, or using only performance metrics of the controllers

(MO). During the training, we can also account for the constraints for the switching sequence as shown in Fig. 1.2.

In this work, we study a bicycle, a tricycle mobile robot navigation problem, and a pendulum mechanical system. Using a bicycle kinematic model we compute multiple stochastic optimal controllers with their corresponding metrics for the navigation at different velocities. Then, we train the neural network for switching among the controllers, but taking into account that the bicycle velocity cannot be abruptly changed. After verifying the control in numerical simulations, we test the controller for a front-wheel drive (FWD) tricycle in a high fidelity robot simulator. Following this, we also find a way to implement the same controller to a back-wheel drive (BWD) tricycle.

In the case of the pendulum mechanism, the neural network is trained to switch among multiple controllers, so that the pendulum reaches the upright position in minimum time. In this case, the controllers are designed analytically and there are no constraints on the switching sequence among them.

Overall, in all presented examples, the switching among the controllers is based on neural networks that are trained using reinforcement learning. The examples show that the learning and the hierarchical control architecture provide a general framework to address control problems in a wider range of systems with arrays with known metrics.

1.1 Related Work

The control architecture in Fig. 1.2 is a version of hierarchical control architectures that have been studied both in control and robotics [52]. Due to the continuous dynamics of the system under control and the switching among multiple controllers, this system is also an example of *hybrid* or *supervisory* control systems [33]. While the switching may result in instability [38, 12, 36], it can also contribute to performance improvements [46, 37]. A performance signal-based switching for mobile robots has been considered in [26] and the optimality of switching has been considered in [7] and [41]. To switch between the navigation to goal and obstacle avoidance in [58], the authors used Lyapunov functions, while in [43], the authors considered a stochastic problem and used a rule for switching based on a probability inequality. This use of probabilistic inference to implement switching inspired our work on learning a safe switching policy. The work presented here is aligned with hierarchical approaches for mobile robot navigation which use learning [18, 22, 59] or stochastic control [45, 44], as well as the emergence of data-driven approaches for learning rules for switching among multiple controllers [1, 27] or options [55, 56, 8].

The options in our work are FCs that can be defined analytically based on control theory or can be computed, e.g., using the numerical stochastic optimal control [34]. Once we have obtained the FCs, we use reinforcement learning and the actor-critic [53, 8] neural network to learn the rule for switching among the controllers. The neural network training is performed under the assumption that the rule is applied at every

time step; in other words, we deal with *single-step options* [55]. A way to formulate the training is to use data associated with the original problem formulation as in [56]. In that case, the KV/MO switch in Fig. 1.2 would be in the position KV, indicating that the learning is based on the KV, otherwise, the switch indicates that the training is based on MO.

A particular focus has been given when generating MO based controllers, which stemmed from studying the problem of steering a vehicle in the presence of uncertain information about the road. In that case, Kalman Filtering [29] or data fusion techniques [10] can be used to reduce the uncertainty associated with the sensory data and precisely estimate the steering relevant variables. Alternatively, the problem can be approached as a robust path-planning problem using various path-planning methods [35], such as Rapidly-exploring Random Trees [50] or Model Predictive Control strategies [32]. Finally, the problem can be considered as a problem of adaptive feedback control [62] or a machine learning problem for which the deep learning approach [9] can be used to learn the feedback control for steering. In this work, we use computationally generated stochastic optimal controllers with properties that can be proven based on the principles of optimal control and stochastic calculus. This approach has been previously used for various types of UAV navigation tasks [4, 3], and has been tested both with a UAV [39] and with differential drive robots [40].

1.2 Outline

After this introductory chapter, we describe the neural network training which is implemented in the PyTorch environment. As a preliminary for the learning, we compute a stochastic optimal control to navigate a vehicle as discussed in Chapter 3. In chapters 4 and 5, we take multiple controllers, which are based on the principles described in Chapter 3, and use the neural network for switching among them. Additional results related to Chapters 3-5 are provided in Appendices A and B. Lastly, in Chapter 6, we provide conclusions and plans for future work. The outline of Chapters 3-5 is provided below.

- **Chapter 3. Stochastic Optimal Approach to the Steering of an Autonomous Vehicle through a Sequence of Roadways.** This chapter discusses the implementation of a stochastic optimal controller for steering a vehicle to robustly follow an unpredictably winding road. The controller is based on a bicycle model and the road is defined as a sequence of roadways. Each roadway has a fixed position, but its orientation is uncertain. To anticipate this uncertainty, we model the orientation with a Brownian stochastic process, which serves as a stochastic process model for the orientation observations. The stochastic controller based on such a model implicitly creates a robust road following controller. The control design is illustrated with numerical simulations and implemented for steering a car in a high-fidelity car simulator.

- **Chapter 4. Metrics-only Training Neural Network for Switching among an Array of Feedback Controllers for Bicycle Model Navigation**. Here we propose a novel training approach for a neural network to perform switching among an array of computationally generated stochastic optimal feedback controllers. The training is based on the outputs of off-line computed lookup-table metric (LTM) values that store information about individual controller performances. Our study is based on a problem of bicycle kinematic model navigation through a sequence of gates and a more traditional approach to the training is based on kinematic variables (KVs) describing the bicycle-gate relative position. We compare the LTM and KV based training approaches to the navigation problem and find that the LTM training has a faster convergence with less variations than the KV based training. Our results include numerical simulations illustrating the work of the LTM trained neural network switching policy.

- **Chapter 5. Metrics-only Training of Neural Networks for Switching among an Array of Feedback Controllers**. Inspired by the novel training approach in Chapter 4, here we provide further results using metrics-only (MO) training. We are repeating the Chapter 4 training methods and further confirm its results with more examples. In this chapter, we include trainings based on the Lyapunov functions. We provide results for a pendulum control problem and a tricycle navigation problem. We show results for both a front-wheel and back-

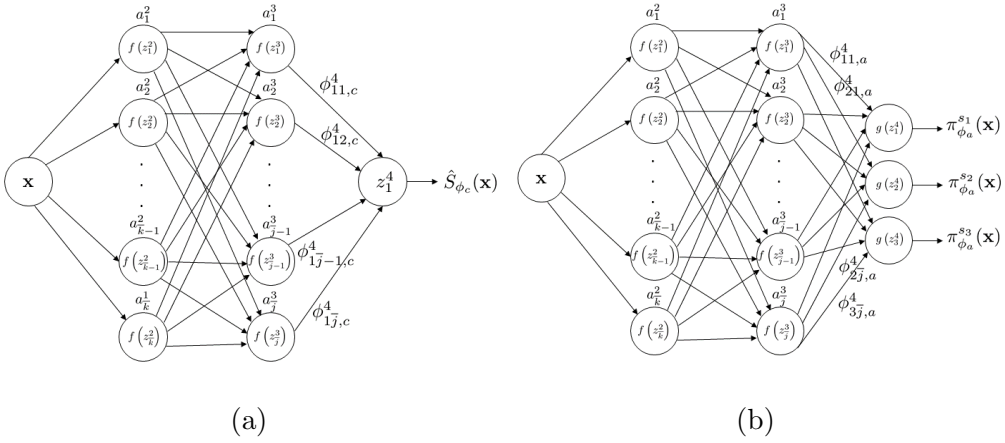
wheel drive tricycles. The training for both problems can be achieved based on MO and observable kinematic variables (KV). The number of inputs to the neural network in the case of the MO approach depends on the number of controllers, which is important for applications requiring a large number of controllers. In the case of the tricycle problem, we also show that the trained neural network can be applied beyond numerically simulated control problems.

Chapter 2

Neural Network Training Implementation in PyTorch

2.1 Introduction

The neural network is a universal function approximator able to approximate any arbitrary continuous function [11]. A pair of neural networks used in this work are the critic network shown in Fig. 2.1a and the actor network shown in Fig. 2.1b. These networks are approximators that have multilayer perceptron structure mapping inputs to outputs using nonlinear functions. We present the neural network architectures used in this work and how we implemented the algorithm for updating the network parameters in PyTorch.



(a) (b)

Figure 2.1: Multilayer feedforward networks: (a) a critic neural network; (b) an actor neural network. Both networks have an array of inputs denoted by $\mathbf{x}$. The output for the critic $\hat{S}_{\phi_c}(\mathbf{x})$ is dependent on $\mathbf{x}$ and its weight parameters ϕ_c and the output for the actor $\pi^s_{\phi_a}$ is dependent on $\mathbf{x}$ and its weight parameters ϕ_a. The forward direction given by the weight parameters ϕ_c and ϕ_a is indicated in the figure by arrows. The nonlinearities of the networks are introduced with nonlinear activation functions $a = f(z)$.

2.2 Neural Network Architecture

The critic network outputs a scalar value that estimates a cost-to-go function of reinforcement learning. The actor network outputs action probabilities for a set of available actions. At the input layer of the networks (layer 1), the inputs can include kinematic variables, discrete variables associated with the current active controller, or metric values. The nonlinear activation a^l_j produced by $f(z)$ at each neuron for hidden

layers $l = 2, 3$ from Fig. 2.2 is given by

$$a_j^l = f(z_j^l) = f\left(\sum_{k=1}^{\overline{k}} \phi_{jk}^l a_k^{l-1} + b_j^l\right) \tag{2.1}$$

where ϕ_{jk}^l weighs the k^{th} activation a_k^{l-1} from the previous layer $l-1$ with the neural connection directed from the k^{th} neuron to the j^{th} neuron of the l^{th} layer in addition biased with value of b_j^l. Note, $l - 1 = 0$ corresponds to the input layer, in this case the activations are the inputs to the networks. In this work, we use the PReLU activation function a_j^l assigned from $f(z_j^l) = \max(0, z_j^l) + c\min(0, z_j^l)$ for some scalar variable parameter c, and set the number of neurons in $l = 2, 3$ to $\overline{k} = \overline{j}$. At the output layer of the networks, the critic outputs $\hat{S}_{\phi_c}(\mathbf{x})$ as an approximate value to the true cost-to-go function $S(\mathbf{x})$, while the actor outputs propabilities $\pi_{\phi_a}^{s_i} = g(\phi^3) = \frac{e^{\phi_{s_i}^3}}{\sum_s e^{\phi_s^3}}$, which are normalized values in the range (0,1).

2.3 Neural Network Weight Parameter Updates

The neural network outputs and updates of its parameters can be computed and efficiently by the use of the PyTorch open source machine learning framework [51]. The learning, i.e., the optimization of the network weights is implemented in two steps: (1) forward propagation of the input values that results in output values dependent on a set of weights ϕ and input $\mathbf{x}$ values as noted in Section 2.2; (2) backpropagation when the network weights are updated taking into account a loss function error. Our loss function or performance metric for the critic network is the mean squared error and the actor training minimizes the negative log-likelihood (for details see Chapter 5, expressions

(5.11)-(5.13)). Given the "loss" functions and using Autograd, the PyTorch automatic differentiation engine, we can compute the gradient of the loss with respect to the weight parameters $\frac{\partial loss}{\partial \phi}$ and $\frac{\partial loss}{\partial b}$, which is the goal of backpropagation. An optimizer is then used to update the weight parameters after the forward and backpropagation steps. The implementation of this in PyTorch requires key features such as the torch.tensor's, Autograd (torch.autograd), nn.Module, optimizer (torch.optim), and a loss function.

The torch.tensor is a fundamental building block in the manipulation of multi-dimensional arrays. We use the torch.tensor's for storing the values output from networks, for the algebraic manipulations, and for the inputs into our respective loss functions. More importantly, it allows tensors to be tracked in the Autograd tree structured computation graph by setting a gradient flag to true, i.e., requires_grad=True. This flag tells Autograd to track every operation on these tensors and by default the flag is set to true for network weight parameters in PyTorch. Thus, the output of the networks will also have the flags set to true since it tracked the operations done on the weight parameters. Subsequently, if the command torch.cat($\cdot$) is used to concatenate arrays as in lines 10-14 in Fig. 2.3, the resulting array will also have the flag set to true. The .backwards() function in line 19 of Fig. 2.3 points to the importance torch.tensors have by identifying the leaves of the tree and accumulating the gradients with respect to the graph leaves (weight parameters). This can be verified by using the x.grad_fn.next_functions function calls in the following snippet of example code

```
def print_graph(g, level=0):
```

```
    if g == None: return
    print('*'*level*4, g)
    for subg in g.next_functions:
        print_graph(subg[0], level+1)

print_graph(policy_loss.grad_fn, 0)
```

where .next_functions is recursively called tracing the graph backwards towards the tree leaves. The print_graph(·) would be placed after the call to .backward() to print the tree graph.

When creating the class to define the neural networks in PyTorch, the nn.Module class is inherited inside our network class. From this class, many important functions can be defined such as the forward(·) function, which allows data to be propagated forward from the input to the output of the neural network. This outputs a prediction value starting from some time point t and predicting a value at a next time point $t+\Delta t$. The next time point prediction relies heavily on the network parameters and on the optimizer parameters, i.e., learning rates, momentum, weight decays, etc. to find local or global minima.

In this work, we selected the Adam optimizer for its robustness since it uses both the mean and the variance to optimize our neural network weight parameters. Updates of the network parameters based on the Adam algorithm are shown in Fig. 2.2. The Adam algorithm combines ideas from both the stochastic gradient descent with

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Figure 2.2: [31] Adam algorithm used as the optimizer in this work.

momentum and Adadelta, where in the former it simply uses the gradient descent algorithm with the moving average and in the latter it uses the square of the gradient descent in the learning rate parameter. This is a major improvement over using a brute force method to compute the optimal weights for each parameter. Adam is computationally efficient, is not memory intensive, and handles noisy gradients. Its robustness is demonstrated by its moving average and a bias correction where the algorithm's initialization $m_0 = 0$ and $v_0 = 0$ is important in directing the gradients with no influence on any specific gradient g_t.

The loss function is of critical importance as this is where the learning algorithm must minimize the loss for the network weights to update its parameters. In the case of a classifier i.e. actor network, the negative log-likelihood "loss" function is used. On the other hand, the mean squared error (MSE) loss function is used for a cost function network, i.e., the critic network. Both work together finding solutions and

output action values for the loss function.

```
1. main():
2.     initialize bicycle or pendulum environment
3.     initialize actor and critic
4.     initialize Adam optimizers
5.     for NUM_EPOCHS:
6.         for BATCH_SIZE:
7.             state_values, action_log_probs, rewards, logits, episode_total_reward = play_episode(·)
8.             discounted_returns = get_discounted_returns(rewards, GAMMA, state_values,
               NUM_STEPS)
9.             advantages = discounted_returns - state_values
10.            epoch_weighted_log_probs = [epoch_weighted_log_probs, sum(action_log_probs*advantages)]
11.            epoch_logits = [epoch_logits, logits]
12.            epoch_state_values = [epoch_state_values, state_values]
13.            epoch_discounted_returns = [epoch_discounted_returns, discounted_returns]
14.            total_rewards = [total_rewards, episode_total_rewards]
15.        mean_entropy = get_entropy(epoch_logits)
16.        policy_loss = -mean(epoch_weighted_log_probs)
17.        critic_loss = mse_loss(epoch_state_values, epoch_discounted_returns)
18.        actor/critic.zero_grad()
19.        policy_loss/critic_loss.backward()
20.        adam_actor/critc.step()
```

Figure 2.3: Outline of the PyTorch code implemented for both the bicycle-gate environment and the pendulum environment. In the algorithm, data are collected and used in the loss function for the updates of the neural networks.

Following the works presented in Chapters 4 and 5, Fig. 2.3 gives psuedocode for the actor-critic network training implemented in PyTorch. In lines 2-4 the initialization takes place, which also includes the initialization of computed metrics and controllers that belong to the bicycle-gate environment. This initialization can be very memory intensive. Then for each BATCH_SIZE, in line 7 data is collected from

play_episode(·). The function play_episode(·) is where multiple trials are ran to reach the target set, while collecting data that will be returned to reduce the error of reaching this set. In line 8, the get_discounted_returns(·) function outputs a prediction of discounted returns over a horizon given by NUM_STEPS with GAMMA discounting factor. Lines 9-14 is code for storing data that is being setup for the computation of the mean entropy or the loss functions, with the exception of line 14, which is used for plotting the performance to reaching the target set. For each NUM_EPOCHS, Line 15 reports the confidence in the selected actions and lines 16-17 evaluate the loss functions. In line 18, we clear the networks gradients found in the previous iterations. Calling .backward() computes the gradients for this iteration and .step() updates the network parameters according to Adam algorithm.

2.4 Conclusions

With the efficiency and ease of use PyTorch provides, we have implemented an actor-critic neural network architecture and updated its parameters. Both the actor and critic networks have multiple layers with nonlinear activation functions for the learning of nonlinear functions. The learning is accomplished by a forward propagation of the input values to compute output values and the update of the weight parameters is done after the backward propagation of the error through the network starting from the last layer (output layer). The resulting weight parameters are updated based on the computed loss error according to the Adam algorithm. This framework allow us to

use these networks in a hierarchical control architecture for switching among multiple controllers.

Chapter 3

Stochastic Optimal Approach to the Steering of an Autonomous Vehicle through a Sequence of Roadways

- M. A. Carmona, A. A. Munishkin, M. Boivin, and D. Milutinović, "Stochastic optimal approach to the steering of an autonomous vehicle through a sequence of roadways," in 2019 American Control Conference (ACC), 2019, pp. 3279–3284.

3.1 Introduction

With an increasing interest in the use and deployment of self-driving cars, there is a pressing need to analyze and develop robust steering strategies for these automated vehicles. These strategies need to be able to steer an autonomous car in such a way that it smoothly guides itself and stays on the road even when the curves are unpredictable. To achieve that, we propose a stochastic optimal steering controller based on a bicycle

kinematics model. This nonholonomic model has been frequently used in the literature as an approximation of car kinematics [29, 50, 32].

We consider a road following scenario in which a car attempts to stay on a road as shown in Fig. 3.1. The road is composed of curvature-bounded road segments. At the beginning of each road segment, we attach a roadway, which is uniquely defined by its position, orientation and width. We assume that the road has a constant width, therefore the width of roadways is the same for all of them. If the roadway is depicted as a gate, then steering the car along the road is equivalent to steering the car through a sequence of gates. If the required sequence of roadways is known in advance, the problem of car steering can be considered as a path-planning problem [35]. However, in a general case, the sequence may not be known in advance due to a limited view of the road. An example of this is when a car follows a curved road which goes along the base of a wall or a cliff. The steering that we propose requires only information about the next roadway, but this information can be uncertain and may change as the car approaches it, specifically when the information about the orientation of the roadway is in question. To anticipate that uncertainty in our control design, we use the model in which each roadway's orientation changes unpredictably following a Brownian process. This type of model accounts for a curved road, as well as for the process of updates on the orientation of the next roadway as the car approaches it.

The problem of steering a vehicle in the presence of uncertain information about the road can be considered as an estimation problem. In that case, Kalman Filtering [29] or data fusion techniques[10] can be used to reduce the uncertainty as-

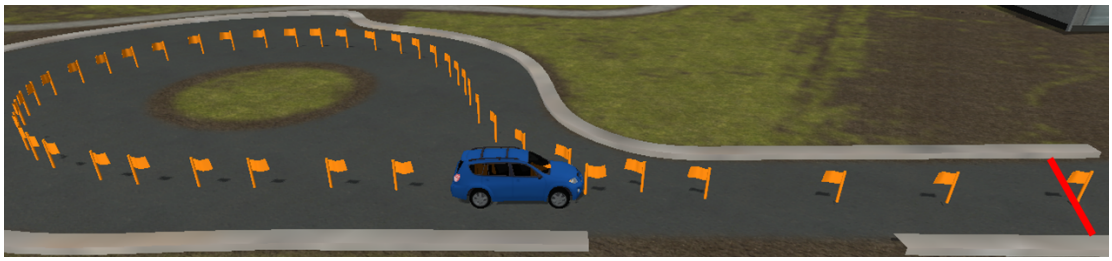


Figure 3.1: Stochastic optimal steering of a car. The figure is from the high-fidelity simulator [5] in which we implemented the steering. The steering implementation represents the road by a sequence of roadways. At every time point, the car is steered through the next roadway. Each roadway is represented by its center (orange flag), orientation and width. The first roadway in the sequence is depicted by the red line. The corresponding numerical data is in Fig. 3.8.

sociated with the sensory data and precisely estimate the steering relevant variables. Alternatively, the problem can be approached as a robust path-planning problem using various path-planning methods [35], such as Rapidly-exploring Random Trees [50] or Model Predictive Control strategies [32]. Finally, the problem can be considered as a problem of adaptive feedback control [62] or a machine learning problem for which the deep learning approach [9] can be used to learn the feedback control for steering.

The steering controller proposed in this paper is not only robust to the uncertainty, but is suitable for real time control applications dealing with very short time scales, such as driving. Once the controller is computed offline, it is stored in a lookup table and its values can be quickly accessed by a minimal amount of computations, which is in contrast with the amount of computations required for other methods, e.g.,

those based on path-planning approaches. Contrary to machine learning methods, the controller does not depend on training sets and has properties that can be proved on the principles of optimality and stochastic calculus. This approach has been previously used for various types of UAV navigation tasks [4, 3], and has been tested both with a UAV [39] and with differential drive robots [40].

The paper is organized as follows. Section 3.2 formulates the problem of steering a bicycle to enter a single roadway. The stochastic optimal control formulation of the steering problem is presented in Section 3.3 and its numerical solution is described in Section 3.4. In Section 3.5, the numerical solution is implemented to steer the bicycle through a sequence of roadways, as well as to steer a car to follow the road in the Anvel 3.0 simulation environment. Section 3.6 gives conclusions.

3.2 Problem Formulation

Let us consider a scenario in which a car drives on a road. If the road is divided in segments and each segment starts with a gate, i.e., with a roadway, then this scenario is equivalent to the one in which the car drives through a sequence of roadways (see Fig 3.1). Therefore, the *basic problem* is to steer the car to enter the next roadway taking into account that the roadway orientation is uncertain and that the information about it may change due to updates from the sensory system.

The car kinematics in this chapter is modeled by the kinematics of a back wheel drive bicycle, as depicted in Fig. 3.2. to describe the *basic problem* precisely, we

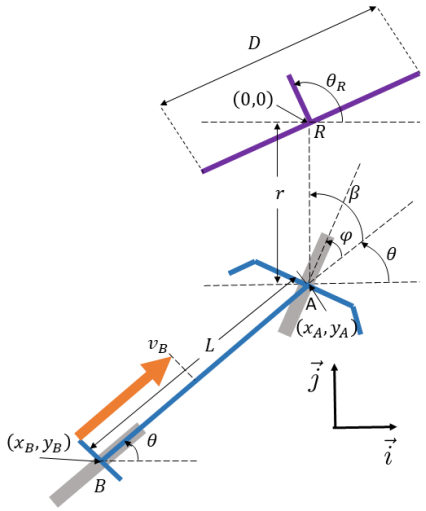


Figure 3.2: The car model is represented as a back wheel drive bicycle model. The center of the front wheel is located at A with its perpendicular steering handle and the center of the back wheel is located at B with its perpendicular bicycle pegs . The velocity vector shown as a solid arrow is applied at point B.

describe below the roadway and bicycle kinematics, and introduce relative coordinates that uniquely define the roadway-bicycle relative position.

Roadway: The roadway is a gate defined with a position width D and orientation angle, i.e., the heading angle θ_R (see Fig. 3.2). In our control design, the roadway heading angle uncertainty is anticipated by a Brownian, i.e., random walk process

$$d\theta_R = \sigma_R dw_R \tag{3.1}$$

where dw_R is the unit intensity Wiener increment [48].

Bicycle: The kinematics of a back wheel drive bicycle (see Fig. 3.2) is given by

$$dx_A = v_B(\cos\theta - \tan\phi \sin\theta)dt \tag{3.2}$$

$$dy_A = v_B(\sin\theta + \tan\phi \cos\theta)dt \tag{3.3}$$

$$d\theta = \frac{v_B}{L}\tan\phi dt \tag{3.4}$$

$$d\phi = udt \tag{3.5}$$

where the back wheel is driven by a constant velocity $v_B > 0$. The bicycle body length $L > 0$ is measured from A to B, θ is the heading angle of the bicycle, i.e., the bicycle heading, and ϕ is the steering angle. Since $\frac{\pi}{2}$ corresponds to the singular configuration of the bicycle kinematics in which the bicycle cannot move, then $\phi \in (-\frac{\pi}{2}, \frac{\pi}{2})$. The control variable u is the steering rate.

Relative coordinates: The coordinates are depicted in Fig. 3.2 and defined as

$$r^2 = x_A^2 + y_A^2,\ r \geq 0 \tag{3.6}$$

$$\beta = \arctan\left(\frac{y_A}{x_A}\right) - \theta,\ \beta, \theta \in (-\pi, \pi] \tag{3.7}$$

$$\alpha = \theta_R - \theta,\ \theta_R \in (-\pi, \pi] \tag{3.8}$$

where r is the distance between the steering wheel A and the roadway R, β is the bearing angle and α is the difference between the roadway and bicycle heading angles θ_R and θ, respectively.

Basic problem: Define the steering control variable u so that the bicycle enters the roadway. Entering the roadway corresponds to the roadway-bicycle relative position which belongs to the target set $\mathcal{T}$ defined as

$$\mathcal{T} = \{(r, \beta, \alpha, \phi) | r \leq R_{min}, -\beta_m \leq \beta \leq \beta_m, -\alpha \leq \alpha \leq \alpha_m, -\phi_m < \phi < \phi_m\} \tag{3.9}$$

The target set can be understood as a sector of a circle, where the circle has the center in the point R (Fig. 3.2) and the radius R_{min}. The sector starts with the angle $-\alpha_m$ and ends with α_m, which means that the bicycle enters the target set only if the heading θ is aligned with the heading of the roadway θ_R and the difference from it is in the range $[-\alpha_m,\alpha_m]$. However, in addition to it the bearing angle β has to be in the range $[-\beta_m,\beta_m]$, so that the bicycle's front wheel heads towards the roadway. Finally, at the point at which the bicycle enters the roadway, the steering angle ϕ has to be bounded to its range $(-\phi_m,\phi_m)$.

However, while the control u steers the bicycle towards the target set, the

bicycle should not enter the roadway from the wrong direction. Therefore, we also define the set of configurations that should be avoided $\mathcal{A}$ as

$$\mathcal{A} = \left\{(r,\beta,\alpha,\phi)| -\frac{\pi}{2} < \phi \leq -\phi_m \text{ or } \phi_m \leq \phi < \frac{\pi}{2}\right\}$$
$$\cup \{(r,\beta,\alpha,\phi)|r \leq R_{min}, (-\pi < \beta < -\beta_m \text{ or } \beta_m < \beta \leq \pi),$$
$$(-\pi < \alpha < -\alpha_m \text{ or } \alpha_m < \alpha \leq \pi)\} \quad (3.10)$$

The avoidance set $\mathcal{A}$ is defined based on the same circle as the target set $\mathcal{T}$, but its angle ranges are complementary to those in the target set. Fig. 3.3(a)-(c) shows three examples of the bicycle configurations which are not in the target set $\mathcal{T}$ and one (d) which is in the set. In summary, the *basic problem* is to define the steering control variable u, so that the bicycle enters the target set $\mathcal{T}$ and avoids the set $\mathcal{A}$.

3.3 Stochastic Optimal Control To Enter A Single Roadway

In this section, we formulate the *basic problem* as a minimum time stochastic optimal control problem. This is adequate since we know that for this type of problems, the optimal control solution is a state feedback controller and the time efficiency of steering is preferred.

Applying Îto calculus [48] to (3.6)-(3.8) and taking into account (3.1)-(3.5)

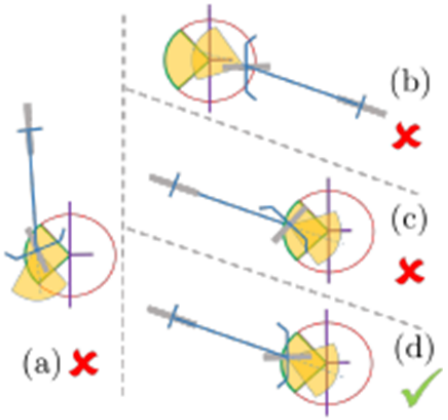


Figure 3.3: Illustration of the target set $\mathcal{T}$. The radius R_{min} is represented by the circle around the center of the roadway, and the yellow sectors are defined by limit angles α_m and β_m. If the bicycle (a) is not aligned with the roadway, (b) enters it from the opposite side, or (c) has the steering angle that is too big, the bicycle is not in the target set $\mathcal{T}$. The configuration example (d) is in the target set $\mathcal{T}$

yields the stochastic kinematics of the roadway-bicycle system

$$dr = b_r dt = -v_B[\cos\beta + \tan\phi \sin\beta]dt \tag{3.11}$$

$$d\beta = b_\beta dt = \left[\frac{v_B}{r}(\sin\beta - \tan\phi\cos\beta) - \frac{v_B}{L}\tan\phi\right]dt \tag{3.12}$$

$$d\alpha = b_\alpha dt + \sigma_R dw_R = -\left(\frac{v_B}{L}\tan\phi\right)dt + \sigma_R dw_R \tag{3.13}$$

$$d\phi = b_\phi dt = u dt \tag{3.14}$$

which we use to define the optimal control with the state space vector $\underline{x} = (r, \beta, \alpha, \phi)$, $r > 0$, $\beta, \alpha \in (-\pi, \pi]$, $\phi \in (-\frac{\pi}{2}, \frac{\pi}{2})$.

Let us introduce a control u dependent cost function

$$\mathcal{J}(u(t)) = E\left\{g(\underline{x}(t_f)) + \int_0^{t_f} 1dt\right\} \tag{3.15}$$

where $\underline{x}(t_f) \in \mathcal{T} \cup \mathcal{A}$ and E is the expectation operator and the evolution of $\underline{x}(t)$ is constrained to (3.11)-(3.14). The symbol t_f denotes a terminal time at which the state $\underline{x}(t)$ enters the target set $\mathcal{T}$ or the avoidance set $\mathcal{A}$ and $g(\underline{x}(t_f))$ is the terminal cost defined as

$$g(\underline{x}(t_f)) = \begin{cases} 0, & \underline{x}(t_f) \in \mathcal{T} \\ M, & \underline{x}(t_f) \in \mathcal{A} \end{cases} \tag{3.16}$$

The terminal cost includes a large penalty $M >> 0$ if the state $\underline{x}(t_f)$ enters the avoidance set $\mathcal{A}$ and there is no penalty for entering the target set $\mathcal{T}$. We formulate the steering optimal control as the control u which minimizes the cost function (3.15). This control steers the bicycle towards the target set $\mathcal{T}$ in the minimum expected time and avoids the set $\mathcal{A}$.

The optimal control u associated with minimizing the cost (3.15) can be found as the solution of the Hamilton-Jacobi-Bellman (HJB) equation [34]

$$\inf_u[\mathcal{L}^u V(\underline{x}) + 1] = 0, \ \underline{x} \notin \mathcal{T} \cup \mathcal{A} \tag{3.17}$$

with the boundary condition $V(\underline{x}) = g(\underline{x})$, $\underline{x} \in \mathcal{T} \cup \mathcal{A}$, where $\mathcal{L}^u$ is the second order differential operator

$$\mathcal{L}^u V = b_r \frac{\partial V}{\partial r} + b_\beta \frac{\partial V}{\partial \beta} + b_\alpha \frac{\partial V}{\partial \alpha} + b_\phi \frac{\partial V}{\partial \phi} + \frac{\sigma_R^2}{2} \frac{\partial^2 V}{\partial^2 \alpha} \tag{3.18}$$

in which b_r, b_β, b_α and b_ϕ are defined by (3.11)-(3.14), respectively. The HJB equation is a second order partial differential equation and we propose to compute the steering control using its numerical solution.

3.4 Numerical Stochastic Optimal Control

The numerical solution of (3.17) can be computed using a locally consistent Markov chain discretization [34]. The discretization approximates the original optimal control problem with an optimal control problem of a Markov chain with control dependent transition probabilities. Therefore, the numerical solution of (3.17) is approximated by a problem that can be solved over a discrete space using dynamic programming, i.e., the so-called value iterations [57]. In this case, the value iterations result in a discrete approximation of the value function V^h and optimal control u^{*h}, both of which are in the form of a four-dimensional lookup table. The superscript h indicates that the value function and control are computed for the discretized problem.

To discretize (3.17) in the state space, we use the discrete steps Δr, $\Delta\beta$, $\Delta\alpha$ and $\Delta\phi$ for the discretization of r, β, α and ϕ, respectively. The discretization is based on the upwind discrete approximation [34] of the derivatives of V in (3.17), and, for the sake of brief expressions, we introduce $\underline{x}^h = (r^h, \beta^h, \alpha^h, \phi^h)$. The discrete derivative

approximations are

$$\frac{\partial V}{\partial r} \approx \frac{b^+_{r^h}}{\Delta r}\Big(V^h(\underline{x}^h + \Delta r) - V^h(\underline{x}^h)\Big) - \frac{b^-_{r^h}}{\Delta r}\Big(V^h(\underline{x}^h) - V^h(\underline{x}^h - \Delta r)\Big) \tag{3.19}$$

$$\frac{\partial V}{\partial \beta} \approx \frac{b^+_{\beta^h}}{\Delta \beta}\Big(V^h(\underline{x}^h + \Delta \beta) - V^h(\underline{x}^h)\Big) - \frac{b^-_{\beta^h}}{\Delta \beta}\Big(V^h(\underline{x}^h) - V^h(\underline{x}^h - \Delta \beta)\Big) \tag{3.20}$$

$$\frac{\partial V}{\partial \alpha} \approx \frac{b^+_{\alpha^h}}{\Delta \alpha}\Big(V^h(\underline{x}^h + \Delta \alpha) - V^h(\underline{x}^h)\Big) - \frac{b^-_{\alpha^h}}{\Delta \alpha}\Big(V^h(\underline{x}^h) - V^h(\underline{x}^h - \Delta \alpha)\Big) \tag{3.21}$$

$$\frac{\partial V}{\partial \phi} \approx \frac{b^+_{\phi^h}}{\Delta \phi}\Big(V^h(\underline{x}^h + \Delta \phi) - V^h(\underline{x}^h)\Big) - \frac{b^-_{\phi^h}}{\Delta \phi}\Big(V^h(\underline{x}^h) - V^h(\underline{x}^h - \Delta \phi)\Big) \tag{3.22}$$

$$\frac{\partial^2 V}{\partial \alpha^2} \approx \frac{\sigma_D^2}{2\Delta \alpha^2}\Big(V^h(\underline{x}^h + \Delta \alpha) - V^h(\underline{x}^h)\Big) - \frac{\sigma_D^2}{2\Delta \alpha^2}\Big(V^h(\underline{x}^h) - V^h(\underline{x}^h - \Delta \alpha)\Big) \tag{3.23}$$

where $b^+_{r^h} = \max[0, b_{r^h}]$, $b^-_{r^h} = \max[0, -b_{r^h}]$ and $b^+_{\beta^h}$, $b^-_{\beta^h}$, $b^+_{\alpha^h}$, $b^-_{\alpha^h}$, $b^+_{\phi^h}$ and $b^-_{\phi^h}$ are defined in the same way. The superscript h indicates terms that are evaluated at the points of the discretized state space in which $r^{h+1} - r^h = \Delta r$, $\beta^{h+1} - \beta^h = \Delta \beta$, $\alpha^{h+1} - \alpha^h = \Delta \alpha$, $\phi^{h+1} - \phi^h = \Delta \phi$. After the substitution of (3.18)-(3.23) in (3.17), we

move all the terms that include $V^h(\underline{x}^h)$ to the left side of expression (3.17) to obtain

$$\begin{aligned} V^h(\underline{x}^h) = \inf_u \big\{ & \Delta t_u^h + \\ & p^+_{\Delta r,u} V^h(\underline{x}^h + \Delta r) \ + p^-_{\Delta r,u} V^h(\underline{x}^h - \Delta r) + \\ & p^+_{\Delta \beta,u} V^h(\underline{x}^h + \Delta \beta) + p^-_{\Delta \beta,u} V^h(\underline{x}^h - \Delta \beta) + \\ & p^+_{\Delta \alpha,u} V^h(\underline{x}^h + \Delta \alpha) + p^-_{\Delta \alpha,u} V^h(\underline{x}^h - \Delta \alpha) + \\ & p^+_{\Delta \phi,u} V^h(\underline{x}^h + \Delta \phi) + p^-_{\Delta \phi,u} V^h(\underline{x}^h - \Delta \phi) \big\} \end{aligned} \tag{3.24}$$

where

$$p^{\pm}_{\Delta r,u} = \Delta t_u^h \left(\frac{b^{\pm}_{r^h}}{\Delta r} \right), \qquad p^{\pm}_{\Delta \beta,u} = \Delta t_u^h \left(\frac{b^{\pm}_{\beta^h}}{\Delta \beta} \right)$$

$$p^{\pm}_{\Delta \alpha,u} = \Delta t_u^h \left(\frac{b^{\pm}_{\alpha^h}}{\Delta \alpha} + \frac{\sigma_R^2}{2\Delta \alpha} \right), \qquad p^{\pm}_{\Delta \phi,u} = \Delta t_u^h \left(\frac{b^{\pm}_{\phi^h}}{\Delta \phi} \right)$$

can be interpreted as discrete Markov-chain transition probabilities from the points $(r^h \pm \Delta r, \beta^h \pm \Delta \beta, \alpha^h \pm \Delta \alpha, \phi^h \pm \Delta \phi)$ of the discrete space to the point $(r^h, \beta^h, \alpha^h, \phi^h)$. The symbol Δt_u^h is the interpolation time interval, which is locally consistent with the transition probabilities [34] and can be expressed as

$$\Delta t_u^h = \left(\frac{|b_r^h|}{\Delta r} + \frac{|b_\beta^h|}{\Delta \beta} + \frac{|b_\alpha^h|}{\Delta \alpha} + \frac{|b_\phi^h|}{\Delta \phi} + \frac{\sigma_R^2}{(\Delta \alpha)^2} \right)^{-1} \tag{3.25}$$

where $|b_r^h| = b^+_{r^h} + b^-_{r^h}$, $|b_\beta^h| = b^+_{\beta^h} + b^-_{\beta^h}$, $|b_\alpha^h| = b^+_{\alpha^h} + b^-_{\alpha^h}$ and $|b_\phi^h| = b^+_{\phi^h} + b^-_{\phi^h}$. To summarize, expression (3.24) is the discrete version of (3.17) and the discrete approximation V^h of the value function V can be solved numerically using the value iterations [57] starting from an initial guess for the $V^h(\underline{x}^h)$ values.

For the computational domain $\mathcal{K}$ of the numerical solution, we use

$$\mathcal{K} = \{[R_{min} - \delta, R_{max}] \times [-\pi, \pi - \Delta\beta] \times [-\pi, \pi - \Delta\alpha] \times [-\phi_m, \phi_m]\} \tag{3.26}$$

and, in our numerical example, we use $R_{min} = 1.4$m, $\delta = 1.2$m, $R_{max} = 30.2$m and $\phi_m = 27\pi/180$. The value of δ defines the smallest value r that is taken into account for the numerical solution, and in this case this value is 0.2m. In our problem formulation, the angles β and α have a full $(-\pi, \pi]$ range, which corresponds to the computational domain being periodic along those states. Thus, all pairs of points $(r^h, -\pi, \alpha^h, \phi^h)$ and $(r^h, \pi - \Delta\beta, \alpha^h, \phi^h)$, as well as $(r^h, \beta^h, -\pi, \phi^h)$ and $(r^h, \beta^h, \pi - \Delta\alpha, \phi^h)$ are next to each other, respectively. The discrete steps are $\Delta r = (R_{max} - R_{min} + \delta)/100$, $\Delta\beta = \Delta\alpha = 5\pi/180$ and $\Delta\phi = 9\pi/180$. In our problem formulation, the car stops once it has reached either the target set $\mathcal{T}$ or the avoidance set $\mathcal{A}$ defined in (3.9) and (3.10), respectively. Therefore, for R_{min}, we use the absorbing boundary condition $V^h(R_{min}, \beta^h, \alpha^h, \phi^h) = 0$ or $V^h(R_{min}, \beta^h, \alpha^h, \phi^h) = M$, which depends on the state $(R_{min}, \beta^h, \alpha^h, \phi^h)$ belonging to the target set $\mathcal{T}$, or the avoidance set $\mathcal{A}$, respectively. The boundary condition M also applies for any state with $\phi^h = \pm\phi^h_m \mp \Delta\phi$ since that boundary belongs to the avoidance set $\mathcal{A}$. For this problem, we are free to choose the boundary condition for states with R_{max} and we choose the reflective boundary conditions, i.e., $V^h(R_{max}, \beta^h, \alpha^h, \phi^h) = V^h(R_{max} - \Delta r, \beta^h, \alpha^h, \phi^h)$. A general explanation of this method for the control of nonholonomic vehicles is in [19]. This method was also used for target tracking problems [4].

Result: Fig. 3.4 shows the result of a numerical simulation of the stochastic optimal control for steering the bicycle (3.2)-(3.5) with the simulation step of 0.1 sec. The bicycle back wheel velocity $v_B = 11.18$m/s, the length $L = 2.66$m and the roadway width $D = 2.8$m. The uncertainty of the roadway heading is modeled with $\sigma_R = 5\pi/180$,

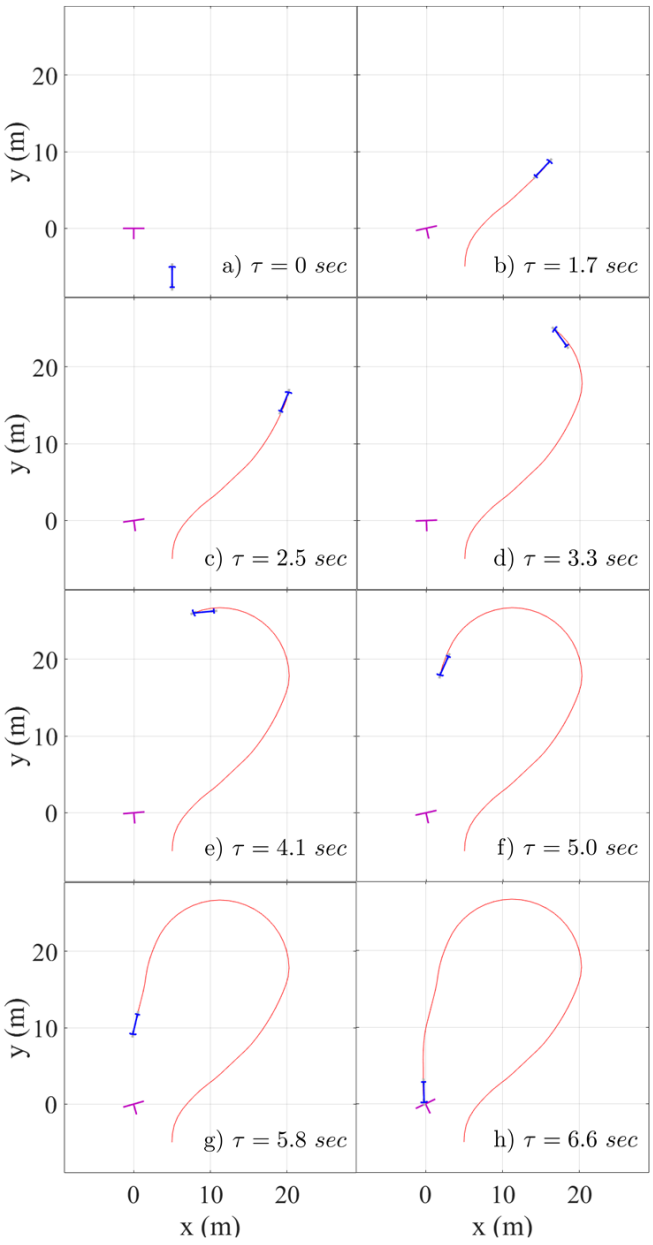


Figure 3.4: Optimal steering to enter the roadway. Initially, the bicycle starts with the front wheel at $(5, -5)$, facing in the opposite direction of the roadway which has the heading angle $\theta_R = -\pi/2$. The motion of the bicycle and the roadway is captured in the (a)-(h) panels.

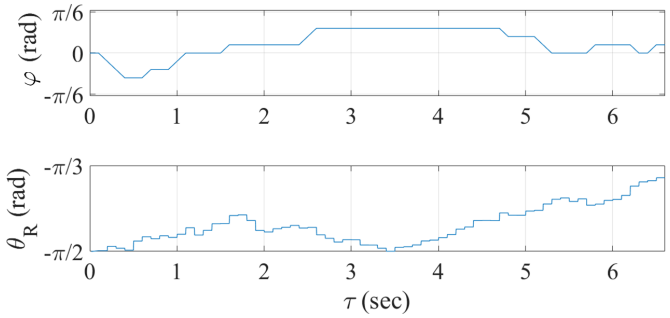


Figure 3.5: Plot showing the car's steering (top) progression as the roadway changes in its orientation (bottom) from $-\pi/2$ towards $-\pi/3$. This progression corresponds to the motion captured in Fig. 3.4.

see expression (3.1). The target set $\mathcal{T}$ and the avoidance set $\mathcal{A}$ used in the simulation are based on $\beta_m = 80\pi/180$, $\alpha_m = 60\pi/180$ and the values used in the definition of the computational domain $\mathcal{K}$ in expression (3.26). The value that is used to penalize for the states that should be avoided is $M = 10^4$.

Fig. 3.4 shows that the bicycle that starts from the position in which it faces the roadway from the opposite direction enters the roadway after $\tau = 6.6$sec. While the bicycle steers toward the roadway, the roadway heading changes stochastically in periodic time intervals. Fig. 3.5 shows the time plots of the steering angle and the roadway heading from the simulation.

3.5 Sequence of Roadways

Using the stochastic optimal control, computed in the previous section, we showed that we were able to steer the back wheel drive bicycle in the minimum expected

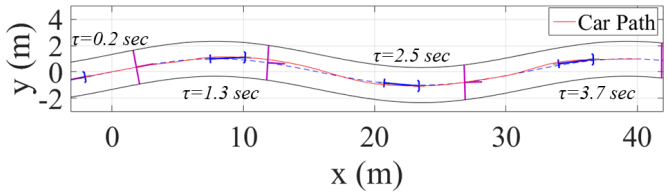


Figure 3.6: Bicycle following the sinusoidal road, $\sin(cx)$, $c = 0.2$. This figure depicts the bicycle trajectory (solid line) and positions at different times as it reaches the roadways in sequence. The road is depicted with the dashed line along its center.

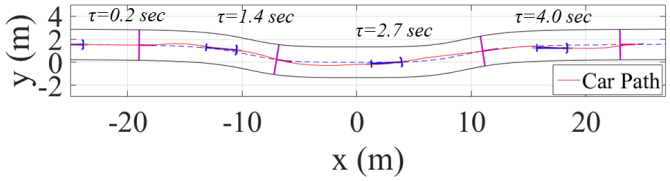


Figure 3.7: Bicycle following the $\arctan(cx^4)$ road, $c = 10^{-4}$. This figure depicts the bicycle trajectory (solid line) and positions at different times as it reaches the roadways in sequence. The road is depicted with the dashed line along its center.

time to the roadway. To follow a sequence of roadways, we use the same controller in the following way: we steer the bicycle to the nearest roadway and once it nearly enters the roadway, we switch to steering to the next roadway in the sequence and Figs. 3.6 and 3.7. This steering is also implemented on a car in the high-fidelity real-time simulator Anvel 3.0 and the simulation data are plotted in Fig. 3.8.

In the case of the sinusoidal ($\sin(cx)$, $c = 0.2$) and $\arctan(cx^4)$, $c = 10^{-4}$ roads, the optimal controllers are computed for $v_B = 11.18$ m/s. For the circular road shown in Figs. 3.1 and 3.8, the optimal controller is computed for $v_B = 4$ m/s. For these three cases, all other parameters are the same as in *Result* of Section 3.3.

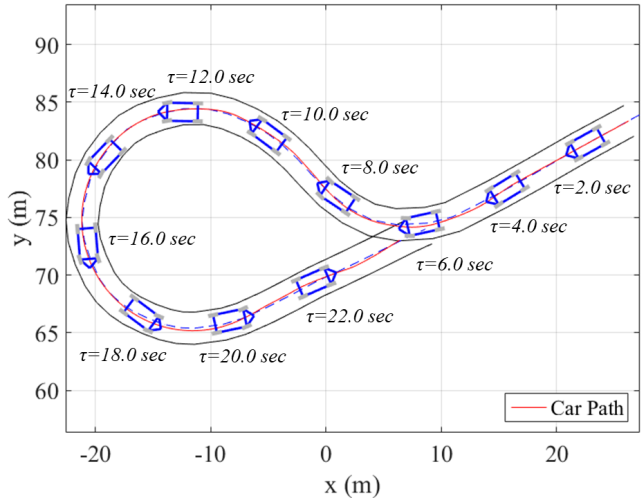


Figure 3.8: The car trajectory generated in Anvel 3.0 is of entering and exiting a cul-de-sac. The road begins at the coordinates $(26, 83)$ and ends at the coordinates $(6, 73)$ after completing the U-turn.

Results: The sequence of roadways to which we apply the optimal steering results in the trajectories shown in Figs. 3.6-3.8. The bicycle following the sequence of roadways is depicted with the solid line and the center of the roads is depicted with the dashed line. In Fig. 3.8, showing the result from the Anvel 3.0 simulation, the car is depicted with the polygon.

As discussed in [2], the biggest overshoot or undershoot, i.e., the misalignment of the vehicle's position with respect to the center of the road occurs when the vehicle steers along a curve. In our results, the biggest misalignment is observed for the $\arctan(cx^4)$ road, where the bicycle has to travel along steep curves located at approximately $x = -10$ and $x = 10$ of Fig. 3.7. In the bicycle results, the speed is higher and roadways are separated further apart (see Figs. 3.6 and 3.7) compared to that the

car simulation in Anvel 3.0. The slower car speed with closer roadways diminishes the misalignment of steering along curves as shown in Fig. 3.8.

3.6 Conclusions

In this work, we presented the stochastic optimal controller that steers the back wheel drive bicycle to enter a fixed roadway with uncertainty in its heading. This uncertainty was anticipated in the design of the bicycle steering control using the random walk process for the roadway heading. Consequently, the roadway-bicycle system kinematics is stochastic and we used it to design the minimum time stochastic optimal controller for steering the bicycle. With the goal of following various types of roads, the designed controller was implemented on the sequence of roadways, where each roadway describes the beginning of a road segment. Finally, this algorithm was implemented for steering the car in the high-fidelity car simulator Anvel 3.0. Future work will be in the direction of an uncertainty anticipating control for both steering angle and velocity of the car.

Chapter 4

Metrics-only Training Neural Network for Switching among an Array of Feedback Controllers for Bicycle Model Navigation

- M. A. Carmona, A. D. Milutinović, and A. Faust, "Metrics-only Training Neural Network for Switching among an Array of Feedback Controllers for Bicycle Model Navigation," accepted to 2022 American Control Conference.

4.1 Introduction

We investigate in this paper a novel method for a neural network approach to switching among an array of off-line computationally generated (CG) stochastic optimal feedback controllers. These controllers can be computed off-line together with controller

associated performances, i.e., metrics. The metrics account for all properties of control problems solved by each CG controller, and our hypothesis is that the metrics alone can be sufficient to train the neural network to implement a policy for switching among the controllers.

Motivated by autonomous vehicles and mobile robot systems, our study is based on a minimal bicycle kinematic model, in which the bicycle is tasked to follow a road which is described by a sequence of gates, see Fig. 4.1. The discrete variable q, i.e., the mode, is associated with a current velocity v_q of the bicycle, i.e., the corresponding controller. *The goal of the neural network training is to obtain a policy for switching among the modes to safely traverse the gates and perform forward-backward motions any time the safety of traverse is compromised due to a sharp turn.*

The figure depicts that the neural network is trained based on the mode q and a set of continuous variables, which can be either *kinematic variables* (KV), or *lookup-table metrics* (LTM). The KV based training represents a standard approach to the training and does not account for available controller performances. On the other hand, the LTM approach uses solely the continuous variables coming from lookup-table stored metrics of control performances.

The control architecture depicted in Fig. 4.1 is a version of hierarchical control architectures that have been studied both in control and robotics [52]. This type of systems is also an example of *hybrid* or *supervisory* control systems [33] because of the presence of both continuous system dynamics and the discontinuous discrete switching signal. It is well known that the switching among stable controllers can result in the

instability of the overall system [38, 12, 36], but it can also contribute to performance improvements [46, 37]. A performance signal based switching for mobile robots has been considered in [26] and the optimality of switching has been considered in [7] and [41]. To switch between the navigation to a goal and obstacle avoidance in [58], the authors used Lyapunov functions, while in [43], the authors considered a stochastic problem and used a rule for the switching that is based on a probability inequality.

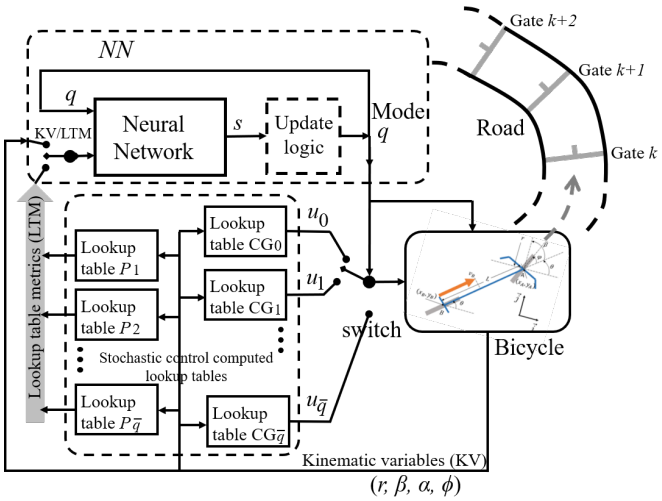


Figure 4.1: A bicycle following a road described by a sequence of gates. Each controller CG_i, $i = 1, 2, ..., \bar{q}$ is designed to navigate the bicycle to the next gate with forward velocity v_i, and has its corresponding performance P_i lookup table. The controller CG_0 navigates the bicycle in the reverse direction with velocity v_0 and it does not have a corresponding performance for reaching the gate. The update logic block in the figure facilitates that modes $q = i$, $i = 0, 1, 2, ..., \bar{q}$ can be only updated in a certain order. The switch KV/LTM defines the inputs to be used by the neural network.

The work presented here is aligned with hierarchical approaches to mobile

robot navigation that use learning [18, 22, 59] or stochastic control [45, 44], as well as the emergence of data-driven approaches for learning rules for the switching among multiple controllers [1, 27] or options [55, 56, 8]. The options in our work are CG controllers computed using the numerical stochastic optimal control [34]. We use the same numerical method to compute the optimal control metrics (LTM) for each option, which are expected bicycle paths associated with each controller. Once we have obtained CG feedback controllers, we use reinforcement learning and the actor-critic [53, 8] neural network to learn the rule for switching among the controllers. The neural network training is performed under the assumption that the rule is applied at every time step, in other words, we deal with *single-step options* [55]. In the case when the training is based on metrics, the switch in Fig. 4.1 is in the LTM position. If the training is based on kinematic variables, similar to [56], the switch is in the KV position.

The contribution of the presented work is (1) that it shows that control performance metrics are sufficient to learn a switching policy. Moreover, from the comparison between the LTM and KV training approaches, (2) the LTM training has a faster convergence with less variations than the KV based training.

In Section 4.2, we provide the problem formulation of switching among multiple CG controllers for the motivating bicycle driving scenario. This is followed by the description of kinematic variables in Section 4.3. Section 4.4 gives the actor-critic neural network and its training process. The results are presented in Section 4.5 and conclusions are given in Section 4.6.

4.2 Problem Formulation

The problem formulation is based on an array of minimum time, stochastic optimal steering rate feedback controllers $u_i = u_i(\mathbf{x})$, $i \in Q = \{1, 2, ...\overline{q}\}$ presented in Fig. 4.1 as CG_i, where $\mathbf{x} = [r, \alpha, \beta, \phi]$ is a vector of kinematic variables defining the relative position between the bicycle and a gate as depicted in Fig. 3.2. Each controller is computed to navigate the *front-wheel driven bicycle* through a stochastic gate with a constant forward velocity $v_i \in \mathcal{V} = \{v_1, v_2, v_3, ...v_{\overline{q}}\}$, $v_i > 0$, $v_{i+1} > v_i$, i, $i + 1 \in Q$. Due to the limited space, we point the reader to our previous work on *back-wheel driven bicycle* [15] which describes the computation of stochastic optimal control $u_i(\mathbf{x})$ for a constant velocity v_i, as well as to [43] which describes the use of the backward Kolmogorov (BK) equation to compute the numerical approximations of minimum time stochastic optimal controller expected times T_i to enter a target set, which in this case corresponds to the gate. From the latter, we can compute the expected path $P_i = v_i T_i$ for the bicycle to enter the gate. Under the problem formulation, we assume here that both controllers $u_i = u_i(\mathbf{x})$, $i \in Q$ and the corresponding expected paths $P_i = P_i(\mathbf{x})$ are computed off-line using the numerical method in [34] and stored in lookup tables.

While each controller can navigate the bicycle through a next gate, if the next gate is not positioned well for the safe traversal at a given speed, the corresponding controller navigates the bicycle on a loop-like trajectory before safely traversing the gate. This type of trajectory helps re-positioning the bicycle for the safe traversal of the next gate, but is undesirable when a gate sequence is associated with a road. A

much preferred approach to the safe traversal of the next gate is that the bicycle reduces its speed, and if necessary performs forward-backward bicycle motions until the bicycle can safely traverse the next gate. For this reason, we define the velocity for moving backward $v_0 = -v_1$ and the set of all velocities is $\mathcal{V}^+ = \mathcal{V} \cup \{v_0\}$.

For the backward motion $i = 0$, we define that the steering rate is $u_0 = u_{\overline{q}}$, but using a different steering rate definition is possible. It is because the control u_0 for moving backward cannot be formulated naturally as an optimal control to reach the gate while the bicycle faces it with its front wheel. For the same reason, in Fig. 4.1 we do not have the controller u_0 corresponding metrics. We can now introduce an integer signal $q(t) \in Q^+$, $Q^+ = Q \cup \{0\}$. At any time t, the value of $q(t) = i, i \in Q^+$ implies that the velocity of the bicycle is v_i and the steering control law is u_i. The signal $q(t)$ updates with a sample time $\Delta t = t_k - t_{k-1}$ as

$$q(t_k) = q(t_{k-1}) + s_k, \quad s_k \in \begin{cases} \{0, 1\}, & q(t_{k-1}) = 0 \\ \{-1, 0\}, & q(t_{k-1}) = \overline{q} \\ \{-1, 0, 1\}, & otherwise \end{cases} \tag{4.1}$$

where s_k is a switching control variable providing that any change of the velocity follows the sequence of velocities and does not go over the lower v_0 and upper $v_{\overline{q}}$ bounds.

We consider here a problem of learning a feedback policy for s_k providing that the bicycle goes through the gate without any loops. The LTM training based policy explicitly depends only on the vector of the lookup-table outputs $\mathbf{L} = \mathbf{L}_{LTM}$ and $q(t_{k-1})$, i.e.,

$$s_k = s_k(q(t_{k-1}), \mathbf{L}). \tag{4.2}$$

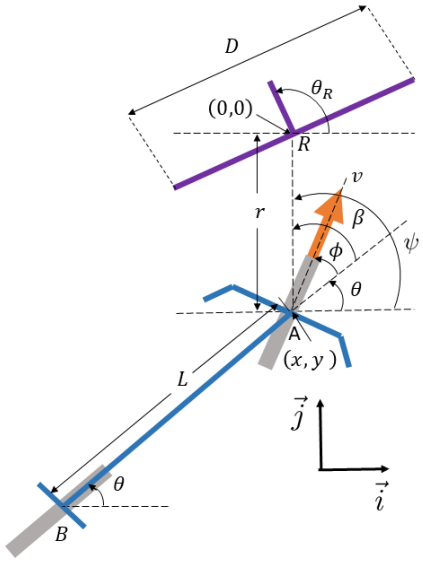


Figure 4.2: A bicycle-stochastically revolving gate $\mathcal{R}$ configuration: the center of the front wheel is located at A with its perpendicular steering handle and the center of the back wheel is at B with its perpendicular bicycle pegs. The velocity vector (shown as the solid arrow) is associated with point A.

For the LTM base policy, the vector $\mathbf{L}_{LTM}$ is composed of normalized values of lookup-table stored expected paths P_i

$$\mathbf{L}_{LTM} = \left[\frac{P_1}{\overline{P}_1}, \frac{P_2}{\overline{P}_2}, ... \frac{P_{\overline{q}}}{\overline{P}_{\overline{q}}}\right], \tag{4.3}$$

where $\overline{P}_i$ is the maximal expected path in the lookup table associated with the controller i, i.e., $\overline{P}_i = \max_{\mathbf{x}} P_i(\mathbf{x})$. The use of normalized continuous variables is typical in the training of neural networks.

For the purpose of comparison, in this work we also train a neural network to find the policy that depends on kinematic variables. In that case, the policy is trained

with $\mathbf{L} = \mathbf{L}_{KV}$ defined as

$$\mathbf{L}_{KV} = \left[\frac{r - \underline{r}}{\overline{r} - \underline{r}}, \frac{\alpha}{\pi}, \frac{\beta}{\pi}, \frac{\phi}{\pi}\right], \tag{4.4}$$

which is composed of normalized values of the kinematic variables. The distance $r \in [\underline{r}, \overline{r}]$ is normalized with the length of its range, while angular kinematic variables are normalized with π.

4.3 Bicycle-Gate Kinematic Variables

The bicycle and the gate which is labeled with $\mathcal{R}$ are depicted in Fig. 4.2. The bicycle kinematic describes the motion of the front-wheel pivot point A as depicted in Fig. 4.2 and under the assumption that the current bicycle velocity $v_q(t)$ is associated with the front-wheel velocity and aligned with the front-wheel heading. The kinematic of such a *front-wheel driven bicycle* is

$$dx = v_{q(t)} \cos(\phi + \theta) dt, \quad dy = v_{q(t)} \sin(\phi + \theta) dt \tag{4.5}$$

$$d\theta = \frac{v_{q(t)}}{L} \sin\phi dt, \quad d\phi = u_{q(t)}\ dt, q(t) \in Q^+, \tag{4.6}$$

where $q(t)$ is the integer signal (4.1) which is updated with the sample time Δt_k, ϕ is the steering angle, and θ is the heading angle which is aligned with the bicycle frame as depicted in Fig. 4.2. The steering angle is measured starting from the heading direction and is bounded, i.e., $\phi \in (-\frac{\pi}{2}, \frac{\pi}{2})$, while $\theta \in (-\pi, \pi]$. The variable $L > 0$ in (5.26) is the distance between the front wheel A and back wheel B (see Fig. 4.2). The kinematic model (4.5)-(4.6) has two control variables. The first one is $q(t)$ and the second one is $u_{q(t)}$ which is the steering rate control variable for the velocity $v_{q(t)}$.

The bicycle navigates towards $\mathcal{R}$, and the bicycle-gate relative coordinates are the distance r, the bearing angle β and the misalignment angle α between the bicycle heading and gate orientation θ_R defined as

$$r^2 = x^2 + y^2, \quad \alpha = \theta_R - \theta, \quad \beta = \arctan_2\left(\frac{y}{x}\right) - \theta. \tag{4.7}$$

To address the uncertainty of gate orientation as perceived by the gate approaching bicycle and obtain a robust navigation strategy, we model the gate orientation angle as $d\theta_R = \sigma_R dw_R$, where dw_R is the unit intensity Wiener process increment and σ_R is a scaling factor. From this and the front-wheel driven bicycle kinematic (4.5)-(4.6), Itô's formula [48] yields the following stochastic kinematic for the front-wheel driven bicycle-gate relative position

$$dr = -v_{q(t)} \cos(\beta - \phi)\ dt \tag{4.8}$$

$$d\alpha = -\frac{v_{q(t)}}{L} \sin\phi\ dt + \sigma_R\ dw_R \tag{4.9}$$

$$d\beta = \frac{v_{q(t)}}{r} \sin(\beta - \phi)\ dt - \frac{v_{q(t)}}{L} \sin\phi\ dt \tag{4.10}$$

$$d\phi = u_{q(t)}\ dt. \tag{4.11}$$

Using the kinematic variables $\mathbf{x} = [r, \alpha, \beta, \phi]$, we can define a set $\mathcal{G}$ which is the *target set* defined as

$$\mathcal{G} = \{\mathbf{x} |\ r \leq \underline{r},\ |\beta| \leq \underline{\beta},\ |\alpha| \leq \underline{\alpha},\ |\phi| \leq \underline{\phi}\}, \tag{4.12}$$

where $|\cdot|$ denotes absolute values and $\underline{r} < D$. We assume that when $\mathbf{x} \in \mathcal{G}$, the bicycle passes safely through the gate. We also introduce a set $\mathcal{A}$ which includes all

configurations $\mathbf{x}$ in which the bicycle misses the gate defined as

$$\begin{aligned}\mathcal{A} =&\Big\{\mathbf{x} \mid -\pi < \phi \leq -\underline{\phi} \text{ or } \underline{\phi} \leq \phi \leq \pi\Big\} \\ &\cup \Big\{\mathbf{x} \mid r \leq \underline{r}, (-\pi < \beta < -\underline{\beta} \text{ or } \underline{\beta} < \beta \leq \pi), \\ &\qquad (-\pi < \alpha < -\underline{\alpha} \text{ or } \underline{\alpha} < \alpha \leq \pi)\Big\}.\end{aligned} \tag{4.13}$$

Both set $\mathcal{G}$ and set $\mathcal{A}$ are in use for computations of the optimal controllers $u_i = u_i(\mathbf{x})$ corresponding to velocities v_i, $i \in Q$. Finally, we define a set $\mathcal{L} = \{\mathbf{x}|\ r > \underline{r}, |\alpha| > \overline{\alpha}\}$. This set is the one with a large misalignment angle at a large distance, which is an indicator that the bicycle trajectory goes along a trajectory with a loop. The set $\mathcal{L}$ together with sets $\mathcal{G}$ and $\mathcal{A}$ is in use for the neural network training to detect a successful traversal of the gate $\mathbf{x} \in \mathcal{G}$, or unsuccessful traversal $\mathbf{x} \in \mathcal{A} \cup \mathcal{L}$.

In our example from Section 4.5, we use the following parameters

$$\begin{aligned}&L = 1.0m,\ \underline{r} = 0.3m,\ \overline{r} = 10m, D = 1m \\ &\underline{\alpha} = 50\tfrac{\pi}{180},\ \overline{\alpha} = 135\tfrac{\pi}{180}, \underline{\beta} = 80\tfrac{\pi}{180}\ \underline{\phi} = 50\tfrac{\pi}{180},\end{aligned} \tag{4.14}$$

where the lengths are measured in m and angles in rad. The array defining the sequence of bicycle velocities is

$$\mathcal{V}^+ = \{-0.6, 0.6, 0.8, 1.0, 1.2, 1.4\}(m/s), \tag{4.15}$$

where v_0 corresponds to the reverse velocity at the minimum forward velocity, i.e., $v_0 = -v_1$. The maximal velocity is $1.4m/s$.

4.4 Actor-Critic Neural Network

To learn the policy, we propose to use an actor-critic neural network. The network learns the stochastic policy $\pi_s : \{-1, 0, 1\} \times Q \times \mathbf{R}^{|\mathbf{L}|} \rightarrow \mathbf{R}$ which returns probabilities for possible actions $s \in \{-1, 0, 1\}$ for the current discrete state q and vector of variables $\mathbf{L}$ used in the training. The symbol $|\mathbf{L}|$ denotes the size of the vector $\mathbf{L}$. The policy at the time step k can be also denoted as $\pi_s(s, q_{k-1}, \mathbf{L}(k))$ with the understanding that at the step k, the current discrete state is the one selected in the previous time step. Once we find the optimal stochastic policy π_s^*, we define the update policy for the step k as $s_k = \arg\max_s \pi_s^*(s, q_{k-1}, \mathbf{L}(k))$, where $s \in \{-1, 0, 1\}$ denotes available actions and q_{k-1} is the discrete state at the time step k before the action is taken. For the KV based training, $\mathbf{L} = \mathbf{L}_{KV}(k)$ and for the LTM based training, $\mathbf{L} = \mathbf{L}_{LTM}(k)$.

The stochastic policy that we learn is based on the maximization

$$\pi_s^* = \arg\max_{\pi_s} E\{\gamma^K R(\mathbf{x}(t_K)\}), \quad \gamma = e^{-c\Delta t}, c > 0, \tag{4.16}$$

where t_K is the discrete time point K at which $\mathbf{x}(t_K)$ enters the terminal set $\mathcal{T} = \mathcal{G} \cup \mathcal{L}$ and $R(\mathbf{x}(t_K))$ is the terminal reward

$$R(\mathbf{x}) = \begin{cases} M, & \mathbf{x} \in \mathcal{G} \\ -M, & \mathbf{x} \in \mathcal{L}. \end{cases} \tag{4.17}$$

The subset $\mathcal{G}$ corresponds to the bicycle poses at which it reaches the gate and the set $\mathcal{L}$ corresponds to the poses at which the bicycle enters a loop trajectory. The overall term under the expectation E is maximized when the policy navigates the bicycle to go

through the gate and the term γ^K favors shorter trajectories for entering the gate.

A. Training Data Collection: The neural network training is based on the numerical simulations of (4.5)-(4.6). Each episode of the training starts with a random selection of the initial relative position between the bicycle and the gate $\mathbf{x}_0 = [r_0, \beta_0, \alpha_0, \phi_0]$. The initial values are defined as

$$\beta_0 = \psi_0 - \theta_0,\ \alpha_0 = \xi_R - \theta_0,\ \phi_0 = 0. \tag{4.18}$$

r_0 and the variables on the right side of the expressions above are independent random variables

$$\begin{aligned} &r_0 \sim U(0, \overline{p}/2),\ \psi_0 \sim U\left(-\tfrac{\pi}{4}, \tfrac{\pi}{4}\right), \\ &\theta_0 \sim U\left(-\tfrac{50\pi}{180}, \tfrac{50\pi}{180}\right), \xi_R \sim \mathcal{N}\left(0, \left(\tfrac{0.1\pi}{180}\right)^2\right), \end{aligned} \tag{4.19}$$

where $U(a, b)$ denotes a uniform random variable distribution in the range $[a, b]$, and $\mathcal{N}(\mu, \sigma^2)$ denotes a Gaussian distribution with a mean value μ and variance σ^2. For the distribution of r_0, the upper bound includes $\overline{p} = 2\pi L/\tan(\phi_{max})$, $\phi_{max} = 50\pi/180$, which is the turning radius of the back wheel for the maximal steering angle ϕ. The specific value is not of critical importance, but we select a value compatible to the scale and the turning radius of the bicycle.

From the initial position, the bicycle follows the current policy π. We perform the simulation with the time step $\Delta t = 0.1s$. Each episode lasts until the bicycle reaches the gate, which is a successful episode, or until we detect that the bicycle starts going on a loop, which is an unsuccessful episode. If K_T is the time step in which the episode terminated, then the successful episode corresponds to $\mathbf{x}_{K_T} \in \mathcal{G}$ and the unsuccessful one corresponds to $\mathbf{x}_{K_T} \in \mathcal{L}$. In the second case, we allow for up to a maximum of 35

new episodes until we obtain a successful episode. In our training, we define an *epoch* as a set with 25 successful episodes. Therefore, the number of episodes in an epoch can be between 25 and $25 \cdot 35 = 875$ episodes. During each episode e, $e = 1, 2, ..., e \leq 875$ of an epoch h, $h = 1, 2, ...$, we record all data necessary for the training and one per epoch update of neural networks.

B. Neural Network Training: The training is based on the actor-critic approach implemented by two neural networks. Both *actor* and *critic* networks have $|\mathbf{L}| + 1$ number of input neurons and 2 hidden layers, each with 256 neurons. Inputs of each hidden layer have parametric rectified linear units (PReLUs). The *actor* network has 3 output neurons for the log-probability σ_a of the three possible actions $a \in \{-1, 0, 1\}$ and the *critic* network has 1 output neuron for $\hat{S}$, which is the estimate of the state value S.

During the training, after each episode, we compute the sequence of *advantages*

$$A(k) = G_{k:k+n} - \hat{S}(k), \tag{4.20}$$

where $\hat{S}(k) = \hat{S}(q_{k-1}, \mathbf{L}(k))$ and $G_{k:k+n}$ is the predictor of the state value S which is computed over the time horizon n using the sequence of recorded rewards $R(k)$, therefore,

$$G_{k:k+n} = \begin{cases} \gamma^n \hat{S}(k+n), & k+n < K_T \\ \gamma^{K_T - k} R(K_T), & k+n \geq K_T, \end{cases} \tag{4.21}$$

where $R(K_T)$ is the terminal reward at the end of the episode and $\hat{S}(k+n)$ is the *critic* network estimation of the state value after the horizon of n steps. In our computations, we use the horizon of $n = 20$ time steps. Finally, after each episode, we compute the

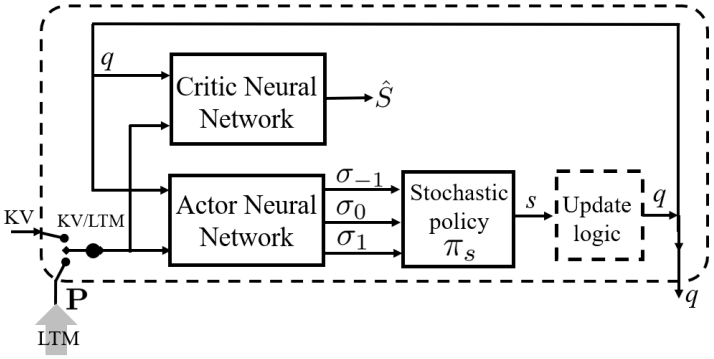


Figure 4.3: Actor-critic neural networks that replace the NN block in Fig. 1 during the training. The KV/LTM switch defines if the learning is based on the kinematic variables (KV) or the vector of metrics P (LTM).

sequence of $\sigma_a(k)A(k)$ values where

$$\sigma_a(k) = \log\left(\frac{e^{\lambda_s^h(a,q(k),\mathbf{L}(k)))}}{\sum_s e^{\lambda_s^h(s,q(k),\mathbf{L}(k))}}\right) \tag{4.22}$$

in which $\lambda_s^h(s, q(k), \mathbf{L}(k))$ is the so-called *logit* and is in a one-to-one relation with the stochastic policy of the epoch h, $h = 1, 2, ...$ as $\pi_s^h(a, q(k), \mathbf{L}(k)) = \frac{e^{\lambda_s^h(a,q(k),\mathbf{L}(k)))}}{\sum_s e^{\lambda_s^h(s,q(k),\mathbf{L}(k))}}$. From the sequence of $\sigma_a(k)A(k)$ values from all episodes, we compute the policy loss estimation after each epoch h as

$$\hat{\Pi}_h^{loss}(\phi_a) = -\frac{1}{\sum_e K_T^e} \sum_{k=1}^{\sum_e K_T^e} \sigma_a(k)A(k), \tag{4.23}$$

where K_T^e denotes the length K_T of the episode e and ϕ_a denotes that the policy loss is dependent on the *actor* network parameters.

The actor neural network parameters ϕ_a are updated to minimize the policy losses given by (4.23). The critic neural network parameters are updated to minimize

the mean square error of state value estimations, i.e.,

$$\mathcal{L}(\phi_c) = \sum \Big(G_{k:k+n} - \hat{S}(q_{k-1}, \mathbf{L}(k)) \Big)^2 . \tag{4.24}$$

All presented computations of the training are implemented in PyTorch. After we have obtained the convergence of training results, we substitute the learned actor network in the NN block in Fig. 4.1. In this case, the stochastic policy is replaced with the action with the highest probability, i.e.,

$$s_k = \arg\max_a \sigma_a(q_{k-1}, \mathbf{L}(k)) \tag{4.25}$$

and this action is used to navigate the bicycle at the time step k.

4.5 Results

The numerical results of this section are based on the bicycle kinematic model (4.5)-(4.6). In our results, we used the parameters (4.14), the sequence of velocity (4.15) and the initial state defined by (4.18)-(4.19). We computed stochastic optimal controllers u_i and tables P_i for each forward velocity v_i, $i > 0$ and for the backward velocity v_0, we navigated by setting $u_0 = u_5$. The computed lookup tables have the size of $72 \times 72 \times 100 \times 21$ which is associated with the discretization of α, β, r and ϕ variables, respectively.

We compared the KV and LTM training approaches based on their learning performances. For that, we ran each training 20 times for 5000 epochs, then computed averages and standard deviations for each epoch across the 20 runs. The computed

averages are plotted in Figs. 4.4-4.6. Figure 4.4 shows that both KV and LTM training average rewards converge to the maximal possible reward $M = 100$, but the convergence of the LTM is faster and its standard deviation is lower than the one from the KV training after about 200 epochs. The second evidence of the LTM based learning efficiency is given in Fig. 4.5 showing the average loss which measures the error of state value estimation $\mathcal{L}(\phi_c)$. For the LTM training, the average critic loss drops close to zero after less than 1000 epochs and has less variations than the plot for the KV based method. The final evidence illustrating the LTM based training efficiency are plots in Fig. 4.6. Both the KV and LTM plots start at $\ln 3 \approx 1.1$, which is the maximal possible value of mean entropy and corresponds to the 3 equally probable actions $a = \{-1, 0, 1\}$. The plots show clearly that in comparison to the KV training, the LTM training converges closer to zero with slightly less variations over the 5000 epochs, therefore, the convergence is to a better defined policy with a narrower distribution of actions.

To illustrate the work of the LTM trained neural network, Fig. 4.7(top) depicts the bicycle navigating gates (a) to (h) in the alphabetical order and coming back to gate (a). The bicycle starts at the initial pose $(0, 0, 90\pi/180)$, which is aligned with the pose of gate (a). Since we depict the pose at gate (a) at the end of the bicycle trajectory, the initial pose is not depicted in the figure. The trajectory shows that anytime that reaching the next gate requires a sharp turn, the bicycle uses backward motion to reposition itself with respect to the next gate. The backward motion can be also observed in the time plot of the bicycle velocity in Fig. 4.7(bottom).

In Fig. 4.8, we have the bicycle navigating a road described by a sequence of

gates. We depicted only 6 out of 17 gates representing the road and separated by the distance that is $\overline{p}/12 \approx 0.44m$, which is a fraction of the circle perimeter with the radius $\overline{p}$ from (4.19). The bicycle trajectory in Fig. 4.8(top) shows that the road is problematic to navigate since the rate of change of the road curvature, i.e., gate orientation, is fast and the bicycle cannot keep up with it due to the sample time $\Delta t = 0.1s$ used in the control update. The sample time is long and does not allow the bicycle enough update steps to steer through the gates, i.e., the bicycle misses gates. Because of that, the neural network dictates the bicycle to reverse the motion direction to pass the gates. Once we have decreased the sample time to $\Delta t = 0.01s$, we see that the bicycle navigation strategy can follow the road without reversing the motion direction. In summary, when the rate of change of the road is fast, the control strategy infers correctly that it should reverse the moving direction of the bicycle.

After an inspection of values P_i in the proximity of a gate, we have found that P_1 has the smallest value. However, the policy did not switch to the velocity v_1 in the proximity of gate. Therefore, as expected we conclude that the neural network implemented policy is not greedy, i.e., it does not switch always to the velocity with the shortest expected paths.

4.6 Conclusions

The paper describes the approach to the actor-critic neural network training for switching among an array of CG stochastic optimal controllers. The novelty of the

method is that instead of observable variables of the problem, the training inputs to the neural network are the off-line computed metrics that are stored in lookup tables for each controller, hence, we denominate it as the lookup table metric (LTM) training.

The LTM training takes advantage of information about the performance of each controller. Associated with it is the feature that the number of inputs to the neural network increases with the number of controllers. This may be of particular importance for practical applications requiring a large number of controller. Moreover, our results show that under the same conditions, the LTM training approach converges faster and with less variations than the more traditional KV approach. These results show the benefit of hierarchical control architecture design by the synergistic use of control theory and artificial intelligence computational methods, which gives us an interesting future research direction as a follow-up to this work.

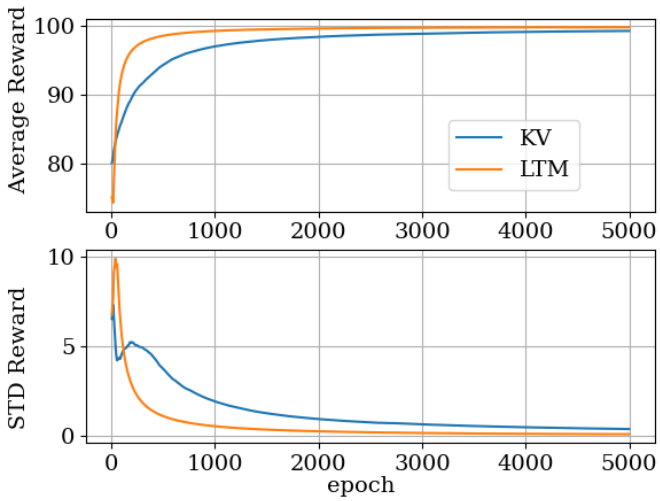


Figure 4.4: Average rewards and their standard deviations (STD) for the kinematic variable (KV) and lookup table metric (LTM) based trainings.

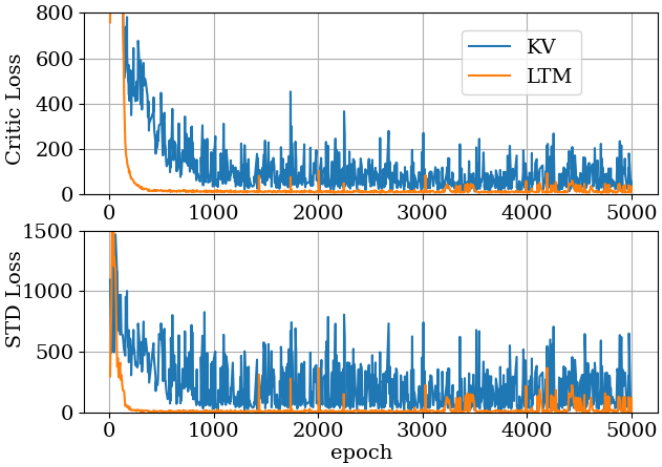


Figure 4.5: Average critic losses and their standard deviations (STD) quantifying the error of value function approximation. The plots are for the kinematic variable (KV) and lookup table metric (LTM) based trainings.

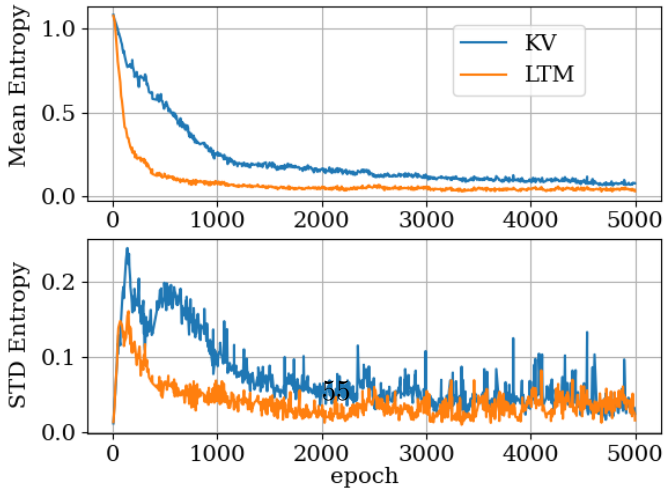


Figure 4.6: Mean action entropies and their standard deviations (STD). The plots are for the kinematic variable (KV) and lookup table metric (LTM) based trainings.

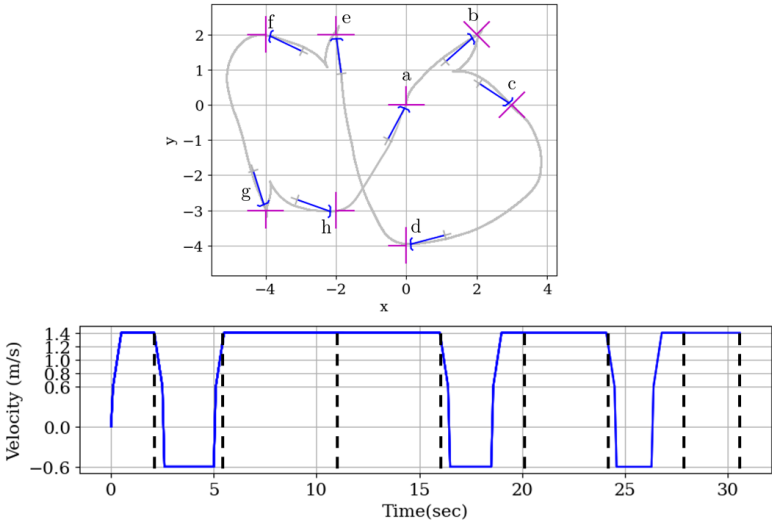


Figure 4.7: A bicycle navigating multiple gates: (top) the trajectory of the bicycle that goes through the gates in the alphabetical order from (a) to (h) and back to (a); (bottom) the velocity profile along the trajectory; the time instants of going through the gate are indicated by the dashed lines.

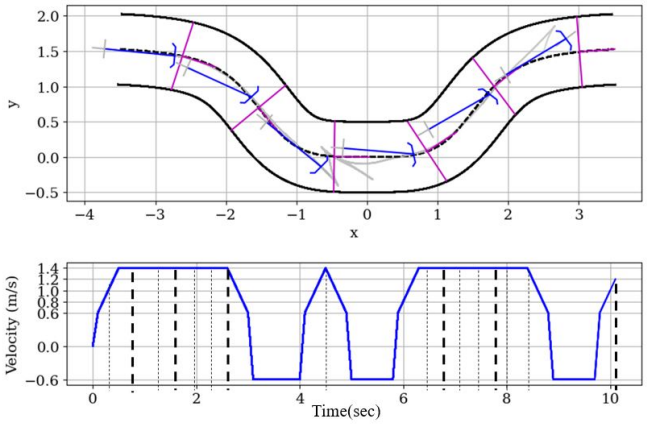


Figure 4.8: The bicycle navigates a road, $\Delta t = 0.1s$: (top) the bicycle trajectory with depicted 6 out of 17 gates separated by 0.44m; (bottom) the velocity profile with the dashed line indicating time points of passing through each gate.

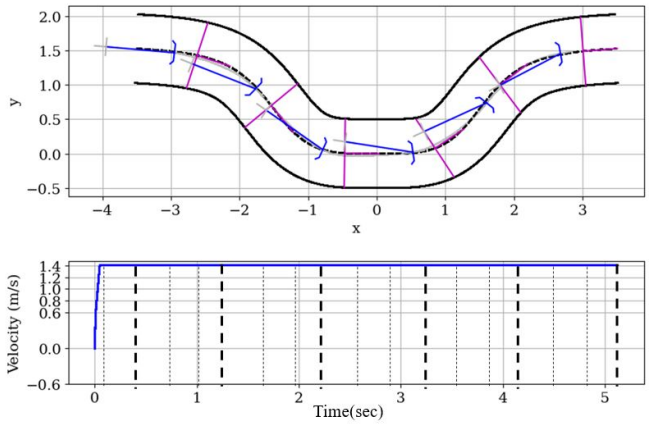


Figure 4.9: The bicycle navigates a road, $\Delta t = 0.01s$: (top) the bicycle trajectory with depicted 6 out of 17 gates separated by 0.44m; (bottom) the velocity profile with the dashed line indicating time points of passing through each gate.

Chapter 5

Metrics-only Training of Neural Networks for Switching among an Array of Feedback Controllers

- M. A. Carmona, A. Faust, and D. Milutinović, "Metrics-only Training of Neural Networks for Switching among an Array of Feedback Controllers," was submitted to 2022 Robotic Automation Letters

5.1 Introduction

We investigate in this paper a training method for neural networks to switch among an array of feedback controllers. Each feedback controller (FC) is verified for the operation mode, which means that there is a metric that measures the distance from the current state to a desired state (or states) under the action of the controller. In nonlinear control, these measures can be associated to the so-called Lyapunov functions

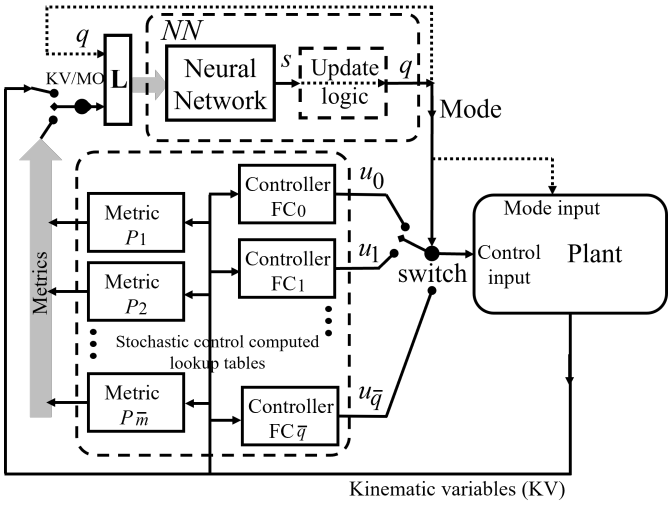


Figure 5.1: Neural network that implements a policy for switching among an array of feedback controllers (FCs). Each of these controllers is designed to control the plant in one of the operation modes $q = 0, 1, 2, ...\bar{q}$. Each FC_q, $q = 0, 1, ...\bar{q}$, takes the kinematic variables (KV) and outputs the control u_q while the neural network outputs the variable s, which updates q. The update logic block in the figure facilitates that modes q can be only updated in a certain order. The block is depicted with the dashed line since it may not exist for some control problems. Similarly, the dotted line towards the neural network input vector $\mathbf{L}$ and the mode input of the plant depicts that q can be used in the training and to set parameters of the plant, when applicable. The switch KV/MO at the input of the block $\mathbf{L}$ illustrates that the neural network can be trained based on KV, or based on the set of available P_m, $m = 1, 2...\overline{m}$ metrics, which we denominate as the metrics-only (MO) training, $\overline{m} + 1 \leq \overline{q}$.

[30], while in optimal control [6], stochastic control [34] and reinforcement learning [54], the measures can be associated to the value functions.

A. Motivation: The hierarchical control architecture in Fig. 5.1 is *inspired* by autonomous vehicles, or mobile robot systems. The figure shows a plant which outputs kinematic variables (KV) and illustrates a typical scenario in which we deal with a number of controllers, each carefully designed using control theory methods and under certain assumptions, i.e., for certain modes q. However, there is a challenge in how to switch among these controllers, i.e., how to update control modes q and achieve a certain control performance. Since the rule for updating the modes q can be complex, we propose to train a neural network to learn the update rule.

The kinematic variable (KV) training is a standard approach to training a neural network to perform an update rule for the mode q and scenario depicted in Fig. 5.1. In the figure, the KV training feeds the neural network with the variable q and KV. *The problem with the KV training approach is ($P1$) that it does not account explicitly for the performance of FCs, i.e., their sophisticated design or training. Another problem is ($P2$) that independently of the number of FCs with which we are dealing, the KV training is always based on the same number of variables.* Due to ($P2$), the KV training does not take an explicit advantage of increasing the number of modes, which is related to the refinement of FCs, i.e., increase of their number and their known performances. This is a disadvantage for trainings with a large number of FCs, which is required for practical applications. Overall, due to ($P1$) and ($P2$), the KV training approach lacks an explicit account for an effort in a careful selection of modes and the

design, or training of the corresponding FCs. Motivated by the disadvantages of the KV training, we propose the metrics-only (MO) training and compare the two approaches using the hierarchical structure in Fig. 5.1.

In the MO training, we feed the neural network with the vector $\mathbf{L}$ of variables, including P_m metrics, which addresses $(P1)$. As depicted in Fig. 5.1, the MO training feeds the neural network with $\overline{m}$ variables, therefore, the number of neural network input variables depends on the number of used controllers, i.e., the number of their available performances, which addresses $(P2)$. Overall, the MO training explicitly accounts for an effort in a careful mode selection and the design, or training of individual FCs.

An important feature of the MO training is that it does not use KV values. They belong to the physical layer of the system and are considered by the design, or training of individual FCs. The updates of modes q generated by the neural network depend on higher-level information about controller performances P_1, P_2,...$P_{\bar{m}}$. If necessary, the updates can also be based on the current mode q and that variable can also be used to set a mode for the plant, which is represented by the dotted line in Fig. 5.1. Without the need to access KV values for the updates, the trained neural network complies fully with the hierarchical structure of the control architecture.

B. Related work: The control architecture in Fig. 5.1 is a version of hierarchical control architectures that have been studied both in control and robotics [52]. Due to the continuous dynamics of the system under control and the switching among multiple controllers, this system is also an example of *hybrid* or *supervisory* control systems [33]. While the switching may result in instability [38, 12, 36], it can also contribute to

performance improvements [46, 37]. A performance signal-based switching for mobile robots has been considered in [26] and the optimality of switching has been considered in [7]. To switch between the navigation to goal and obstacle avoidance in [58], the authors used Lyapunov functions, while in [43], the authors considered a stochastic problem and used a rule for switching based on a probability inequality. This use of probabilistic inference to implement switching inspired our work on learning a safe switching policy. The work presented here is aligned with hierarchical approaches to mobile robot navigation which use learning [18, 22, 59] or stochastic control [45, 44], as well as the emergence of data-driven approaches for learning rules for switching among multiple controllers [1, 27] or options [55, 56, 8].

The options in our work are FCs that can be defined analytically based on control theory or can be computed, e.g., using the numerical stochastic optimal control [34]. Once we have obtained the FCs, we use the reinforcement learning and the actor-critic [53, 8] neural network to learn the rule for switching among the controllers. The neural network training is performed under the assumption that the rule is applied at every time step; in other words, we deal with *single-step options* [55]. A way to formulate the training is to use data associated with the original problem formulation as in [56]. In that case, the KV/MO switch in Fig. 5.1 would be in the position KV, indicating that the learning is based on the KV.

C. Contribution: The novelty of our work is that the training of the neural network accounts for the controller performances by using the MO values. This performance aware training corresponds to the MO position of the MO/KV switch in Fig. 5.1.

It is expected that the MO training is more efficient than the KV based one since the latter does not take into account any performance information about the controllers. We compare convergences of the KV and MO data based trainings under identical conditions for a pendulum control problem and a tricycle navigation one. For both problems, our results confirm that the MO based training is faster and less noisy than the KV based training. Furthermore, for the tricycle navigation problem, we provide results showing that the trained neural network can be used beyond numerically simulated control problems.

D. Paper layout: In Section 5.2, we formulate the maximization of a reward function returning a stochastic policy for the KV and MO training approaches. This is followed by the description of the actor-critic neural network training in Section 5.3 and KV/MO training comparison for the pendulum control problem from [56] in Section 5.4. In Section 5.5, we compare the KV/MO for an array of stochastic controllers and the tricycle navigation problem that motivated this work. We present our conclusions in Section 5.6.

5.2 Problem Formulation

To learn the policy, we propose to use an actor-critic neural network. The network learns the stochastic policy $\pi_s : \mathcal{S} \times \mathbf{R}^{|\mathbf{L}|} \to \mathbf{R}$, which returns probabilities for possible actions $s \in \mathcal{S}$ for the current vector of variables $\mathbf{L}$ used in the training. The symbol $|\mathbf{L}|$ denotes the size of the vector $\mathbf{L}$. The policy at the time step k can be also

denoted as $\pi_s(s, \mathbf{L}(k))$.

In the case of the KV based learning, the vector $\mathbf{L}$ has components that are normalized values of KVs. The normalization of each variable is based on its upper and lower bounds and, in general, these values can be positive or negative. The MO based learning uses the vector $\mathbf{L}$ composed of normalized non-negative metrics P_i, $i = 1, 2, ...$ The normalization of each metric P_i is based on its upper bound and the normalized values are always positive. In both KV and MO training methods, $\mathbf{L}(k)$ at time step k can also include the normalized value of the current discrete state q with the understanding that at the step k, the current value of q is the one selected in the previous time step, i.e., q_{k-1}.

The stochastic policy π_s that we learn is based on the maximization of the expected reward J as

$$\pi_s^* = \arg\max_{\pi_s} E\{J\} \tag{5.1}$$

with the reward J defined as

$$J = g_T(\mathbf{x}_{K_T}) + \sum_{k=1}^{K_T} \gamma^{k-1} r_k, \tag{5.2}$$

where r_k is the reward following the decision s_{k-1} in the discrete step $k-1$, K_T is the number of discrete steps after which the state $\mathbf{x}_{K_T}$ enters the terminal set $\mathcal{T} = \mathcal{G} \cup \mathcal{L}$ and $g_T(\mathbf{x}_{K_T})$ is the terminal reward. The terminal set $\mathcal{T}$ is composed of two disjoint sets ($\mathcal{G} \cap \mathcal{L} = \emptyset$), where the set $\mathcal{G}$ denotes a desired target set or a goal set for the optimal policy π_s^*. The set $\mathcal{L}$ denotes a set of states that should be avoided by the policy π_s^*. These two sets are associated with different values of the terminal reward

function $g_T(\mathbf{x})$. Since not all of the problems we consider have the avoidance set, we can state that $\mathcal{G} \subseteq \mathcal{T}$ and $\mathcal{G} = \mathcal{T} \Leftrightarrow \mathcal{L} = \emptyset$.

Once we have found the optimal stochastic policy π_s^*, we define the update policy for the step k as

$$s_k = \arg\max_s \pi_s^*(s, \mathbf{L}(k)), \tag{5.3}$$

where $s \in \mathcal{S}$ denotes available actions and $\mathbf{L}(k)$ is composed either of the normalized KVs, or the normalized metric values for the policy learned from the KV or the MO training, respectively. In both cases, the vector $\mathbf{L}$ can also include the normalized value of the discrete state q_{k-1} at the time step k before the action s_k is taken.

5.3 Actor-Critic Neural Network

The neural network training is based on the numerical simulations of the system under control. In general, each episode e, $e = 1, 2, ...$ of the training starts with a random selection of the initial state. From the initial position, we perform the simulation with the time step T_s. Each episode lasts until the system reaches the terminal set, otherwise the episode stops if it is too long. A sequence of multiple episode runs is an epoch h, $h = 1, 2, ...$ During each episode e of an epoch h, we store the following data

$$D_{eh}(k) = (\sigma_a(k), \Sigma_\pi(k), R(k), \hat{S}(k)), \tag{5.4}$$

where $\sigma_a(k)$ is the log-probability of the selected action $a(k) \in \mathcal{S}$ from the step k computed as

$$\sigma_a(k) = \log\left(\frac{e^{\lambda_s^h(a, \mathbf{L}(k)))}}{\sum_s e^{\lambda_s^h(s, \mathbf{L}(k))}}\right). \tag{5.5}$$

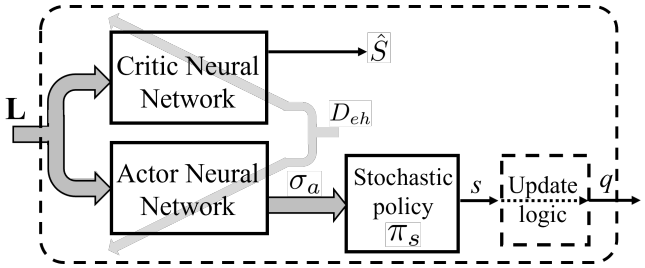


Figure 5.2: Actor-critic neural networks that replace the *NN* block in Fig. 1 during the training: the vector D_{eh} depicts the data collected during each episode e of the training epoch h, see expression (5.4). The light gray arrows across the network indicate that the data D_{eh} are used for network updates. The output of the critic network is the estimation of the value function and the output of the actor one are probabilities σ_a of actions $a \in \mathcal{S}$ that result in the switching of the controllers.

Σ_π is the vector of log-probabilities of possible actions at the step k and $\lambda_s^h(s, \mathbf{L}(k))$ is the so-called *logit*, which is in a one-to-one relation with the stochastic policy of the epoch h, $h = 1, 2, ...$ as

$$\pi_s^h(a, q(k), \mathbf{L}(k)) = \frac{e^{\lambda_s^h(a, \mathbf{L}(k)))}}{\sum_s e^{\lambda_s^h(s, \mathbf{L}(k))}}. \tag{5.6}$$

The third variable $R(k)$ in $D_{eh}(k)$ that we store is the reward r_k at each time step, including the terminal reward $g(x_{K_T})$. The last variable that we store in $D_{eh}(k)$ is $\hat{S}(k)$, which is the estimated value $S(k)$ of the state $\mathbf{L}(k)$, i.e.,

$$\hat{S}(k) = \hat{S}(\mathbf{L}(k)). \tag{5.7}$$

This variable is the output of the *critic* neural network that we use in our neural network

training. In addition to the data D_{eh} that are stored for each time step k, at the end of each episode we also store the total reward J, see expression (5.2)

The training is based on the actor-critic approach implemented by two neural networks. The learning in both neural networks is based on the state $\mathbf{L}(k)$. The *critic network* takes the state as the input and outputs $\hat{S}$, which is the estimate of the state values S. Therefore, after each episode, we can compute the sequence of *advantages*

$$A(k) = G_{k:k+n} - \hat{S}(k), \tag{5.8}$$

where $\hat{S}(k) = \hat{S}(\mathbf{L}(k))$ and $G_{k:k+n}$ is the predictor of the state value S, which is computed over the time horizon n using the sequence of recorded rewards r_k. Therefore, for $k + n < K_T$,

$$G_{k:k+n} = \gamma^n \hat{S}(k+n) + \sum_{i=1}^{n} \gamma^{i-1} r_{k+i} \tag{5.9}$$

and for $k + n \geq K_T$,

$$G_{k:k+n} = \gamma^{K_T - n} g(\mathbf{x}_{K_T}) + \sum_{i=1}^{K_T - n} \gamma^{i-1} r_{k+i}, \tag{5.10}$$

where $g(\mathbf{x}_{K_T})$ is the terminal reward at the end of the episode and $\hat{S}(k+n)$ is the *critic* network estimation of the state value after the horizon of n steps. Finally, after each episode, we compute the sequence $\sigma_a(k)A(k)$, then we concatenate these sequences and after each epoch h, compute the policy loss as

$$\hat{\Pi}_h^{loss} = -\frac{1}{\sum_e K_T^e} \sum_{k=1}^{\sum_e K_T^e} \sigma_a(k) A(k), \tag{5.11}$$

where K_T^e denotes the length K_T of the episode e and the expression is a numerical

estimation of the policy loss

$$\Pi^{loss} = -E_\tau \left\{ \sum_{k=1}^{K_T} \sigma_a(\mathbf{L}(k)) A(\mathbf{L}(k)) \right\} \tag{5.12}$$

with E_τ denoting the expectation operator with respect to the episode trajectories.

In the above listed computations, $\hat{S}(k)$ are outputs of the critic neural network and σ_a are outputs of the actor neural network. Therefore, during the training, the dashed *NN* block in Fig. 5.1 is replaced by the one presented in Fig. 5.2, which is composed of the actor and critic neural networks.

The actor neural network parameters ϕ_a are updated to minimize the policy losses given by (5.11). The critic neural network parameters are updated to minimize the mean square error of state value estimations, i.e.,

$$\mathcal{M}(\phi_c) = \sum \left(G_{k:k+n} - \hat{S}(\mathbf{L}(k)) \right)^2, \tag{5.13}$$

where $\hat{S}$ depends on the critic network parameters ϕ_c.

All presented computations of the training are implemented in PyTorch. After we have obtained the convergence of training results, we substitute the learned actor network in the *NN* block in Fig. 5.1. In this case, the stochastic policy is replaced with the action with the highest probability, i.e.,

$$s_k = \arg\max_a \sigma_a(\mathbf{L}(k)). \tag{5.14}$$

5.4 Pendulum Control

Figure 5.3 depicts a pendulum with the dynamics

$$\dot{\theta} = \omega, \tag{5.15}$$

$$\dot{\omega} = g \sin\theta + u, \tag{5.16}$$

where $\theta \in (-\pi, \pi]$ is the pendulum angular position and $\omega \in \mathbb{R}$ is the pendulum angular velocity $\omega = \dot{\theta}$. The model and all parameters are from [56], including that the pendulum control variable is a bounded torque $u \in [-u_{max}, u_{max}]$, $u_{max} = 0.224$. In [56], the mechanical energy (ME) of the pendulum for the state $\mathbf{x} = (\theta, \omega)$ is defined as

$$ME(\mathbf{x}) = 1 + \cos\theta + \frac{1}{2}\omega^2. \tag{5.17}$$

Therefore, in the initial state $\mathbf{x}_0 = (\pi, 0)$ in which the pendulum does not move, the mechanical energy $ME(\mathbf{x}_0) = 0$. As described in [56], the goal of pendulum control is to learn a feedback control policy for u such that the pendulum reaches the target state $\mathbf{x}_T = (0, 0)$. In that target state, the pendulum is vertically up, it stays motionless in

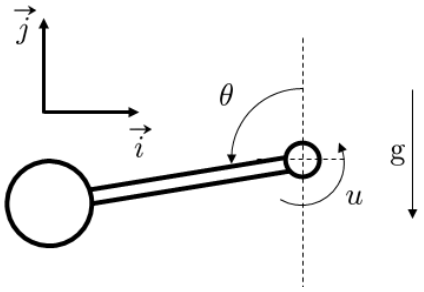


Figure 5.3: Pendulum mechanism: angular position θ, torque u, the direction of gravitation g and orthogonal unit vectors $\vec{i}$ and $\vec{j}$.

that position and $ME(\mathbf{x}_T) = 2$.

Let us introduce the state of the pendulum as $\mathbf{x} = (\theta, \omega)$. In [56], the authors used the mechanical energy to introduce a Lyapunov function $L(\mathbf{x})$ as

$$L(\mathbf{x}) = 2 - ME(\mathbf{x}). \tag{5.18}$$

In the initial state $L(\mathbf{x}_0) = 2$ and in the target state $L(\mathbf{x}_T) = 0$. Using the analysis of Lyapunov function L and ME, the authors in [56] formulated four control actions:

$$u_0 = u_{max}, \qquad u_1 = -u_{max}, \tag{5.19}$$

$$u_2 = EA(u_{max}, \theta, \omega), \quad u_3 = EA(\frac{1}{2}u_{max}, \theta, \omega), \tag{5.20}$$

where

$$EA(u, \theta, \omega) = \begin{cases} \text{sgn}(\theta)u & \text{if } \omega = 0 \\ \frac{1}{2}\text{sgn}(\omega)u & \text{if } (0 < \omega \leq \delta_\omega \text{ and} \\ & (\theta \in [\theta_3 - \delta_\theta, \theta_3) \text{ or} \\ & \theta \in [\theta_4 - \delta_\theta, \theta_4))) \\ & \text{or } (-\delta_\omega < \omega < 0 \text{ and} \\ & (\theta \in (\theta_1, \theta_1 + \delta_\theta] \text{ or} \\ & \theta \in (\theta_2, \theta_2 + \delta_\theta])) \\ \text{sgn}(\omega)u & \textit{otherwise} \end{cases} \tag{5.21}$$

with $\text{sgn}(x) = 1$ if $x \geq 0$ and -1 otherwise. The values $\delta_\theta = 0.1$, $\delta_\omega = 0.1$ and $\theta_1, \cdots, \theta_4$ are the equilibrium points with $\dot{\omega} = 0$, thus, $\theta_1 = \sin^{-1}(u)$, $\theta_2 = \pi - \theta_1$, $\theta_3 = \theta_1 - \pi$, $\theta_4 = -\theta_1$. The reasoning behind these controllers is provided in Appendix A.1.

In [56], the authors used reinforcement learning to learn the policy for switching

among the controllers u_0, u_1, u_2 and u_3 with the goal to reach the terminal state $\mathbf{x}_T$ from the initial state $\mathbf{x}_0$. Since $ME = 2$ allows for reaching $\mathbf{x}_T$, once the pendulum reaches $ME \geq 2$, there is a mandatory switch to the u_4 controller (Section 4.2 in [56]) that "shaves out" the energy excess above $ME = 2$ and keeps the energy at that level until the terminal state $\mathbf{x}_T$ is reached. To accommodate for numerical imprecision, the reaching of the terminal state $\mathbf{x}_T$ is detected by the criterion $\sqrt{\theta^2 + \omega^2} < 0.1$.

In the work presented here, we use the controllers u_0, u_1,u_2 and u_3 from (5.19)-(5.20) and an improved energy "shave out" controller u_4

$$u_4 = \begin{cases} u_{max}, & -K\omega(ME(\mathbf{x}) - 2) \geq u_{max} \\ -K\omega(ME(\mathbf{x}) - 2), & otherwise \\ -u_{max}, & -K\omega(ME(\mathbf{x}) - 2) \leq -u_{max} \end{cases} \tag{5.22}$$

with the gain $K = 6.0$. The improved controller u_4 alleviates the irregular motion due to a large torque gain in the controller and guarantees that the control torque is bounded to the magnitude of u_{max}. The bound of the torque is such that a single controller cannot be used to reach $M(\mathbf{x}) = 2$ ($L(\mathbf{x}) = 0$) and ultimately the state $\mathbf{x}_T$. For that, we need to learn a switching policy for the four controllers (5.19)-(5.20) which will swing the pendulum back and forth until the final state $\mathbf{x}_T$ is reached or $ME(\mathbf{x}) \geq 2$. In the latter case, we apply the control u_4 until the final state $\mathbf{x}_T$ is reached.

5.4.1 KV vs. MO Training of the Actor-Critic Neural Network

For both the KV and MO training methods, we use episodes that start at the initial state $\mathbf{x}_0$. The episode time step is $T_s = 0.1$. The reward J that is maximized is

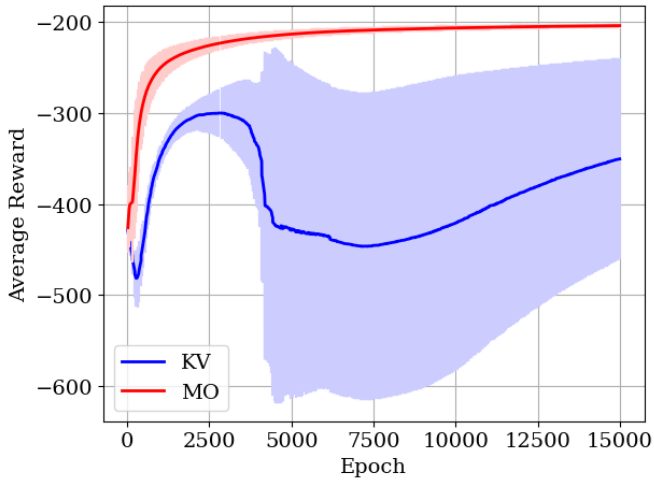


Figure 5.4: The average reward for the kinematic variable (KV) training and metrics-only (MO) training. The shaded region depicts one-standard deviation bands.

given by (5.2) in which $\gamma = 1$, $r_k = -1$ and $g_T(\mathbf{x}_{K_T}) = 0$. The target set of the criterion $\mathcal{G} = \{(\theta, \omega) | \sqrt{\theta^2 + \omega^2} < 0.1\}$ and $\mathcal{L}$ is an empty set. The maximization of this reward minimizes the total time it takes from the initial state $\mathbf{x}_0$ to the final state $\mathbf{x}_T$ in which the pendulum is vertically up.

We use θ and ω as kinematic variables (KV) for the training of the actor-critic neural network to switch among four controllers (5.19)-(5.20), i.e., $\overline{q} = 3$ in Fig. 5.1. The metrics-only (MO) training is based on the single metric $L(\mathbf{x})$ from (5.18), i.e., $\overline{m} = 1$ in Fig. 5.1.

Figures 5.4 and 5.5 show plots of the KV and MO training results. The plots are averaged over 5 independent training runs for each type of the training. The averages are depicted by the solid lines and one-standard deviation bands are depicted by the shaded regions.

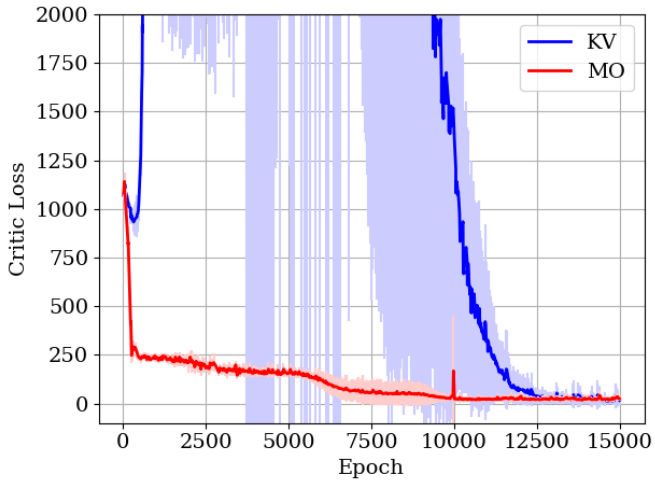


Figure 5.5: The training average critic loss and its standard deviation for swinging a pendulum to the upright position with the initial pose $(\pi, 0)$. KV denotes the kinematic variable and MO the metrics-only training results.

The plot in Fig. 5.4 shows that the MO training is faster and it results in a higher performance after 15k epochs of the training. The KV training improves its updates after 7.5k epochs, which coincides with the positive slope in Fig. 5.4 and the decline in Fig. 5.5. Yet, even in the latter figure, the MO training shows a faster convergence with less variability. Therefore, the MO training is not only faster, but also more reliable.

5.4.2 Validation of the Trained Switching Policy

After 30k epochs of the KV training and 15k epochs of the MO training, we obtained the switching policies that can be used to control the pendulum from the initial state $\mathbf{x}_0$ to the final state $\mathbf{x}_T$. As a benchmark, we also computed an optimal feedback

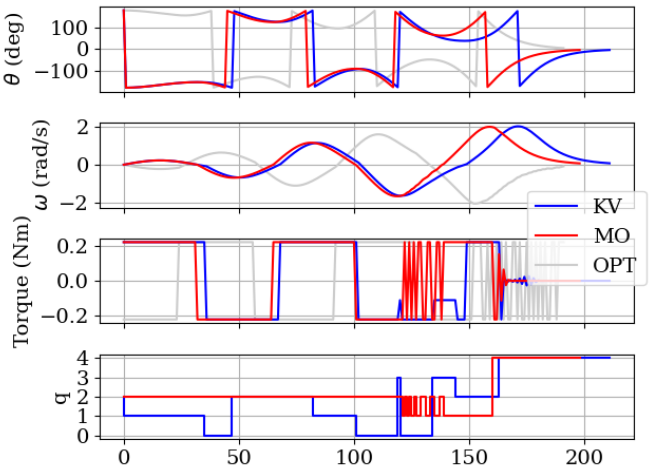


Figure 5.6: Plots of the angle θ, angular velocity ω, control torque and the index q of the currently active controller. The index $q = 0, 1, 2, 3$ corresponds to the controllers (5.19)-(5.20). The index $q = 4$ is active after $ME \geq 2$.

control for the torque $u = u(\theta, \omega)$ that takes the pendulum from the initial to the final state in the minimum time, see Appendix A.2.

Figure 5.6 shows the results for the KV and MO trained switching policies, as well as for the numerically computed optimal controller. From these, we can see that the KV trained policy takes more than 200 steps to reach the final state. The time for the MO trained policy is shorter (199 steps) and comparable to the time (192 steps) resulting from the optimal controller (OPT).

5.5 Tricycle Navigation

We consider below a problem of tricycle navigation through a gate as depicted in Fig. 5.7. The tricycle is a *front-wheel* drive (FWD) tricycle with the kinematics

$$dx_A = v_i \cos(\phi + \theta)dt, \tag{5.23}$$

$$dy_A = v_i \sin(\phi + \theta)dt, \tag{5.24}$$

$$d\theta = \frac{v_i}{L} \sin \phi dt, \tag{5.25}$$

$$d\phi = u_i \ dt, \tag{5.26}$$

where (x_A, y_A) denotes position coordinates of the steering wheel pivot point A, $\theta \in (-\pi, \pi]$ is the heading angle, $\phi \in (-\frac{\pi}{2}, \frac{\pi}{2})$ is the steering angle, $L = 1.0m$ is the tricycle length and $u_i \in [-1, 1]$ is the bounded steering rate control variable. The kinematics describes the motion of the tricycle for a constant velocity v_i of the front wheel, i.e., of the pivot point A. The set of forward velocities is $v_i = 0.6 + 0.2(i - 1)$, $i = 1, ...5$ and the single backward velocity is $v_0 = -v_1 = -0.6$. All velocities are in units of m/s.

Forward velocity controllers u_1 to u_5: For each constant forward velocity v_i, $i = 1, ...5$, we solve numerically a minimum time optimal control u_i for steering the tricycle through a gate G positioned at (x_G, y_G), see Fig. 5.7. In our solution, we address the uncertainty of gate orientation as perceived by the gate approaching tricycle by describing the gate orientation angle as $d\theta_G = \sigma_G dw_G$, where dw_G is the unit intensity Wiener process increment and σ_G is a scaling factor. From this and the FWD tricycle kinematics (5.23)-(5.26), Itô's formula [48] yields the following stochastic kinematics for

the FWD tricycle-gate relative position

$$dr = b_r dt = -v_i \cos(\beta - \phi)\ dt, \tag{5.27}$$

$$d\beta = b_\beta dt = \frac{v_i}{r} \sin(\beta - \phi)\ dt - \frac{v_i}{L} \sin\phi\ dt, \tag{5.28}$$

$$d\alpha = b_\alpha dt + \sigma_G dw_G = -\frac{v_i}{L} \sin\phi\ dt + \sigma_G\ dw_G, \tag{5.29}$$

$$d\phi = b_\phi dt = u_i\ dt, \tag{5.30}$$

where $r = \sqrt{(x_G - x_A)^2 + (y_G - y_A)^2}$ is the tricycle-gate distance, $\beta = -\theta + \arctan(\frac{y_G - y_A}{x_G - x_A})$ is the bearing angle, and $\alpha = \theta_G - \theta$ is the misalignment angle between the tricycle heading and the gate orientation. By defining the state $\mathbf{x} = (r, \beta, \alpha, \phi)$, $\mathbf{x} \in \mathbf{R}^4$, we use the same approach as in [15] to solve numerically optimal feedback control policies $u_i = u_i(\mathbf{x})$. For that reason, we define a *target* set $\mathcal{G}_T$, which should be reached within the minimum time as

$$\mathcal{G}_T = \{\mathbf{x} |\ r \leq \underline{r},\ |\beta| \leq \underline{\beta},\ |\alpha| \leq \underline{\alpha},\ |\phi| \leq \underline{\phi}\}, \tag{5.31}$$

where $|\cdot|$ denotes absolute values and $\underline{r} < D$, where D is the gate width. We assume that when $\mathbf{x} \in \mathcal{G}_T$, the tricycle passes safely through the gate. The specific values that we use to define this set are $\underline{r} = 0.3m$, $D = 1m$, $\underline{\beta} = 80\frac{\pi}{180}$, $\underline{\alpha} = 50\frac{\pi}{180}$, and $\underline{\phi} = 50\frac{\pi}{180}$. To describe the configurations in which the tricycle misses the gate, i.e., the set of states $\mathbf{x}$ that should be avoided, we introduce set $\mathcal{A}_T$ as

$$\begin{aligned} \mathcal{A}_T = &\Big\{\mathbf{x} \mid -\pi < \phi \leq -\underline{\phi} \text{ or } \underline{\phi} \leq \phi \leq \pi\Big\} \\ &\cup \Big\{\mathbf{x} \mid r \leq \underline{r}, (-\pi < \beta < -\underline{\beta} \text{ or } \underline{\beta} < \beta \leq \pi), \\ &\qquad (-\pi < \alpha < -\underline{\alpha} \text{ or } \underline{\alpha} < \alpha \leq \pi)\Big\}, \end{aligned} \tag{5.32}$$

where the bounds $\underline{r}$, $\underline{\beta}$, $\underline{\alpha}$, $\underline{\phi}$ are the same as in (5.31). Given the kinematics (5.27)-(5.30), the sets $\mathcal{G}_T$ and $\mathcal{A}_T$, and the cost, which is the time to reach the target set $\mathcal{G}_T$, as in [15] we solve numerically the Hamilton-Jacobi-Bellman equation (HJB) for each velocity v_i, $i = 1, ...5$. With each solution, we obtain the optimal control $u_i = u_i(\mathbf{x})$, as well as the corresponding value function with the expected time $T_i = T_i(\mathbf{x})$. By multiplying the expected times with their corresponding velocities, we obtain expected paths $P_i = P_i(\mathbf{x}) = v_i T_i(\mathbf{x})$. All of these values are computed on a grid $72 \times 72 \times 100 \times 21$ associated with the discretization of $\alpha \in (-\pi, \pi]$, β, r, and $\phi \in (-\frac{\pi}{2}, \frac{\pi}{2})$ variables.

Backward velocity controller u_0: When the forward moving tricycle has to navigate towards a gate with a sharp turn, there may not be a fixed speed v_i feedback controller u_i, $i = 1, ...5$ that can navigate the tricycle to go on a trajectory with a loop before passing through the gate. The loop provides the re-positioning of the tricycle relative to the gate, but turning away from the gate can be undesirable, for example, when the sequence of the gates describes a road and the tricycle should stay on the road all the time. To allow for the re-positioning of the tricycle with a sequence of forward-backward motions, we use the velocity $v_0 = -0.6 < 0$. With the negative velocity, the gate approaching tricycle moves away from the gate. Under this condition, we cannot define the optimal control for reaching the target set $\mathcal{G}$. Therefore, we select that $u_0 = u_5$ and let the metrics for the controller u_0 be undefined.

Switching Policy Training: In this example, we train the actor-critic neural network to switch among the array of six controllers u_i and the FWD tricycle velocities v_i, $i = 0, 1, ...5$, ($\overline{q} = 5$ in Fig. 5.1). To prevent an abrupt change of the velocity, we

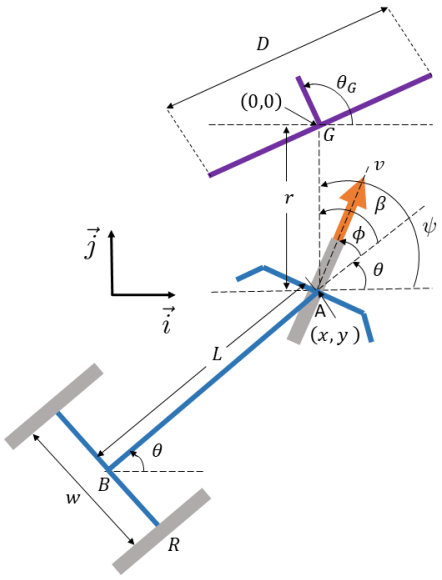


Figure 5.7: Tricycle-gate relative position: gate G is positioned at (x_G, y_G), θ_G is the heading angle of the gate and D is the width of the gate. The tricycle of length L and width w is depicted with the heading angle θ, steering angle ϕ, distance to the gate r, and bearing angle β. For the front-wheel drive (FWD) tricycle, the velocity vector (shown as the solid orange arrow) is associated with the pivot point A.

introduce a constraint that if $q(t)$ is the index of a currently active controller $u_{q(t)}$, then the index in the next time step $q(t + T_s)$ has to satisfy $|q(t + T_s) - q(t)| \leq 1$, where $T_s = 0.1s$ is the discrete time step of the controller execution.

For both the KV and MO training methods, the reward J that is maximized is given by (5.2) in which $\gamma = 0.99$, $r_k = 0$ and $g_T(\mathbf{x}_{K_T}) = 100$ for the tricycle that reaches the target set $\mathbf{x}_{K_T} \in \mathcal{G}$, which is the gate, i.e., $\mathcal{G} = \mathcal{G}_T$, see expression (5.31). The avoidance $\mathcal{L}$ set for the reward (5.2) maximization is defined as $\mathcal{L} = \{\mathbf{x} | \ r > \underline{r}, |\alpha| > \overline{\alpha}\}$

with $\overline{\alpha} = 135\frac{\pi}{180}$, which is the set of states $\mathbf{x}$ with a large misalignment angle at a large distance. This is an indicator that the tricycle goes along a trajectory with a loop. Consequently, if the tricycle starts going on a loop, i.e., $\mathbf{x}_{K_T} \in \mathcal{L}$, then the terminal cost is $g_T(\mathbf{x}_{K_T}) = -100$. The maximization of the reward J that is discussed here minimizes the number of time steps to the gate while penalizes the tricycle for going on a loop trajectory. Regardless of whether we train the actor-critic neural network using the KV or the MO training method, the training begins with no prior knowledge of the value function $\hat{S}$, see Fig. 5.2.

For the KV training, the input vector $\mathbf{L}$ to the actor-critic neural network is composed of normalized values of q and KV, $\mathbf{x} = (r, \beta, \alpha, \phi)$. For the MO training, the input $\mathbf{L}$ to the network is composed of normalized values of q and the metrics $(P_1, P_2, P_3, P_4, P_5)$, where P_i, $i = 1, ...5$, are expected paths corresponding to controllers u_i. Note that the metric for the controller u_0 is unavailable. Other details related to both KV and MO training methods are provided in Appendix A.3.

5.5.1 KV vs. MO Training of the Actor-Critic Neural Network

We compare here the KV and MO training approaches based on the average reward and critic loss performance measures. The performances were computed for each epoch of the trainings. We ran 20 trainings of 5000 epochs and computed average and standard deviation values across the runs that are plotted in Figs. 5.8 and 5.9.

Fig. 5.8 shows average rewards for the MO and KV based trainings. From the figure, we see that both KV and MO trainings converge to the maximal reward 100,

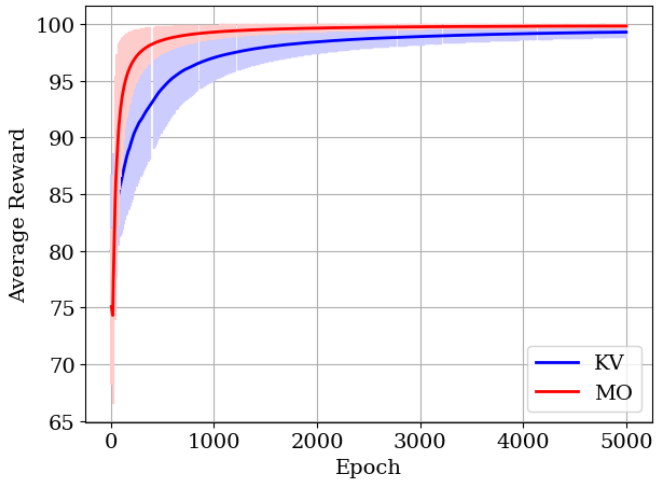


Figure 5.8: Average rewards and their standard deviations for the kinematic variables (KV) and metrics-only (MO) based trainings.

therefore, we conclude that in both trainings, the neural networks learn update rules, which yield that the tricycle converges to the gate. The figure also shows that the MO based training is faster in reaching the maximal value and that the standard deviation from the MO training is lower after about 200 epochs.

The second evidence of the MO based learning efficiency is given in Fig. 5.9 showing the average loss that measures the error of state value estimation $\mathcal{M}(\phi_c)$. For the MO training, the average critic loss drops close to zero after less than 1000 epochs and has less variations than the plot for the KV based method.

5.5.2 Validation of the FWD Tricycle MO Trained Switching Policy

We validate the trained FWD switching policy using a right back-wheel drive (RBWD) tricycle in the virtual robot experimentation platform CoppeliaSim (previously

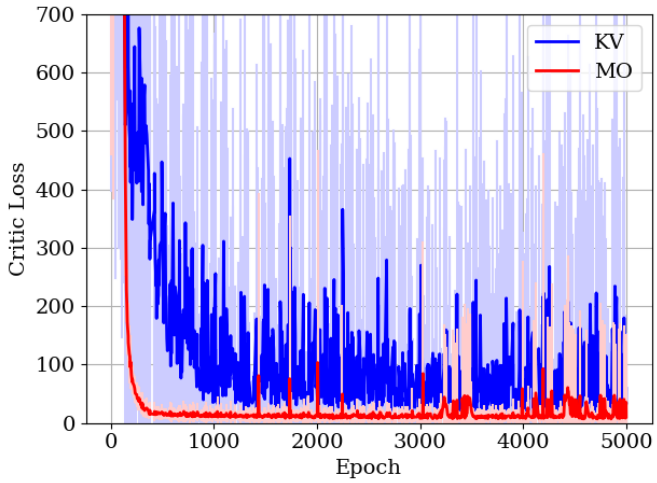


Figure 5.9: Average critic losses and their standard deviations quantifying the error of value function approximation. The plots are for the kinematic variable (KV) and metrics-only (MO) based trainings.

V-REP). We do that to illustrate that our results can be applied beyond the control of ordinary differential equation-simulated mathematical models. First, the wheels of the RBWD tricycle in CoppeliaSim can slip on the ground (see Appendix B.1). Second, the kinematics of the RBWD tricycle is different from the FWD tricycle kinematics (5.23)-(5.26), which was used in the policy training.

The mathematical model for the motion of the pivot point A for the RBWD

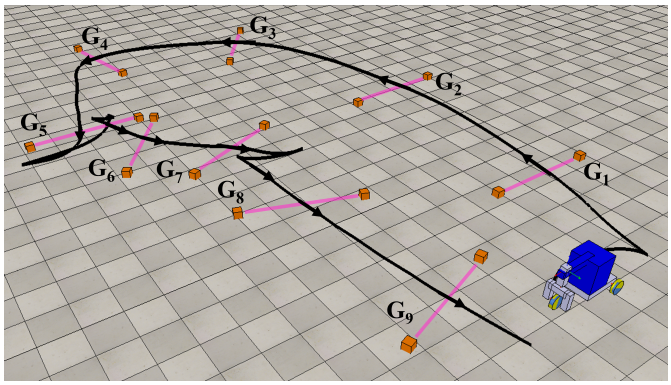


Figure 5.10: The blue tricycle navigates a course in the CoppeliaSim Simulator. The orange cuboids mark each gate towards which the bicycle must navigate and the black line marks the path towards each gate. The experiment panels are in Appendix B.2

tricycle is

$$dx_A = \frac{Lv_R/\cos\phi}{L + \frac{w}{2}\tan\phi}\cos(\phi+\theta)dt, \tag{5.33}$$

$$dy_A = \frac{Lv_R/\cos\phi}{L + \frac{w}{2}\tan\phi}\sin(\phi+\theta)dt, \tag{5.34}$$

$$d\theta = \frac{\tan\phi}{L + \frac{w}{2}\tan\phi}v_R dt, \tag{5.35}$$

$$d\phi = u_i\ dt, \tag{5.36}$$

where v_R is the velocity of the right back wheel and w is the tricycle width. All other variables have the same definition as in (5.23)-(5.26). For the RBWD tricycle, the control variables are the right back-wheel velocity v_R and the steering rate u_i. However, the FWD trained policy is trained under the assumption that we can control the front-wheel velocity v_i, $i = 0, ...5$, (5.23)-(5.26). Therefore, we introduce a PI feedback controller for the velocity v_R. For that, we measure the velocity of the front wheel

V_A^{CSim} from CoppeliaSim to compute the error $e = V^{CSim} - v_i$ between the velocity and the velocity v_i requested by the FWD tricycle policy and control V_R by the PI controller as $v_R = K_p e + K_i \int e dt$. The implementation details are provided in Appendix A.4.

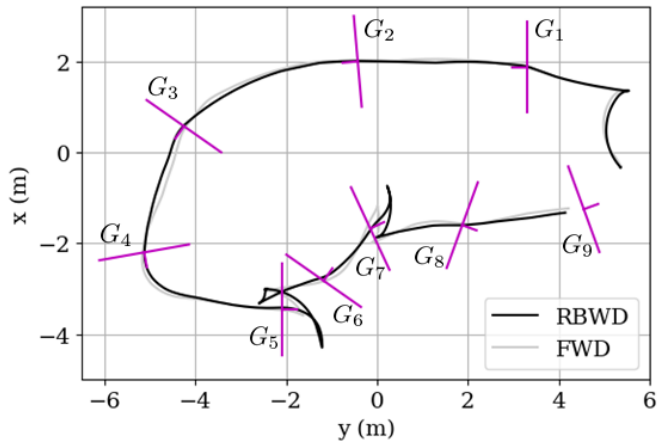


Figure 5.11: RBWD and FWD tricycle trajectories using data retrieved from the CoppeliaSim simulator. The experiment slide panels are in Appendix B.2 and B.3, respectively.

Figure 5.10 shows the RBWD tricycle at its initial position in CoppeliaSim. The tricycle is set to go through a sequence of 9 gates, G_i, $i = 1, ...9$, and in its initial position the tricycle is heading away from the first gate G_1 as shown in Fig. 5.10. In going through the gates, the tricycle accounts only for its relative position to the next gate. To go through G_1, the switching policy navigates the tricycle to go in reverse and adjusts its heading as shown in Fig. 5.11. Then, the tricycle goes forward to gate G_1 at $(x = 2, y = 3)$. Following G_1, the tricycle moves forward through gates G_2, G_3, G_4 and G_5 as shown in Fig. 5.11. After it passes G_5, to go through gate G_6 at $(x = -3, y = -1)$,

the switching policy navigates the tricycle to go forward-backward to adjust the tricycle heading for a safe passage through gate G_6. The same happens after gate G_7 when the tricycle heading is adjusted for a safe passage through gate G_8. After gate G_8, the tricycle moves forward and reaches gate G_9 at $(x = -1.5, y = 5)$.

Fig. 5.11 shows the whole path of the RBWD tricycle while going through the 9 gates. For a reference, we also provide the path that is the result of the FWD tricycle navigated by the same switching policy (for the slide panels, see Appendix B.3). While both types of tricycle can go through all 9 gates, the way they do that is different due to the significant difference in their kinematics. Overall, we show that the switching policy and the array of controllers for the FWD tricycle can be implemented on the RBWD tricycle with the kinematics that was not anticipated in the policy training.

5.6 Conclusions

In this chapter, we presented the metrics-only (MO) training of the actor-critic neural network for switching among the array of feedback controllers. The main benefit of using the MO training is that the number of variables that is in use for the neural network training depends on the number of available feedback controller performances. Therefore, the training depends on the number of feedback controllers. We compared the method with the more classical approach in which the training is based on kinematic variables (KV). Therefore, the number of variables that is used in the training is independent on the number of feedback controllers in the array. We showed that for

both pendulum control and tricycle navigation problems, the MO training method is more efficient and converges with less variability than the KV one. Furthermore, for the tricycle navigation problem, we showed that the MO trained neural network can be applied beyond numerically-simulated control problems. This work shows the benefit of using control theory method designed feedback controllers and exploiting their known performance metrics to achieve their synergistic work using the methods of artificial intelligence.

Chapter 6

Conclusions

We presented a hierarchical control architecture for the switching among multiple feedback controllers. The architecture is composed of an array of feedback controllers and a neural network which is trained to switch among them. In this work, we used the architecture for bicycle and tricycle navigation problems, as well as for the control of a pendulum mechanism. In the navigation problems, the feedback controllers were computationally generated, while in the pendulum case, the controllers were defined analytically. The neural network was implemented as an actor-critic network in PyTorch and was trained using reinforcement learning. In this thesis, we specifically tested two approaches to training the neural-network for switching. In one approach, which we denoted as KV, we fed to the neural network the variables describing the motion of the system we controlled. In the other, i.e., the MO approach, we fed the variables related to the feedback control performances.

In this thesis we found experimentally that in all examples the MO training

approach was more efficient and converged with less variability than the KV training approach. We find these results meaningful since the MO training not only takes into account controller performances, but also their number, which is important for practical applications requiring a large number of controllers.

Future work can be related to a mathematical analysis of the proposed architecture, its properties and performance in learning. Related works in this direction are using Koopman operators for Dynamic Mode Decomposition [23],[20] Perron-Frobenius theory for the analysis on a graph neural network [25], or results related to stochastic switching diffusion [60].

Appendix A

A.1 Pendulum Control: Neural Network Training

Controllers: In the case of the pendulum control problem from Section IV, we train the actor-critic neural networks for switching among four controllers u_0, $i = 0, 1, 2, 3$, from [56], which are given by (5.19)-(5.21) in the paper. From (5.16) and (5.18) of the paper we can find

$$\frac{d}{dt}\left(L(\mathbf{x})\right) = \omega \sin\theta - \omega\dot{\omega} = \omega \sin\theta - g\omega \sin\theta - \omega u \tag{A.1}$$

This yield that a rapid decrease of the Lyapunov function $L(\mathbf{x})$ can be achieved by $u = u_{max}$ for $\omega > 0$ and $u = -u_{max}$ for $\omega < 0$. Because of that we have controllers u_0 and u_1 given by (5.19) in the paper. However, difficulties arise for $\dot{\omega} = 0$ and $\theta \neq 0$, i.e., when the pendulum is not in the desired $\theta = 0$ position. Then the expression $EA(u, \theta, \omega)$ used in the formulation for u_2 and u_3 helps avoid reaching these equilibrium points by checking for low angular velocities and reducing the torque by $\frac{1}{2}$ when using the u_3 controller or by $\frac{1}{4}$ when using the u_4 controller. Even by using these four controllers, it can take a long time to reach the upright position. During the training, if

an episode lasts longer than 1000 steps the episode has timed out and we set the reward $r_{1000} = -10^4$ and finish the episode.

Training: The pendulum training of the actor-critic networks uses the prediction horizon value $n = 30$ in (5.9)-(5.10) of the paper. Both networks consist of 2 hidden layers with each layer having 256 neurons and each neuron activates according to the PReLU activation function. The neuron weight parameters are optimized with the Adam algorithm, their gradients are clipped to a value of 0.1 and are updated after each epoch with a learning rate of 0.0003. All training is coded in PyTorch.

A.2 Pendulum Control: Stochastic Optimal Controller

To get an insight into control performances obtained from the KV and MO training methods, we computed a minimum-time stochastic optimal feedback control policy $u_{OPT}(\mathbf{x})$, $\mathbf{x} = (\theta, \omega)$. The policy minimizes expected time to reach the set of states $\sqrt{\theta^2 + \omega^2} < 0.1$ of a pendulum dynamics defined as

$$d\theta = \omega dt + 0.01 dw \tag{A.2}$$

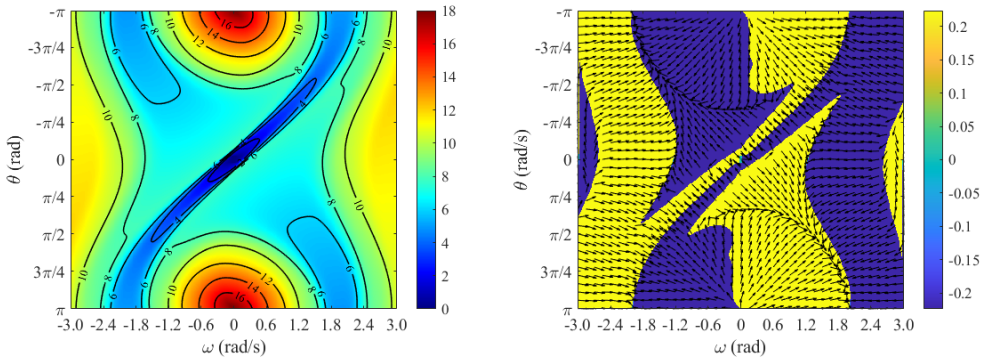
$$d\omega = g \sin\theta dt + u dt \tag{A.3}$$

The dynamics is identical to the one from (5.15)-(5.16) of the paper, except for the unit intensity Wiener process dw multiplied by 0.01. This small intensity stochastic term allows us to solve the minimum time optimal control problem using locally consistent Markov chain approximation method from [34]. Using the method we computed the

value function of the policy u_{OPT} (see Fig. A.1a)

$$V(\mathbf{x}) = \min_{u \in \mathcal{U}} E\left\{ \int_0^{\tau} 1 dt \right\} \tag{A.4}$$

where E is the expectation operator and τ is the terminal time the pendulum reaches the desired upright position. The corresponding optimal policy u_{OPT} is depicted in Fig. A.1b.



(a) (b)

Figure A.1: Minimum time stochastic optimal control: (a) Level curves of the minimum time-to-go value function $V(\mathbf{x})$; (b) Stochastic optimal feedback control u_{OPT}.

A.3 FWD Tricycle Navigation: Neural Network Training

The training of the actor-critic neural networks for the front-wheel drive (FWD) navigation is based on the prediction horizon $n = 20$ in expressions (5.9)-(5.10) of the paper. Both actor and critic networks consist of 2 hidden layers with each layer having 256 neurons. Each neuron activates according to the PReLU activation function. The

neuron weight parameters are optimized with the Adam algorithm, their gradients are clipped to a value of 0.1 and are updated after each epoch with a learning rate of 0.0005. All training is coded in PyTorch.

For both the KV and MO training methods we use episodes in which the initial FWD tricycle-gate relative positions $\mathbf{x}_0 = (r_0, \beta_0, \alpha_0, \phi_0)$ are defined as

$$\beta_0 = \psi_0 - \theta_0,\ \alpha_0 = \xi_G - \theta_0,\ \phi_0 = 0 \tag{A.5}$$

where r_0, ψ_0, θ_0 and ξ_G are independent random variables sampled as

$$\begin{aligned} &r_0 \sim U(0, \overline{p}/2),\ \psi_0 \sim U\left(-\tfrac{\pi}{4}, \tfrac{\pi}{4}\right), \\ &\theta_0 \sim U\left(-\tfrac{50\pi}{180}, \tfrac{50\pi}{180}\right), \xi_G \sim \mathcal{N}\left(0, \left(\tfrac{0.1\pi}{180}\right)^2\right), \end{aligned} \tag{A.6}$$

In this expression, $U(a, b)$ denotes a uniform random variable distribution in the range $[a, b]$, and $\mathcal{N}(\mu, \sigma^2)$ denotes a Gaussian distribution with a mean value μ and variance σ^2. For the distribution of r_0, the upper bound includes $\overline{p} = 2\pi L/\tan(\phi_{max})$, $\phi_{max} = 50\pi/180$, which is the turning radius of the back wheel for the maximal steering angle ϕ. With this we select a value compatible to the scale, i.e., the turning radius of the tricycle. The episode time step is $T_s = 0.1s$.

A.4 RBWD Tricycle Navigation: Controller Implementation

After the training for the front-wheel drive (FWD) tricycle switching policy, we used the trained switching policy and validated using the right-back-wheel drive (RBWD) tricycle. To achieve that we used front wheel velocities used during the training

as v_A^{ref} inputs for the PI controller of the front wheel velocity v_A of the RBWD tricycle, as depicted in Fig. A.2. The PI controller uses the the difference $e = v_A^{ref} - v_A$ and computes the control action $v_R = K_p e + K_i \int e dt$, which is the velocity of the right-back wheel drive of the RBWD tricycle. We use the discrete time implementation of the controller as

$$
\begin{aligned}
e &= v_A^{ref}(k) - v_A(k) && \text{(A.7)} \\
u_p(k) &= K_p e && \text{(A.8)} \\
u_i(k) &= u_i(k-1) + K_i T_s e && \text{(A.9)} \\
v_R(k) &= u_p(k) + u_i(k) && \text{(A.10)} \\
& && \text{(A.11)}
\end{aligned}
$$

where u_p and u_i are proportional and integral part of the PI controller output. k indicates the discrete time step and $T_s = 50ms$ the sample time of the feedback control loop. In our work the PI control parameters are $K_p = 0.1$ and $K_i T_s = 0.3$. This controller was implemented in a Python script that takes v_A measurements from the RBWD tricycle created in the CoppeliaSim environment, and send the control actions v_R back to the CoppeliaSim environment.

Fig. A.3 shows the plots of the v_A^{ref}, v_A and v_R during the experiment depicted in Fig. 5.10 and Fig. 5.11. For the slide panels of the experiment see, Appendix B. The plots in Fig. A.3 show the discrepancy between v_A^{ref} from switching policy for the FWD tricycle and actual front-wheel velocity v_A of the RBWD tricycle in the CoppeliaSim environment. This discrepancy is due to the more complex dynamics of the RBWD

tricycle.

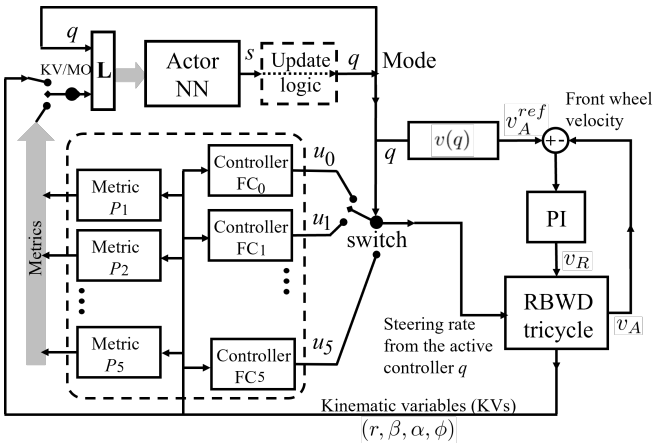


Figure A.2: Controller implementation for RBWD tricycle navigation: The neural network (actor) switches among the array of controllers that steer the tricycle. The network is trained based on the FWD tricycle kinematics model, while the tricycle in the experiment is the right-back wheel drive (RBWD) tricycle. The proportional-integral (PI) controller takes $v_A^{ref} = v(q)$ values generated by the switching policy and controls the front wheel velocity v_A of the RBWD tricycle, $v(0) = -0.6$, $v(q) = 0.6 + 0.2 \cdot (q - 1)$ for $q = 1, 2, ...5$.

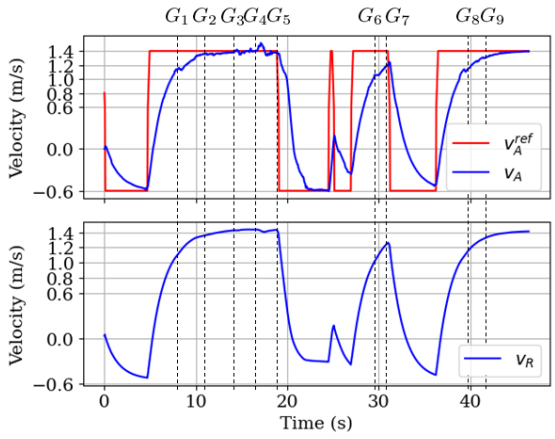


Figure A.3: The tricycle must first reverse and then navigate forward to reach the 1^{st} gate G_1 at 8.2 s. Navigating at maximum velocity the tricycle goes through $G_2 - G_4$ gates and on the G_5 gate at 18.9 s the tricycle must reverse and adjust towards the next gate. Going towards the G_6 gate it adjusted a second time navigating forward and backward at about 25.0 s and finally position itself to be able to reach the G_6 gate at 29.6 s going forward. To reach the G_8 gate the tricycle has to readjust after reaching the G_7 at 31.0 s. The last two gates $G_8 - G_9$ are reached as the tricycle navigates forward at maximum forward velocity at 39.4 s and 42.0 s, respectively.

Appendix B

B.1 Vehicle Wheel-Ground Interaction in CoppeliaSim

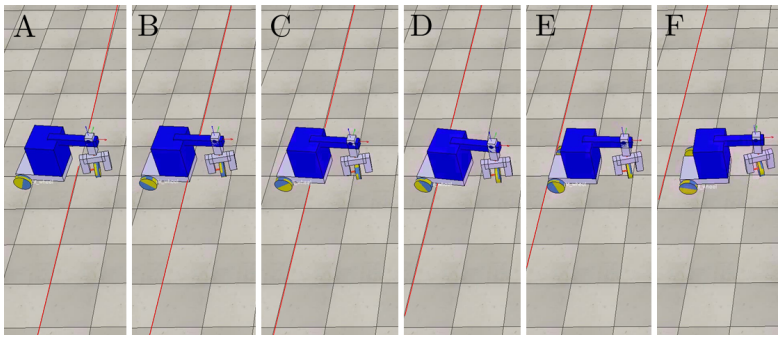


Figure B.1: Panels A-F show the tricycle's interaction (slipping) with the ground where the red line depicted is used as a reference line. This is the right back-wheel drive (RBWD) tricycle with the steering that makes the vertical axis of the tricycle body rotation go through the center of the driving right back wheel. Consequently, the front wheel slips on the ground and the tricycle motion does not correspond to the steering of the front wheel.

B.2 Right Back-Wheel Drive (RBWD) Tricycle Navigation

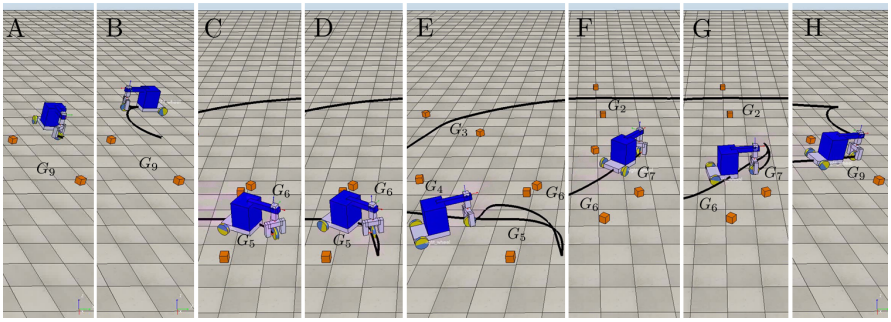


Figure B.2: The trajectory corresponds to the RBWD tricycle trajectories of Fig. 5.10-5.11. Panel A shows the tricycle at an initial pose facing away from its first gate (G_1). It reverses and adjusts to navigate forward in panel B. Going forward it reaches G_5 and G_6 is too steep of a turn towards it as shown in panel C. The tricycle must reverse and adjust to navigate forward again as depicted in panels D-E. The same occurs in panels F-G going from G_7 to G_8. Finally, it reach G_9 in panel H.

B.3 Front-Wheel Drive (FWD) Tricycle Navigation

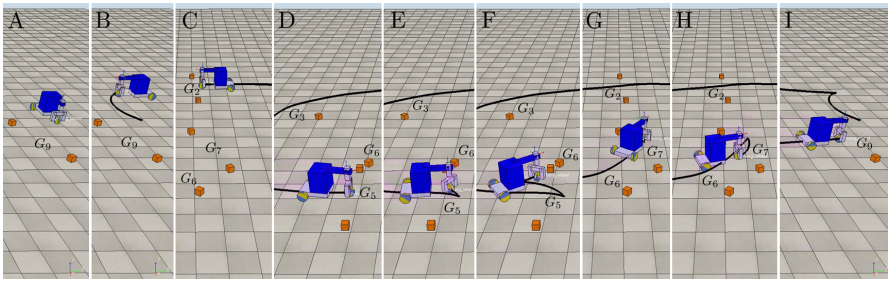


Figure B.3: The trajectory corresponds to the FWD tricycle trajectory of Fig. 5.11. Panel A shows the tricycle at an initial pose facing away from its first gate (G_1). It reverses and adjusts to navigate forward in panel B. In panels C-D the tricycle navigates forward. Upon reaching G_5, the gate G_6 is too steep to reach. So in panels D-F, the tricycle navigates backwards and forwards adjusting its way to reach G_7 depicted in panel G. One final backward and forward adjustment in panels G-H and the tricycle successfully navigates towards G_9 in panel I.

Bibliography

[1] Hany Abdulsamad and Jan Peters. Hierarchical decomposition of nonlinear dynamics and control for system identification and policy distillation. In *Proceedings of Machine Learning Research*, volume 120, pages 904–914, The Cloud, 10–11 Jun 2020. PMLR.

[2] M. Aicardi, G. Casalino, A. Bicchi, and A. Balestrino. Closed loop steering of unicycle like vehicles via lyapunov techniques. *IEEE Robotics Automation Magazine*, 2(1):27–35, Mar 1995.

[3] R. Anderson and D. Milutinović. Anticipating stochastic observation loss during optimal target tracking by a small aerial vehicle. In *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 278–287, May 2013.

[4] R. P. Anderson and D. Milutinović. A stochastic approach to dubins vehicle tracking problems. *IEEE Transactions on Automatic Control*, 59(10):2801–2806, Oct 2014.

[5] ANVEL contributors. Anvel 3.1. https://anvelsim.com/home, 2018. [Online; accessed 9-September-2018].

[6] Michael Athans and Peter L. Falb. *Optimal Control: An Introduction to the Theory and Its Applications*. Dover, 2007.

[7] H. Axelsson, M. Egerstedt, and Y. Wardi. Optimal switching surfaces in behavior-based robotics. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 1954–1959, 2006.

[8] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), 2017.

[9] J. A. Bagnell, D. Bradley, D. Silver, B. Sofman, and A. Stentz. Learning for autonomous navigation. *IEEE Robotics Automation Magazine*, 17(2):74–84, June 2010.

[10] J. C. Becker and A. Simon. Sensor and navigation data fusion for an autonomous vehicle. In *Proceedings of the IEEE Intelligent Vehicles Symposium 2000 (Cat. No.00TH8511)*, pages 156–161, Oct 2000.

[11] D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[12] M. S. Branicky. Multiple lyapunov functions and other analysis tools for switched and hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):475–482, 1998.

[13] Marco A. Carmona, Faust Aleksandra, and Dejan Milutinović. Metrics-only train-

ing of neural networks for switching among an array of feedback controllers. In *submitted to IEEE Open Journal of Control Systems*, 2022.

[14] Marco A. Carmona, Dejan Milutinović, and Faust Aleksandra. Metrics-only training neural network for switching among an array of feedback controllers for bicycle model navigation. In *2022 American Control Conference (ACC)*, 2022.

[15] Marco A. Carmona, Alexey A. Munishkin, Megan Boivin, and Dejan Milutinović. Stochastic optimal approach to the steering of an autonomous vehicle through a sequence of roadways. In *2019 American Control Conference (ACC)*, pages 3279–3284, 2019.

[16] L. Chen and K. S. Narendra. Nonlinear adaptive control using neural networks and multiple models. In *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No.00CH36334)*, volume 6, pages 4199–4203 vol.6, 2000.

[17] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *CoRR*, abs/1806.07366, 2018.

[18] H. L. Chiang, A. Faust, M. Fiser, and A. Francis. Learning navigation behaviors end-to-end with autorl. *IEEE Robotics and Automation Letters*, 4(2):2007–2014, 2019.

[19] Jongeun Choi and Dejan Milutinović. Tips on stochastic optimal feedback control and bayesian spatiotemporal models: Applications to robotics. *Journal of Dynamic Systems, Measurement and Control, Transactions of the ASME*, 137(3), 3 2015.

[20] Akshunna S. Dogra and William T. Redman. Optimizing neural networks via koopman operator theory. *CoRR*, abs/2006.02361, 2020.

[21] Weinan E, Chao Ma, Stephan Wojtowytsch, and Lei Wu. Towards a mathematical understanding of neural network-based machine learning: what we know and what we don't. *CoRR*, abs/2009.10713, 2020.

[22] A. Francis, A. Faust, H. T. L. Chiang, J. Hsu, J. C. Kew, M. Fiser, and T. W. E. Lee. Long-range indoor navigation with PRM-RL. *IEEE Transactions on Robotics*, 36(4):1115–1134, 2020.

[23] Keisuke Fujii, Yuki Inaba, and Y. Kawahara. Koopman spectral kernels for comparing complex dynamics: Application to multiagent sport plays. In *ECML/PKDD*, 2017.

[24] Mrinal K. Ghosh, Aristotle Arapostathis, and Steven I. Marcus. Ergodic control of switching diffusions. *SIAM Journal on Control and Optimization*, 35(6):1952–1988, 1997.

[25] Fangda Gu, Heng Chang, Wenwu Zhu, Somayeh Sojoudi, and Laurent El Ghaoui. Implicit graph neural networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 11984–11995. Curran Associates, Inc., 2020.

[26] João P. Hespanha, Daniel Liberzon, and A. Stephen Morse. Logic-based switching

control of a nonholonomic system with parametric modeling uncertainty. *Systems Control Lett*, 38:167–177, 1999.

[27] D. Iberraken, L. Adouanc, and D. Denis. Multi-controller architecture for reliable autonomous vehicle navigation: Combination of model-driven and data-driven formalization. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 245–251, 2019.

[28] D. Iberraken, L. Adouanc, and D. Denis. Multi-controller architecture for reliable autonomous vehicle navigation: Combination of model-driven and data-driven formalization. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 245–251, 2019.

[29] S. J. Julier and H. F. Durrant-Whyte. On the role of process models in autonomous land vehicle navigation systems. *IEEE Transactions on Robotics and Automation*, 19(1):1–14, Feb 2003.

[30] Hassan K. Khalil. *Nonlinear Systems*. Prentice Hall, 2002.

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

[32] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 1094–1099, June 2015.

[33] X. D. Koutsoukos, P. J. Antsaklis, J. A. Stiver, and M. D. Lemmon. Supervisory control of hybrid systems. *Proceedings of the IEEE*, 88(7):1026–1049, 2000.

[34] H. Kushner and P.G. Dupuis. *Numerical Methods for Stochastic Control Problems in Continuous Time*. Springer, 2001.

[35] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[36] A. Leonessa, W. M. Haddad, and V. S. Chellaboina. Nonlinear system stabilization via hierarchical switching control. *IEEE Transactions on Automatic Control*, 46(1):17–28, 2001.

[37] B. Lincoln and B. Bernhardsson. Lqr optimization of linear system switching. *IEEE Transactions on Automatic Control*, 47(10):1701–1705, 2002.

[38] J. Malmborg, B. Bernhardsson, and K.J. Åström. A stabilizing switching scheme for multi controller systems. volume 29, pages 2627–2632, 1996.

[39] D. Milutinović, D. W. Casbeer, D. Kingston, and S. Rasmussen. A stochastic approach to small uav feedback control for target tracking and blind spot avoidance. In *2017 IEEE Conference on Control Technology and Applications (CCTA)*, pages 1031–1037, Aug 2017.

[40] D. Milutinović and David W. Casbeer. Automatic alignment of motion capturing system markers to a small differential drive robot's position and heading angle. In *2015 ASME Dynamic Systems and Control Conference*, volume 3, 2015.

[41] Dejan Milutinović and Pedro Lima. Modeling and optimal centralized control of a large-size robotic population. *IEEE Transactions on Robotics*, 22(6):1280–1285, 2006.

[42] Dejan Milutinović and Lima Pedro. *Cells and Robots Modeling and Control of Large-Size Agent Populations*. Springer, 2007.

[43] Dejan Milutinović, David W. Casbeer, and Meir Pachter. Markov inequality rule for switching among time optimal controllers in a multiple vehicle intercept problem. *Automatica*, 87:274 – 280, 2018.

[44] A. A. Munishkin, D. Milutinović, and D.W. Casbeer. Min–max time efficient inspection of ground vehicles by a uav team. *Robotics and Autonomous Systems*, 125:103370, 2020.

[45] A.A. Munishkin, A. Hashemi, D.W. Casbeer, and D. Milutinović. Scalable markov chain approximation for a safe intercept navigation in the presence of multiple vehicles. *Autonomous Robots*, 43(3):575–588, 2019.

[46] K. S. Narendra and J. Balakrishnan. Improving transient response of adaptive control systems using multiple models and switching. *IEEE Transactions on Automatic Control*, 39(9):1861–1866, 1994.

[47] K. S. Narendra, J. Balakrishnan, and M. K. Ciliz. Adaptation and learning using multiple models, switching, and tuning. *IEEE Control Systems Magazine*, 15(3):37–51, 1995.

[48] B. Øksendal. *Stochastic Differential Equations: An Introduction with Applications.* Springer, 2003.

[49] Aaron Palmer and Dejan Milutinović. A hamiltonian approach using partial differential equations for open-loop stochastic optimal control. In *Proceedings of the 2011 American Control Conference*, pages 2056–2061, 2011.

[50] 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, 2006.

[51] PyTorch contributors. From research to production. https://pytorch.org, 2020. [Online; accessed 16-February-2022].

[52] G. Saridis. Intelligent robotic control. *IEEE Transactions on Automatic Control*, 28(5):547–557, 1983.

[53] J. Si, A. G. Barto, W. B. Powell, and D. Wunsch. *Supervised ActorCritic Reinforcement Learning*, pages 359–380. John Wiley & Sons, Ltd, 2004.

[54] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 2018.

[55] Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999.

[56] Andrew G. Barto Theodore J. Perkins. Lyapunov design for safe reinforcement learning. *Journal of Machine Learning Research*, 3:803–832, 2002.

[57] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.

[58] Juan Marcos Toibero, Flavio Roberti, Ricardo Carelli, and Paolo Fiorini. Switching control approach for stable navigation of mobile robots in unknown environments. *Robotics and Computer-Integrated Manufacturing*, 27(3):558 – 568, 2011.

[59] R. Wang, J.C. Kew, D. Lee, T-W. Lee, T. Zhang, B. Ichter, J. Tan, and A. Faust. Model-based reinforcement learning for multiagent goal alignment. *Conference on Robot Learning (CoRL)*, 2020.

[60] George G. Yin and Chao Zhu. *Hybrid Switching Diffusions*. Springer, 2010.

[61] J. Yong and X.Y. Zhou. *Stochastic controls Hamiltonian Systems and HJB Equations*. Springer-Verlag, 1999.

[62] P. Zhao, J. Chen, Y. Song, X. Tao, T. Xu, and T. Mei. Design of a control system for an autonomous vehicle based on adaptive-pid. *International Journal of Advanced Robotic Systems*, 9(2), 2012.