

Learning to Concurrently Execute Multiple Robotic Tasks – Analysis and Optimization-Based Synthesis

by

Sheikh A. Tahmid

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2025

© Sheikh A. Tahmid 2025

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

Chapter 1, Chapter 2, Chapter 3, Section 4.1 and Sections 5.1 - 5.4 borrow content from [41] which was presented in the 24th International Conference on Autonomous Agents and Multiagent Systems.

Abstract

In this thesis, we develop and analyze a general framework for learning and concurrently executing multiple robotic control tasks learned within the paradigm of reinforcement learning – each encoded as a value function and executed together as constraints in a pointwise min-norm controller. Applying reinforcement learning to robotic control tasks is an active area of research as it has the potential to generalize to complex tasks. For robots that are redundant – meaning they have the physical capability of executing multiple tasks at the same time – it can be useful to learn to concurrently execute multiple control objectives. Previous and relatively recent work proposes a model-based approach of concurrently executing multiple tasks learned using reinforcement learning by incorporating the learned value functions as constraints in a pointwise min-norm controller. This work, however, does not enforce nor encourage any constraints on the learned value functions themselves, causing there to be no guarantees on how trained tasks execute together. Building on this approach, this thesis proposes ideas for enabling learned value functions to be concurrently executable when combined using the proposed pointwise min-norm controller. Firstly, we define notions of independence, orthogonality, concurrent executability and concurrent controllability to describe the relationships between value functions and their ability to execute together concurrently, based on the Lie derivatives of each value function. We then propose the idea of an “interference cost” which we prove can be used to train a set of value functions to be independent – meaning that their Lie derivatives are linearly independent, making it possible to pick inputs which concurrently execute each task. This interference cost can be used with any reinforcement learning algorithm, but we also propose a variant of fitted value iteration which can be used to efficiently learn such value functions. Additionally, we propose theory that provides necessary and sufficient conditions for value functions to be concurrently executable, using the proposed pointwise min-norm controller, within subsets of the state space. Finally, we test our proposed methods and theorems on simulations involving teams of mobile robots.

Acknowledgements

I would like to thank my supervisor Prof. Gennaro Notomista for accepting me as a student and being a great supervisor. I did not know what robotics and control theory graduate school would be like, but I enjoyed it a lot more than I thought I would and a big part of that was getting to work with Gennaro. I would like to thank the readers Prof. Chris Nielsen and Prof. Kate Larson for taking the time to read this thesis and provide valuable feedback. I would like to thank everyone who has been part of the young Ecological and Resilient Autonomous roBots Lab and every other graduate student that I interacted with the past two years. Speaking with, working with and/or wasting time with all of you has made the experience much more enjoyable than it otherwise would have been. Last but not least, I would like to thank my parents. I would not have had the opportunities that I have had without them deciding to move to this country years ago and making all the sacrifices that they did.

Table of Contents

Author’s Declaration	ii
Statement of Contributions	iii
Abstract	iv
Acknowledgements	v
List of Figures	ix
List of Tables	x
1 Introduction	1
2 Related Work	5
2.1 Reinforcement Learning for Robotic Control Tasks	5
2.2 Multi-Objective Reinforcement Learning	6
2.3 Definitions of Robotic Control Tasks and Task Independence	7
2.4 Composition of Learned Control Tasks	7
2.5 Constrained Optimization-Based Control Paradigm	7

3	Mathematical Background and Problem Formulation	9
3.1	Assumptions on System and Learned Control Tasks	9
3.2	The Pointwise Min-Norm Optimization Based Controller	11
3.2.1	Slack Variables for Task Prioritization	11
3.2.2	Online and Real-Time Execution	12
3.2.3	Relation to Optimal Controller	13
3.3	Definitions of Relationships Between Value Functions	13
3.3.1	Independence and Orthogonality	14
3.3.2	Concurrent Executability and Controllability	14
3.4	Continuous Fitted Value Iteration	15
3.4.1	Implementation Details of Continuous Fitted Value Iteration	17
3.4.2	Use of Continuous Fitted Value Iteration to Estimate Different Kinds of Value Functions	17
3.5	Summary of Problems and Proposed Solution	19
4	Main Results	20
4.1	Learning Independent Value Functions	21
4.1.1	Interference Cost for Independent Value Functions	21
4.1.2	Variant of Continuous Fitted Value Iteration	26
4.1.3	Implementation Details for Our Variant of Continuous Fitted Value Iteration	30
4.2	Conditions for Concurrent Executability	30
4.2.1	Necessary Conditions for the Concurrent Executability of Learned Value Functions	31
4.2.2	Existence of a Concurrently Executable Set of States	33
4.3	Summary of Results	35

5	Simulations and Experiments	37
5.1	Going to a Point and Avoiding a Square Region	37
5.2	Forming a Shape and Avoiding a Square Region	38
5.3	Forming a Shape and Going to a Point while Avoiding a Square-Shaped Region	41
5.4	Experiment with Physical Robots	45
5.5	Illustrating Necessary and Sufficient Conditions for Concurrent Executability	46
6	Conclusion and Future Work	50
	References	53
	APPENDICES	59
A	Implementation Details	60

List of Figures

5.1	(a): Heatmap of function, $\tilde{J}_1$, trained for avoidance task. (b): Heatmap of function, $\tilde{J}_{\text{base}}$, trained using CFVI. (c): Heatmap of function, $\tilde{J}_{\text{ind}}$, trained using proposed method to be independent to avoidance task. Note that gradients of $\tilde{J}_{\text{ind}}$ appear to be linearly independent to those of $\tilde{J}_1$ as values in (c) appear to warp around the square region.	39
5.2	(a): Trajectory when combining avoidance task, $\tilde{J}_1$, with baseline go-to-point task, $\tilde{J}_{\text{base}}$. (b): Trajectory when combining $\tilde{J}_1$ with go-to-point task trained using proposed approach, $\tilde{J}_{\text{ind}}$. Green: Positions at each time step. Red: Final position for each trajectory. Blue: Region to avoid.	39
5.3	Robot team forming triangle while avoiding region. Coloured Dots: Robots 1-3. Lighter Blue: Region to avoid. Grey: Trajectories of each robot. . . .	42
5.4	Robot team successfully forming triangle, sending first robot to origin and avoiding regions. Coloured Dots: Robots 1-3. Lighter Blue: Regions to avoid. Grey: Trajectories of each robot.	44
5.5	Physical team of robots forming a shape and sending one robot to a point while avoiding an obstacle	45
5.6	Heatmaps of $\tilde{J}_1$ and $\tilde{J}_2$ respectively	46
5.7	Heatmap of angles between the gradients of $\tilde{J}_1$ and $\tilde{J}_2$. Points that are closer to red are points where $\tilde{J}_1$ and $\tilde{J}_2$ are not concurrently executable.	47
5.8	Heatmap of $\tilde{J}_3$	48
5.9	Heatmap of angles between the gradients of $\tilde{J}_2$ and $\tilde{J}_3$. Points that are closer to red are points where $\tilde{J}_2$ and $\tilde{J}_3$ are not concurrently executable.	49

List of Tables

5.1	Success Rates of Avoiding Region and Forming a Triangle	41
5.2	Success Rates of Avoiding Region, Going to Origin and Forming a Triangle	43

Chapter 1

Introduction

Training a robot to be able to concurrently execute multiple control objectives is an active area of research in robotics and Reinforcement Learning (RL). Using RL for robotic control tasks is promising as it has the potential to generalize to complex tasks, as opposed to more classical methods which often require particular assumptions of the system and control objectives. Algorithms for learning to execute multiple tasks would be especially useful for robotic systems that have the physical capability of executing multiple tasks concurrently — also known as redundancy — such as multi-robot systems [23], [35], [16], manipulators [23], [25], [38] and humanoids [46], [10]. Multi-objective RL [15], [36], which aims to train an agent to solve multiple possibly competing objectives, is a possible approach that has been applied to robotic control problems [45], [17]. While multi-objective RL methods may work, they generally result in policies that cannot execute subsets of the objectives or reprioritize tasks during online execution.

In this thesis, we develop and analyze an alternative approach where we instead train multiple tasks using RL to be compatible with each other. This allows us to combine possibly time-varying subsets of tasks which may be useful in many systems such as those operating over long durations where objectives are likely to change over time [8]. Furthermore, several works that study concurrently executing multiple tasks using redundant robots focus on executing tasks in a prioritized fashion [4], [25], [30], [28]. This is useful in robotics applications where one should, for instance, prioritize safety-critical tasks over others. The approach in this thesis extends this idea as our method encourages each task’s learned policy to not interfere with the execution of higher priority tasks.

Several works propose methods for combining multiple deep RL policies together [43], [14], [32], [44], [26]. We opt to make use of the work in [26] which combines multiple value

functions implemented as neural networks—which may be learned through various methods within RL [13], [21]—into a single pointwise, min-norm, optimization-based controller capable of concurrently executing each task in a prioritized fashion. The controller in [26] allows us to execute possibly time-varying subsets of tasks with possibly time-varying priorities. The work in [26], however, does not enforce nor encourage any constraints on the learned functions themselves, causing there to be no guarantees on how trained tasks execute together.

In this thesis, we propose a particular cost functional that can be used to encourage learned value functions to satisfy a notion of independence, defined based on definitions of independence in [3] and [30], which allows the tasks to be executed concurrently. We also propose a variant of the continuous fitted value iteration method proposed in [21] to be able to efficiently approximate this proposed cost functional. We also develop propositions for analyzing the ability to concurrently execute value functions. In addition to our notion of independence, we also define notions of orthogonality, concurrent executability and concurrent controllability which our propositions refer to. We provide necessary and sufficient conditions for being able to concurrently execute a set of learned value functions within a subset of the state space as well as necessary and sufficient conditions for a learned value function to be orthogonal to a set of other learned value functions.

The methods used and proposed in this thesis can be grouped with the paradigm of model-based RL. We assume in this thesis that we have a reasonable model of the robotic system which we wish to control. We use the model of the robot in order to define relationships between value functions that characterize the ability to concurrently execute learned tasks. We also rely on the model to train tasks to be compatible and the work in [26], which we utilize, relies on the model in order to pick inputs at each time step that makes progress on each learned task during execution. While most current work in RL is within model-free RL, we believe that assuming one has a nominal model of the systems is a reasonable assumption in robotics.

We believe our proposed approach can be especially useful in applications that involve the coordinated control of multi-robot systems. In persistent environmental monitoring with multiple robots [22], [29], for example, a team of robots must efficiently explore and monitor the given environment while maintaining battery levels. Being able to switch priorities or the set of tasks being executed would also be useful in such an application in order to respond to emergencies or monitor particular areas of interest. Multi-robot systems are, in general, not differentially flat [6], resulting in it being difficult to plan their behaviour using classical methods. As a result, RL has shown to be a useful tool in controlling multi-robot systems to perform complex tasks [29].

The main contributions of this thesis are as follows:

- We define notions of task *independence*, *orthogonality*, *concurrently executability* and *concurrently controllability* for learned cost-to-go functions that characterize the ability to concurrently execute tasks together.
- We propose a form of a cost functional and show with a proof that a set of tasks can be made *independent*, and therefore possible to execute together concurrently, by solving our proposed cost functional.
- We propose a version of fitted value iteration [5] for the continuous control setting, similar to [21], to directly learn our proposed cost functional within the deep RL paradigm.
- We propose necessary and sufficient conditions for being able to concurrently execute multiple learned value functions within a subset of the state space.
- We demonstrate our proposed approaches and theorems in several scenarios involving a robotic system executing multiple learned tasks.

The remainder of this thesis will be structured as follows:

- Chapter 2: In this chapter we will present related works in the literature. We will discuss work related to reinforcement learning for robotic control tasks, multi-objective reinforcement learning, definitions of robotic control tasks as well as relationships between tasks and methods for composing learned tasks.
- Chapter 3: Here, we will present the mathematical background required for understanding the thesis and more formally describe the problems which we wish to solve.
- Chapter 4: In this chapter, we will propose theoretical results and methods to solve the problems presented in chapter 3. We will propose a particular cost functional for encouraging learned value functions to satisfy our notion of independence as well as a variant of fitted value iteration for approximating this proposed cost functional. We will also propose a few propositions for analyzing the relationship between learned value functions and the ability to concurrently execute multiple tasks within subsets of the state space.
- Chapter 5: Here, we will demonstrate the theoretical results and ideas presented in chapter 4 on simulations of multi-robot systems trained to concurrently execute multiple learned tasks.

- Chapter 6: To conclude the thesis, we will summarize the ideas and work presented in the thesis as well as discuss possible future directions.

Chapter 2

Related Work

This thesis is concerned with training robotic control tasks using RL in such a way that multiple tasks can be executed concurrently, which is a goal shared by existing multi-objective RL methods. Sections 2.1 and 2.2 discuss RL for robotic control tasks and multi-objective RL respectively. In the next chapter, we define notions of independence, orthogonality, concurrent executability and concurrent controllability between learned value functions. These are similar to definitions of independence and orthogonality for Jacobian-based and Extended Set-Based tasks from existing work discussed in Section 2.3. As we will discuss in Section 2.4, we combine learned tasks using a min-norm controller proposed in [26], which unlike other similar works, allows us to execute learned tasks in possibly time-varying prioritized stacks. The work in [26] builds upon the constraint-based control paradigm proposed in [27] which we describe in Section 2.5.

2.1 Reinforcement Learning for Robotic Control Tasks

Reinforcement Learning (RL) is a field within machine learning concerned with training agents to behave optimally in an environment over time. Utilizing RL for robotics is an active area of research as RL has the potential to handle situations with high-dimensional state spaces or highly non-linear dynamics which conventional approaches may not be able to handle effectively [42].

RL has connections to the field of optimal control theory [5]. RL typically focuses on training agents to maximize a reward or minimize a cost by learning from data collected over past interactions with the environment. Optimal control theory, on the other hand,

focuses on minimizing some cost by solving an optimal control problem analytically given a reasonable model of the system’s dynamics. Any optimal control problem can be seen as a reinforcement learning problem and when an optimal control problem is intractable, one may consider solving it using an RL algorithm. The robotic tasks considered in this thesis are represented as optimal control problems which we approximately solve using techniques within the paradigm of RL.

Existing work in deep RL for continuous control tasks in robotics predominantly use policy iteration methods such as [37], [20]. Furthermore, these methods are typically model-free methods, where the algorithms do not have explicit knowledge of how the system behaves, nor do they explicitly learn that knowledge from the data. Model-based methods, on the other hand, are RL algorithms which either make use of a known model of the system or explicitly learn it through data [33]. In robotic control tasks, it is reasonable to assume that one has a nominal model of the system. The work in [21], which extends similar work in [7], proposes the Continuous Fitted Value Iteration (CFVI) algorithm. CFVI extends the idea of classical value iteration [40] to systems where the input space is continuous instead of discrete. CFVI performs approximate dynamic programming to approximately solve optimal control problems for the continuous control setting under assumptions such as the system being control-affine. The work in [21] also compared CFVI with state-of-the-art deep RL algorithms on continuous control tasks and showed that it performed favourably. In this thesis, we propose a similar version of fitted value iteration as CFVI for a similar problem setting with different assumptions on the instantaneous cost.

2.2 Multi-Objective Reinforcement Learning

Multi-objective RL has been used to solve decision and control problems that involve optimizing for multiple objectives [15], [36]. It is possible to treat multi-objective RL as regular RL by using a linear combination of instantaneous rewards. However, this has problems discussed in [15] such as the increased complexity of reward-shaping and the lack of explainability in resulting policies. The work in [45] and [17], which both find Pareto optimal policies, show that multi-objective RL is viable in robotic control problems. However, multi-objective RL methods generally result in policies that are not separable – meaning that an objective cannot be executed on its own when desired – nor capable of complying with time-varying task priorities, unlike our proposed approach of training multiple tasks to be compatible with each other.

2.3 Definitions of Robotic Control Tasks and Task Independence

There exist previous work that mathematically define robotic control tasks, along with notions of *independence* and *orthogonality* which affect the ability to concurrently execute multiple tasks. Several works such as [39], [4] have studied executing multiple Jacobian-based tasks concurrently, often in a prioritized fashion, for redundant kinematic robots. The work in [3] analyzes the stability of such algorithms, also defining notions of independence and orthogonality between Jacobian-based tasks. Set-based tasks were introduced in [9] and the concurrent execution of multiple of them is analyzed in [24]. The work in [30] uses Extended-Set Based tasks [28] to generalize Jacobian-based tasks, as well as the definitions of independence and orthogonality introduced in [3]. The definitions in [30] can be used to describe the cost-to-go functions that we train in this paper. However, these definitions do not take into account the system dynamics, unlike Definitions 1 and 2 in Chapter 3 which do.

2.4 Composition of Learned Control Tasks

Section 2.1 in [26] summarizes some of the previous literature on composing learned deep RL policies [43], [14], [32], [44]. In our work, we opt to use the min-norm controller proposed in [26] as our method of combining multiple learned tasks as it allows us to execute possibly time-varying subsets of tasks with possibly time-varying priorities. We explain the controller in [26] further in the next chapter. While [26] proposes a method of combining multiple learned tasks, it does not offer a way to ensure that the learned tasks are compatible with each other. Our proposed method allows us to train tasks in such a way that it is possible to combine and execute them together using the method in [26].

2.5 Constrained Optimization-Based Control Paradigm

The controller in [26], mentioned in the previous section, is based off previous work in [27] which proposes the idea of combining multiple robotic control tasks together as constraints

in an optimization problem:

$$\begin{aligned}
& \min_u \quad \text{control_effort}(u) \\
& \text{s.t.} \quad c_{\text{task}_1}(x, u) \leq 0 \\
& \quad \quad \quad \vdots \\
& \quad \quad \quad c_{\text{task}_N}(x, u) \leq 0
\end{aligned}$$

where u is the input to the system and x is the state of the system. A controller of this form may be referred to as a “pointwise min-norm” controller, where at each point in time, the input with the minimal “control effort”, often quantified by the squared norm of the input, is selected that satisfies each of the constraints which are functions of the state and input. Satisfying a constraint represents accomplishing a task. Constraints could be based on Control Lyapunov Functions (CLFs) [19], designed to make progress toward some goal, or Control Barrier Functions (CBFs) [2], designed to maintain the system within some safe set. In the case where the system is control-affine and we have a set of differentiable CLFs and CBFs, the constraints could be the time derivatives of the CLFs and CBFs, leading to the pointwise min-norm controller becoming a quadratic program and easy to solve in real-time in order to control a physical robotic system. In this thesis, we treat learned value functions as CLFs, where each constraint is the time derivative of a value function. Picking inputs that drive the time derivative to be negative represents making progress on that task. Previous work in [27] also introduce the idea of slack variables for prioritizing the execution of certain tasks over others when it may not be feasible to accomplish every task.

Chapter 3

Mathematical Background and Problem Formulation

We consider the problem of using RL to learn multiple tasks for a redundant robotic system in such a way that the tasks can be executed concurrently. This chapter presents the mathematical background necessary to understand the rest of this thesis and in particular, the main theoretical results that will be presented in Chapter 4. In Section 3.1, we specify the assumptions we make on the robotic systems and type of control tasks that we aim to use RL to train for. In Section 3.2, we formally introduce the pointwise min-norm controller originally proposed in [26] which we use to simultaneously execute multiple robotic tasks learned using RL, by encoding each corresponding value function as a constraint in the optimization problem. In Section 3.3, we provide definitions of relationships between value functions which will be used later to develop theorems and algorithms related to the concurrent execution of learned tasks.

3.1 Assumptions on System and Learned Control Tasks

Firstly, we assume that the system is deterministic and control-affine. These assumptions may not be applicable to all applications. However, in robotics, it is common to assume that the dynamics are approximately known and many robotic systems, including most mobile robots and manipulators, can be modelled as control-affine systems. The dynamics can be represented with:

$$\dot{x}(t) = f(x) + g(x)u \tag{1}$$

where $x \in \mathcal{X} \subseteq \mathbb{R}^n$ and $u \in \mathcal{U} \subseteq \mathbb{R}^m$ denote the state and input, respectively, and $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ and $g : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times m}$ are continuously differentiable vector fields.

We assume that each task $1, \dots, N$ is learned sequentially in the order of importance with task 1 being the most important. This is a reasonable assumption in robotics as it is often desirable to prioritize more safety-critical tasks over others. We assume that each task i has an associated state cost, $q_i(x)$, that is nonnegative. To learn each task i , we consider solving the following infinite horizon optimal control problem:

$$J_i^*(x(t)) = \min_{u(\cdot)} \int_t^\infty e^{-\beta\tau} (q_i(x) + u^\top R_i(x)u) d\tau \quad (2)$$

where $\beta \geq 0$ and $R_i : \mathbb{R}^n \rightarrow \mathbb{S}_{++}^n$ maps each state x to a positive definite matrix. We assume that each task i has at least one reachable terminal goal state $x_T \in \mathcal{X}$ where $q_i(x_T) = 0$ and x_T is an equilibrium point. Since each task i has at least one reachable terminal goal state that is an equilibrium point, we will assume, unless otherwise stated, that each optimal control problem J_i^* is undiscounted, meaning that it is of the form:

$$J_i^*(x(t)) = \min_{u(\cdot)} \int_t^\infty (q_i(x) + u^\top R_i(x)u) d\tau \quad (3)$$

Most ideas presented in this thesis, however, can be applied to both the undiscounted and discounted case. Anything that is said of the value function in (3), can, unless otherwise specified, be said about the value function in (2). While the discounted value functions are not needed under our assumptions, they were used in practice due to discount factors making RL algorithms more likely to converge.

As the problem in (3) cannot be solved analytically in the general case, we consider using deep RL to learn parametric approximations which we will denote as $\tilde{J}_i$ for each task i . RL is typically posed as a maximization problem, but we opt to pose it as a minimization problem here, as is typically done in optimal control theory. In this thesis, we will use the terms cost-to-go function and value function interchangeably to refer to functions of the form in (3). Note that the continuous dynamics of the system described in (3) can be discretized which would allow us to use an actor-critic [13] method or version of fitted value iteration [5] to find some $\tilde{J}_i$. In the case where the input cost in (3) does not rely on the state and $R_i(x)$ is a constant, positive definite matrix, one can use the version of fitted value iteration for continuous systems proposed in [21].

3.2 The Pointwise Min-Norm Optimization Based Controller

If we learn a value function approximation $\tilde{J}_i$ for task i , one way to execute the optimal policy is to use as input the $u^*(.)$ that solves the optimization problem in (3). However, to simply make progress on task i , we can pick an input that drives $\tilde{J}_i$ toward 0 or, in other words, makes the time derivative negative at a particular state. To make progress on task i at every state $x \in \mathcal{X}$ as fast as if we were executing the optimal policy, we can pick an input $u \in \mathcal{U}$ that makes the time derivative negative by the same amount as if we were explicitly using the optimal input $u^*(x)$ at state x . This motivates the use of the pointwise min-norm controller proposed in [26].

Given a set of learned cost-to-go functions $\tilde{J}_1, \dots, \tilde{J}_N$, each implemented with a neural network, we can select inputs that execute each task concurrently using the min-norm quadratic program proposed in [26]:

$$\begin{aligned} \min_{u \in \mathcal{U}} \quad & \|u\|^2 \\ \text{s.t.} \quad & L_f \tilde{J}_1(x) + L_g \tilde{J}_1(x)u \leq -\sigma_1(x) \\ & \vdots \\ & L_f \tilde{J}_N(x) + L_g \tilde{J}_N(x)u \leq -\sigma_N(x) \end{aligned} \tag{4}$$

Note that $L_f \tilde{J}_i(x)$ and $L_g \tilde{J}_i(x)$ are Lie derivatives where $L_f \tilde{J}_i(x) = \frac{\partial \tilde{J}_i}{\partial x}(x)f(x)$ and $L_g \tilde{J}_i(x) = \frac{\partial \tilde{J}_i}{\partial x}(x)g(x)$, meaning that the lefthand side of each constraint is simply the time derivative of the cost-to-go function for a task. Since each $J_1^*, \dots, J_N^*$ is nonnegative and smaller values represent being closer to a goal state, each corresponding $\tilde{J}_1, \dots, \tilde{J}_N$ can be treated as a candidate Control Lyapunov Function (CLF). Note that since $\tilde{J}_1, \dots, \tilde{J}_N$ are implemented with neural networks, $L_f \tilde{J}_i(x)$ and $L_g \tilde{J}_i(x)$ can be calculated using back-propagation. As mentioned previously, picking an input u that makes the time derivative of a particular $\tilde{J}_i$ negative represents making progress on that task.

3.2.1 Slack Variables for Task Prioritization

There may be states where the pointwise min-norm controller shown in (4) is not feasible. To account for such states, the work in [26], based on work in [27], introduces slack variables

$\delta_1, \dots, \delta_N \in \mathbb{R}$, which form $\delta \in \mathbb{R}^N$, to the optimization problem in (4) to make the quadratic program feasible for every state $x \in \mathcal{X}$ as well as prioritize the tasks.

$$\begin{aligned}
\min_{u \in \mathcal{U}, \delta \in \mathbb{R}^N} \quad & \|u\|^2 + \kappa \|\delta\|^2 \\
\text{s.t.} \quad & L_f \tilde{J}_1(x) + L_g \tilde{J}_1(x)u \leq -\sigma_1(x) + \delta_1 \\
& \vdots \\
& L_f \tilde{J}_N(x) + L_g \tilde{J}_N(x)u \leq -\sigma_N(x) + \delta_N \\
& K\delta \geq 0
\end{aligned} \tag{5}$$

$K \in \mathbb{R}^{N \times N}$ is used to specify the possibly time-varying priorities between tasks by specifying the relationship between different values of δ_i for $i \in \{1, \dots, N\}$. Higher priority tasks are allowed less slack than lower priority ones, meaning that the controller will prioritize picking inputs close to optimal for higher priority tasks than lower priority ones. The priorities set in K are usually manually specified, but methods for *automatic task prioritization* are presented in [30]. $\kappa \geq 0$ is a hyperparameter used in the objective function in (5) to control the total amount of slack allowed by the controller at each point in time.

3.2.2 Online and Real-Time Execution

The optimization problem in (5) is designed to be able to run in the control loop of a physical robot. This requires, for every state $x \in \mathcal{X}$, to be able to form and solve the optimization problem in (5) at a quick enough speed relative to the physical movement of the robotic system. Feedback controllers for robotic systems typically run at a frequency that is anywhere between 5 and 100 Hz, with the frequency that is needed depending on the physical system itself. Backpropagation on a sequence of neural networks, $\tilde{J}_1, \dots, \tilde{J}_N$, with respect to a single state can be done very quickly, even on a CPU. This results in a quadratic, convex optimization program. Such an optimization problem can also be solved very quickly by a computer, and off-the-shelf convex optimization solvers can further limit the maximum number of iterations used in order to satisfy stricter real-time guarantees. This results in a feedback controller that forms and solves the optimization problem in (5) fast enough to control the physical movement of robots.

3.2.3 Relation to Optimal Controller

The scalar-valued functions $\sigma_1(x), \dots, \sigma_N(x)$ are positive semi-definite functions of the current state x , designed to recover inputs equivalent to the optimal input with respect to each cost-to-go function. The expressions for $\sigma_1(x), \dots, \sigma_N(x)$ are:

$$\sigma_i = \sqrt{\left(L_f \tilde{J}_i\right)^2 + q(x) L_g \tilde{J}_i \left(L_g \tilde{J}_i\right)^\top} \quad (6)$$

This expression is derived from work in [11], [34]. The work in [26] assumes that each value function $\tilde{J}_i$ fits an optimal control problem of the form $\min_{u(\cdot)} \int_t^\infty (q_i(x) + u^\top u) d\tau$, or in other words, one of the form in (3) where $R_i(x) = I$. In such a case, the expression shown in (6) recovers the optimal input for $\tilde{J}_i$ when it is the only constraint in the controller in (5). In the case where $\tilde{J}_i$ is one of several constraints in the controller in (5), each value function can be thought of as accomplishing its respective task as fast as if the optimal controller were being executed for each value function, since the time derivatives are the same as the case where the optimal input were to be executed for each value function.

In this thesis, we will later propose a cost functional where $R(x) \neq I$. However, it will still be close enough that is still reasonable to use the expressions for σ_i shown in (6). In experiments shown later in the thesis, we will also train value functions using a discount factor. However, we still believe using the expression for σ_i is still reasonable in this case as well, as the resulting inputs should still be close enough to optimal given a slight discount factor, meaning a value of β close to 0 in the continuous formulation (2).

Practitioners are also free to use the controller in (5), in combination with the ideas proposed later in this thesis, without necessarily using the exact expression for σ_i shown in (6). σ_i needs to only be positive semi-definite and equal to 0 if and only if $\tilde{J}_i = 0$. For example, one may use $\sigma_i = \gamma(\tilde{J}_i)$ where γ is a class- K function.

3.3 Definitions of Relationships Between Value Functions

In this work, we opt to use the pointwise, min-norm controller in (5) to execute our learned tasks encoded as value functions. As a result, being able to execute N tasks concurrently then depends on the relationships between $L_g \tilde{J}_1(x), \dots, L_g \tilde{J}_N(x)$. For example, if each of $L_g \tilde{J}_1(x)^\top, \dots, L_g \tilde{J}_N(x)^\top$ were linearly independent at some state $x \in \mathcal{X}$, then it would be

possible to pick an input $u \in \mathcal{U}$ at that state that drives each $\tilde{J}_1, \dots, \tilde{J}_N$ toward 0 and therefore makes progress on each task, assuming that there are no additional constraints on the input. This motivates the definitions of independence, orthogonality, concurrent executability and concurrent controllability which we define in this section.

3.3.1 Independence and Orthogonality

Work in [30] and [3] define notions of independence and orthogonality for Extended Set-Baese and Jacobian-based tasks respectively. We define similar notions of independence and orthogonality for learned value functions based on our use of the pointwise min-norm controller presented in (5).

Definition 1 (Independent Value Functions). Consider a set of learned value functions $\tilde{J}_1, \dots, \tilde{J}_N$ of the form in (3). For each $x \in \mathcal{X}$ and $i \in \{1, \dots, N\}$, we assume that $L_g \tilde{J}_i(x) = 0$ if and only if $\tilde{J}_i(x) = 0$. $\tilde{J}_1, \dots, \tilde{J}_N$ are *independent* to each other at state $x \in \mathcal{X}$ if the nonzero vectors within $L_g \tilde{J}_1(x)^\top, \dots, L_g \tilde{J}_N(x)^\top$ are linearly independent.

Remark 1. Assuming that $L_g \tilde{J}_i(x) = 0$ if and only if $\tilde{J}_i(x) = 0$ is equivalent to assuming that $\tilde{J}_i$ has a relative degree of 1 wherever $\tilde{J}_i(x) > 0$.

Definition 2 (Orthogonal Value Functions). Consider a set of learned value functions $\tilde{J}_1, \dots, \tilde{J}_N$ of the form in (3). $\tilde{J}_1, \dots, \tilde{J}_N$ are *orthogonal* to each other at state x if $\langle (L_g \tilde{J}_i(x))^\top, (L_g \tilde{J}_j(x))^\top \rangle = 0 \forall i, j \in \{1, \dots, N\}$ where $i \neq j$.

Remark 2. Note that if value functions are orthogonal at x , then they are also independent at x .

If a set of learned value functions $\tilde{J}_1, \dots, \tilde{J}_N$ are orthogonal at a state $x \in \mathcal{X}$, it is possible to make progress on each task without having any effect on the execution of other tasks. If they are at least independent, then it is possible to at least make progress at each task.

3.3.2 Concurrent Executability and Controllability

Value functions $\tilde{J}_1, \dots, \tilde{J}_N$ do not need to be independent at every state $x \in \mathcal{X}$ to be possible to concurrently execute together. For example, if for two value functions $\tilde{J}_i, \tilde{J}_j \in \{\tilde{J}_1, \dots, \tilde{J}_N\}$, $L_g \tilde{J}_i$ and $L_g \tilde{J}_j$ were colinear, but pointed in the same direction, then it would

still be possible to pick an input that made progress at both tasks. Even in the case where $L_g \tilde{J}_i(x)$ and $L_g \tilde{J}_j(x)$ pointed in opposite directions, the values of the “drift” terms $L_f \tilde{J}_i(x)$ and $L_f \tilde{J}_j(x)$ may still make it possible to make progress at both tasks. This motivates the formal definitions of *concurrent executability* and *concurrent controllability* in this section, which not only define these ideas for pairs of value functions, but can be generalized to a set of value functions $\tilde{J}_1, \dots, \tilde{J}_N$. We later refer to these definitions in Section 4.2 in Chapter 4 where we provide necessary and sufficient conditions for a set of value functions to be concurrently executable or controllable within subsets of the state space.

Definition 3. Consider a set of learned value functions $\tilde{J}_1, \dots, \tilde{J}_N$ of the form in (3). $\tilde{J}_1, \dots, \tilde{J}_N$ are *concurrently executable* at $x \in \mathcal{X}$ if $\exists u \in \mathcal{U}$ such that $L_f \tilde{J}_i(x) + L_g \tilde{J}_i(x)u < 0$ for all $i \in \{1, \dots, N\}$ where $\tilde{J}_i(x) > 0$.

Definition 4. Consider a set of learned value functions $\tilde{J}_1, \dots, \tilde{J}_N$ of the form in (3). For each $x \in \mathcal{X}$ and $i \in \{1, \dots, N\}$, we assume that $L_g \tilde{J}_i(x) = 0$ if and only if $\tilde{J}_i(x) = 0$. $\tilde{J}_1, \dots, \tilde{J}_N$ are *concurrently controllable* at $x \in \mathcal{X}$ if there is no $P, Q \subseteq \{1, \dots, N\}$ where $\sum_{i \in P} L_g \tilde{J}_i(x) = -k \sum_{i \in Q} L_g \tilde{J}_i(x)$ and $\sum_{i \in P} L_g \tilde{J}_i(x) \neq 0$ for some $k > 0$.

Remark 3. If a set of learned value functions $\tilde{J}_1, \dots, \tilde{J}_N$ are *concurrently controllable*, then they are *concurrently executable*.

Remark 4. If a set of learned value functions $\tilde{J}_1, \dots, \tilde{J}_N$ are *independent*, then they are *concurrently controllable* and thus also *concurrently executable*.

3.4 Continuous Fitted Value Iteration

As mentioned previously, policy gradient methods [37], [20] and often a subclass of them, actor-critic methods [13], are commonly used in RL for learning to execute continuous control tasks. Both these classes of methods can be used to approximate value functions of the form in (2) and (3). For actor-critic algorithms, the value functions can simply be extracted if the state-based value function is used as the critic as opposed to a Q-value function, or value function based on a state-action pair. For methods that only learn a policy function, a value function can be estimated by evaluating the learned policy [40].

When one has a known model of their system and a discrete and finite input space, value iteration, a form of dynamic programming within the RL paradigm can be used. In the case where the system is not stochastic and has a discrete and finite state space, the

value function, represented as a table of values, can be exactly learned using the following iteration step based on Bellman’s equation:

$$J_{k+1} = \min_{u_1, \dots, u_l \in \mathcal{U}} \sum_{i=0}^{l-1} \gamma^i \cdot c(x_i, u_i) + \gamma^l J_k(x_l) \quad (7)$$

with the cost function $c(x, u) : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}_{\geq 0}$, a sequence of states $x_1, \dots, x_l \in \mathcal{X}$ reached by selecting the sequence of inputs $u_1, \dots, u_n \in \mathcal{U}$, discount factor $0 < \gamma \leq 1$ and number of lookahead steps $l \geq 1$. Value iteration in the case of (7) is guaranteed to converge given enough iterations [5].

In the case where the state space is continuous, it is no longer possible to store the exact values of the value function in a table. Instead, value iteration can be approximated with a function approximator such as a neural network using a similar iteration step:

$$\begin{aligned} J_{\text{tar}}^k(x_0) &= \min_{u_1, \dots, u_l \in \mathcal{U}} \sum_{i=0}^{l-1} \gamma^i \cdot c(x_i, u_i) + \gamma^l J(x_l; \theta_k) \quad \forall x_0 \in \mathcal{D} \\ \theta_{k+1} &= \min_{\theta} \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \|J(x; \theta) - J_{\text{tar}}^k(x)\|^2 \end{aligned} \quad (8)$$

with neural network parameters $\theta \in \mathbb{R}^d$, dataset of states $\mathcal{D}$ and everything else being the same as the equation in (7).

In the case where the input space is continuous, it is not clear how to pick the optimal inputs $u_1, \dots, u_l$ in (7) or (8). There is no way to do so in the general case and as a result, value iteration is not typically done with systems that have a continuous input space. However, Continuous Fitted Value Iteration (CFVI) proposed in [21], extends value iteration to systems with a continuous input space under certain assumptions. The CFVI method in [21], like in this thesis, assumes that the system is control-affine. It also assumes that the cost function is separable with respect to the state and input of the system and that the input cost is strictly convex. Therefore, the value functions of the form in (2) and (3) can be estimated using the CFVI method in the case where $R(x) = M$, where M is some positive definite matrix. In this work, when we use CFVI, we will let $R(x)$ equal I , meaning that the instantaneous input cost will simply equal $\|u\|^2$. The offline variant of CFVI is described in Algorithm 1. The main idea behind CFVI is to pick inputs in (8) based on a solution to the Hamilton-Jacobian Bellman equation. For cost functionals of the form in (2) and (3), this would be:

$$u^* = -\frac{1}{2}g(x)^\top \nabla_x \tilde{J}(x, \theta) = -\frac{1}{2}L_g \tilde{J}(x) \quad (9)$$

By picking a small enough value of Δt , we effectively perform Euler integrations to simulate the dynamics and attempt to more accurately estimate the value function. The discount factor γ is the discrete version of β in (2). The calculation with λ is the forward view of TD(λ) [40] used to average between several n -step value function estimates. This allows us to calculate less biased estimates of the value function.

3.4.1 Implementation Details of Continuous Fitted Value Iteration

Some implementation details regarding continuous fitted value iteration are not included in Algorithm 1 in order to more clearly and concisely explain the algorithm. The two for loops, for example, can be combined in the same loop for efficiency and this is what we do in practice. Instead of iterating through each state in the dataset $\mathcal{D}$, we instead iterate through mini-batches of states in order to effectively leverage GPUs. Furthermore, $\mathcal{D}$ does not have to be a static dataset and can be resampled each iteration from a suitable probability distribution of the state space. Each of the steps relating to the neural network are done in batches, including calculating the gradients on line 13, calculating the outputs of the network on line 17 and updating the parameters of the network on line 33. The calculation of the gradients is done using backpropagation while the update of the parameters is done using backpropagation and gradient descent, as is typically done for supervised learning tasks using neural networks. We also, in practice, implement $q(x)$, $f(x)$ and $g(x)$ to be able to take batches of states as input.

3.4.2 Use of Continuous Fitted Value Iteration to Estimate Different Kinds of Value Functions

As mentioned previously, one of the assumptions of the work on CFVI [21] is that the instantaneous cost is separable with respect to the state and the input. Later in this work, we propose an input cost, in order to train value functions to be independent as defined in 3.3, which does not satisfy this assumption, but still results in a cost functional of the form in (3). As a result, we are required to develop a variant of the CFVI algorithm in order to train value functions in a similar way.

Algorithm 1 Offline Continuous Fitted Value Iteration

Input:

- 1: state cost function $q : \mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$; dynamics functions $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ and $g : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times m}$;
- 2: # of iterations $K \geq 1$; # of lookahead steps $l \geq 1$; time step $\Delta t > 0$;
- 3: discount factor $\gamma \in (0, 1]$; $\lambda \in (0, 1]$ for averaging n -step returns;
- 4: initial parameters $\theta_0 \in \mathbb{R}^d$ of neural network; neural network $\tilde{J}$ parameterized by θ_0 ;
- 5: dataset of states $\mathcal{D} \subset \mathcal{X}$;

Output:

- 6: Value function approximator $\tilde{J}$ implemented with neural network parameterized by θ_K
 - 7:
 - 8: **function** GETNSTEPESTIMATE(x_0, n, θ)
 - 9: $r := 0$
 - 10: $x := x_0$
 - 11: **for** $i \in \{1, \dots, n\}$ **do**
 - 12: $r := r + \gamma^{i-1} \cdot q(x) \cdot \Delta t$
 - 13: $u := -\frac{1}{2}g(x)^\top \nabla_x \tilde{J}(x; \theta)$
 - 14: $r := r + \gamma^{i-1} \cdot \|u\|^2 \cdot \Delta t$
 - 15: $x := x + (f(x) + g(x)u) \Delta t$
 - 16: **end for**
 - 17: $r := r + \gamma^n \cdot \tilde{J}(x)$
 - 18: **return** r
 - 19: **end function**
 - 20: **for** $k \in \{1, \dots, K\}$ **do**
 - 21: $J_{\text{tar}} \rightarrow$ Initialize array to store target value function estimates for each $x_0 \in \mathcal{D}$.
 - 22: **for** $x_0 \in \mathcal{D}$ **do**
 - 23: $J_{\text{tar}}[x_0] := 0$
 - 24: **for** $n \in \{1, \dots, l\}$ **do**
 - 25: $r := \text{GETNSTEPESTIMATE}(x_0, n, \theta_{k-1})$
 - 26: **if** $n = l$ **then**
 - 27: $J_{\text{tar}}[x_0] := J_{\text{tar}}[x_0] + \lambda^{n-1} \cdot r$
 - 28: **else**
 - 29: $J_{\text{tar}}[x_0] := J_{\text{tar}}[x_0] + (1 - \lambda) \cdot \lambda^{n-1} \cdot r$
 - 30: **end if**
 - 31: **end for**
 - 32: **end for**
 - 33: $\theta_k := \min_{\theta \in \mathbb{R}^d} \frac{1}{|\mathcal{D}|} \sum_{x_0 \in \mathcal{D}} \|\tilde{J}(x_0; \theta) - J_{\text{tar}}[x_0]\|^2$
 - 34: **end for**
-

3.5 Summary of Problems and Proposed Solution

In this chapter, we more formally defined the problems which we aim to solve. We specified the scope of robotic systems which we aim to train tasks for using RL, as well as the optimal control problem of the form in (3) which each of the learned tasks we consider can be represented by. The pointwise, min-norm optimization-based controller described in (5) and initially proposed in [26] gives us a way to execute a set of learned tasks encoded as value functions $\tilde{J}_1, \dots, \tilde{J}_N$. However, the ability to execute each of these learned tasks depends on the relationships between the Lie derivatives $L_g \tilde{J}_1, \dots, L_g \tilde{J}_N$. This prompts us to formally define notions of *independence*, *orthogonality*, *concurrent executability* and *concurrent controllability*. We also describe the idea of fitted value iteration with continuous input spaces, initially proposed in [21]. However, the CFVI method does not, in the general case, approximate optimal control problems of the form in (3), meaning that we may have to develop a variant in order to do so.

In the next chapter, we propose solutions and theoretical results to address these problems. In 4.1.1, we propose an input cost that we prove can be used to train a set of value functions to be independent. The resulting value function cannot be learned using CFVI, leading us to develop a variant of CFVI which we describe in 4.1.2. In Section 4.2, we propose theorems that provide both necessary and sufficient conditions for a set of learned value functions to be concurrently executable within subsets of the state space. In Chapter 5, we demonstrate our theoretical ideas in practice on simulations and experiments of one or more mobile robots concurrently performing multiple tasks.

Chapter 4

Main Results

In this chapter, we present the main theoretical results of this thesis. In Section 4.1, we present ideas related to training value functions to satisfy the notion of independence defined in the previous chapter. Cost functionals with an “interference cost” are proposed, which we prove, using Proposition 1, can be used to train a sequence of value functions to be independent to one another. We also present Proposition 2 which shows the relationship between orthogonality and the optimal input for such cost functionals. We also propose a variant of continuous fitted value iteration that can be used to approximate the proposed cost functional. In Section 4.2, we present ideas related to the necessary and sufficient conditions for a set of value functions to be concurrently executable. Propositions 4 and 5 present necessary conditions for a set of value functions to be concurrently executable within a compact subset of the state space. Proposition 6 presents sufficient conditions for a set of value functions to be concurrently executable within at least one subset of the state space. While Section 4.1 presents ideas related to training value functions to be independent and therefore concurrently executable, Section 4.2 provides insight into when control tasks are possible to be made concurrently executable, when they might already inherently be concurrently executable and when it is not possible at all to make a set of tasks concurrently executable. The results presented in this chapter are later demonstrated in the next chapter on simulations involving mobile robots. Each of the results in this chapter and their significance are summarized in Section 4.3.

4.1 Learning Independent Value Functions

In this section, we present ideas that can be used to train value functions using RL to be independent, and thus as mentioned in Remarks 2 and 3, concurrently executable. The main idea of Section 4.1 is to present a cost functional that is of the form in (3) where $R_i(x)$ encodes a cost on “interfering” with the execution of previously trained, higher-priority tasks. This newly proposed cost functional can be approximated, in theory, using any deep RL algorithm. The CFVI algorithm presented in the previous chapter would be useful as we also mentioned in the previous chapter that our systems have deterministic, control-affine models. However, the state and input cost are not separable in this newly proposed cost functional, leading us to present a variant of CFVI in Section 4.2.

4.1.1 Interference Cost for Independent Value Functions

Consider a situation where we have already learned a sequence of $N - 1$ value functions of the form in (3), $\tilde{J}_1, \dots, \tilde{J}_{N-1}$, and we wished to train an N th value function, $\tilde{J}_N$, to be independent to these $N - 1$ previously trained value functions. To do so, we propose training a value function to fit the following cost functional:

$$\min_{u(\cdot)} \int_t^\infty \left(q_N(x) + \|u\|^2 + \sum_{i=1}^{N-1} (L_g \tilde{J}_i(x) u)^2 \alpha_i \right) d\tau \quad (10)$$

We can see that (10) is an instance of (3), as $\|u\|^2 + \sum_{i=1}^{N-1} (L_g \tilde{J}_i(x) u)^2 \alpha_i$ can be rearranged as:

$$\begin{aligned} \|u\|^2 + \sum_{i=1}^{N-1} (L_g \tilde{J}_i(x) u)^2 \alpha_i &= \|u\|^2 + \sum_{i=1}^{N-1} \left(u^\top L_g \tilde{J}_i(x) \right)^\top \left(L_g \tilde{J}_i(x) u \right) \alpha_i \\ &= u^\top u + u^\top \left(\sum_{i=1}^{N-1} \left(L_g \tilde{J}_i(x) \right)^\top L_g \tilde{J}_i(x) \alpha_i \right) u \\ &= u^\top \left(I + \sum_{i=1}^{N-1} \left(L_g \tilde{J}_i(x) \right)^\top (L_g \tilde{J}_i(x)) \alpha_i \right) u \end{aligned} \quad (11)$$

We can also see that $I + \sum_{i=1}^{N-1} (L_g \tilde{J}_i(x))^\top (L_g \tilde{J}_i(x)) \alpha_i$ is positive definite since:

$$I \succ 0 \quad (12)$$

$$\begin{aligned} \sum_{i=1}^{N-1} (L_g \tilde{J}_i(x))^\top (L_g \tilde{J}_i(x)) \alpha_i &\succeq 0 \quad \forall x \in \mathcal{X} && \text{since it is symmetric} \\ I + \sum_{i=1}^{N-1} (L_g \tilde{J}_i(x))^\top (L_g \tilde{J}_i(x)) \alpha_i &\succ 0 \quad \forall x \in \mathcal{X} \end{aligned}$$

$\alpha_1, \dots, \alpha_{N-1}$ are hyperparameters that determine how much to penalize inputs which interfere with the execution of previously trained, higher priority tasks. Through Proposition 1, we show that choosing large enough $\alpha_1, \dots, \alpha_{N-1}$ would result in a newly trained value function, $\tilde{J}_N$ that fits our proposed cost functional, to be independent to $\tilde{J}_1, \dots, \tilde{J}_{N-1}$.

Proposition 1. *There exist scalars $\alpha_1, \dots, \alpha_{N-1} > 0$, such that if $\tilde{J}_N$ fits a cost functional of the form in (10) that uses $\alpha_1, \dots, \alpha_{N-1}$, then for every state $x \in \mathcal{X}$ where $\tilde{J}_1, \dots, \tilde{J}_{N-1}$ are independent, $\tilde{J}_1, \dots, \tilde{J}_{N-1}, \tilde{J}_N$ are also independent at x .*

Proof. If our deep RL algorithm for training the function $\tilde{J}_N$ converges such that it solves the HJB equation for the cost functional in (6), then the optimal input, based on referencing equation (7) in [7], minimizes the following unconstrained optimization problem:

$$\begin{aligned} &\underset{u \in \mathcal{U}}{\operatorname{argmin}} \left\{ q_N(x) + \|u\|^2 + \sum_{i=1}^N (L_g \tilde{J}_i(x) u)^2 \alpha_i + L_f \tilde{J}_N(x) + L_g \tilde{J}_N(x) u \right\} \\ &= \underset{u \in \mathcal{U}}{\operatorname{argmin}} \left\{ \|u\|^2 + \sum_{i=1}^N (L_g \tilde{J}_i(x) u)^2 \alpha_i + L_g \tilde{J}_N(x) u \right\} \end{aligned} \quad (13)$$

We can find the solution to this by calculating the gradient with respect to u . Let $R(x) =$

$I + \sum_{i=1}^N (L_g \tilde{J}_i(x))^\top L_g \tilde{J}_i(x)$. We can see that:

$$\begin{aligned}
\nabla_u \left(\|u\|^2 + \sum_{i=1}^N (L_g \tilde{J}_i(x)u)^2 \alpha_i + L_g \tilde{J}_N(x)u \right) &= 0 \\
\nabla_u \left(u^\top \left(I + \sum_{i=1}^{N-1} (L_g \tilde{J}_i(x))^\top (L_g \tilde{J}_i(x)) \alpha_i \right) u + L_g \tilde{J}_N(x)u \right) &= 0 \\
\nabla_u \left(u^\top R(x)u + L_g \tilde{J}_N(x)u \right) &= 0 \\
2R(x)u + L_g \tilde{J}_N(x) &= 0 \\
2R(x)u &= -L_g \tilde{J}_N(x) \\
R(x)u &= -\frac{1}{2}L_g \tilde{J}_N(x) \\
u &= -\frac{1}{2}R(x)^{-1}L_g \tilde{J}_N(x)
\end{aligned} \tag{14}$$

Therefore, the solution to optimization problem in (13) is $u^\star = -\frac{1}{2}R(x)^{-1}L_g \tilde{J}_N(x)$ where $R(x) = I + \sum_{i=1}^N (L_g \tilde{J}_i(x))^\top L_g \tilde{J}_i(x)$. Note that $R(x)^{-1}$ is defined since $R(x)$ is positive definite. Later, in Proposition 3 in 4.1.2, we will derive why the optimal input solves the optimization problem in (13).

If $\alpha_1, \dots, \alpha_{N-1}$ are large enough, $u^\star$ approximates a solution to the following constrained optimization problem:

$$\begin{aligned}
\min_u \left\{ \|u\|^2 + L_g \tilde{J}_N(x)u \right\} \\
\text{s.t. } \sum_{i=1}^N (L_g \tilde{J}_i(x)u)^2 &= 0
\end{aligned} \tag{15}$$

which means that $\langle L_g \tilde{J}_i(x), -\frac{1}{2}R(x)^{-1}(L_g \tilde{J}_N(x))^\top \rangle = 0$ for all $i \in \{1, \dots, N\}$. For the sake of a contradiction, assume that for some $x \in \mathcal{X}$, $\tilde{J}_1, \dots, \tilde{J}_{N-1}$ are independent at x and $\tilde{J}_1, \dots, \tilde{J}_{N-1}, \tilde{J}_N$ are not independent at x meaning that $L_g \tilde{J}_N(x)$ can be written as a linear combination of $L_g \tilde{J}_1(x), \dots, L_g \tilde{J}_{N-1}(x)$. Let $c_1, \dots, c_{N-1} \in \mathbb{R}$ be the multipliers for this

linear combination. We can see that:

$$\begin{aligned}
\langle c_i(L_g \tilde{J}_i(x))^\top, -\frac{1}{2}R(x)^{-1}(L_g \tilde{J}_N(x))^\top \rangle &= 0 \forall i \in \{1, \dots, N-1\} \quad (16) \\
\sum_{i=1}^{N-1} \langle c_i(L_g \tilde{J}_i(x))^\top, -\frac{1}{2}R(x)^{-1}(L_g \tilde{J}_N(x))^\top \rangle &= 0 \\
\langle \sum_{i=1}^{N-1} c_i(L_g \tilde{J}_i(x))^\top, -\frac{1}{2}R(x)^{-1}(L_g \tilde{J}_N(x))^\top \rangle &= 0 \\
\langle (L_g \tilde{J}_N(x))^\top, -\frac{1}{2}R(x)^{-1}(L_g \tilde{J}_N(x))^\top \rangle &= 0 \\
\langle (L_g \tilde{J}_N(x))^\top, R(x)^{-1}(L_g \tilde{J}_N(x))^\top \rangle &= 0
\end{aligned}$$

$R(x)$ is positive definite, so its inverse must also be positive definite. This means that $(L_g \tilde{J}_N(x))^\top$ must be 0. However, we assumed $\tilde{J}_1, \dots, \tilde{J}_{N-1}$ to be independent, so a linear combination of $(L_g \tilde{J}_1(x))^\top, \dots, (L_g \tilde{J}_{N-1}(x))^\top$ cannot be 0 unless $(L_g \tilde{J}_1(x))^\top, \dots, (L_g \tilde{J}_{N-1}(x))^\top$ are all 0. In the latter case, $\tilde{J}_1, \dots, \tilde{J}_{N-1}, \tilde{J}_N$ would be independent under the definition in 3.1.1. However, in the case where at least one of $(L_g \tilde{J}_1(x))^\top, \dots, (L_g \tilde{J}_{N-1}(x))^\top$ is non-zero, it is a contradiction. Therefore, if $\tilde{J}_1, \dots, \tilde{J}_{N-1}$ are independent at x , then $\tilde{J}_1, \dots, \tilde{J}_{N-1}, \tilde{J}_N$ are also independent at x . $\square$

Proposition 1 suggests that we can train a sequence of N tasks to be independent by training them in order using the cost functional in (10) with large enough choices of $\alpha_1, \dots, \alpha_N$. For example, one can first train $\tilde{J}_1$, then train $\tilde{J}_2$ to be independent to $\tilde{J}_1$ with a large enough value of α_1 , then train $\tilde{J}_3$ to be independent to $\tilde{J}_1$ and $\tilde{J}_2$ with large enough values of α_1, α_2 and so on until we have N learned value functions that are all independent to one another. We can use numerous method within RL to try to approximate the cost functional in (10), such as versions of fitted value iteration or actor-critic algorithms that learn a value function in addition to a policy. We will describe a particular variant of fitted value iteration to learn these value functions in 4.1.2.

In some cases, it is possible for a resulting learned task $\tilde{J}_N$ to not only be independent to previously trained tasks $\tilde{J}_1, \dots, \tilde{J}_{N-1}$, but to also be orthogonal. We show in Proposition 2 that if $\tilde{J}_1, \dots, \tilde{J}_{N-1}$ are independent, then the optimal policy to execute task $\tilde{J}_N$ is $-\frac{1}{2}(L_g \tilde{J}_N(x))^\top$ if and only if $\tilde{J}_N$ is orthogonal to each of $\tilde{J}_1, \dots, \tilde{J}_{N-1}$.

Proposition 2. *Assume that a function $\tilde{J}_N$ solves the HJB equation and is defined based on the cost functional in (10). Without loss of generality, let $\alpha_1 = \dots = \alpha_{N-1} = 1$. At a state $x \in \mathcal{X}$, if $\tilde{J}_1, \dots, \tilde{J}_{N-1}$ in the cost functional of $\tilde{J}_N$ are independent, then $\tilde{J}_N$ is*

orthogonal to each of $J_i \in \{J_1, \dots, J_{N-1}\}$ at x if and only if the optimal input at x with respect to $\tilde{J}_N$ is equal to $-\frac{1}{2}(L_g \tilde{J}_N(x))^\top$.

Proof. ($\Rightarrow$) Assume that $\tilde{J}_N$ is orthogonal to each of $\tilde{J}_1, \dots, \tilde{J}_{N-1}$ at $x \in \mathcal{X}$. As shown in the work in [7] and used similarly in Proposition 1, the optimal input with respect to $\tilde{J}_N$ is an input that minimizes the following optimization problem:

$$\min_{u \in \mathcal{U}} \left\{ \|u\|^2 + \sum_{i=1}^{N-1} (L_g \tilde{J}_i(x)u)^2 + L_g \tilde{J}_N(x)u \right\} \quad (17)$$

We can see that if $\tilde{J}_N$ is orthogonal to each of $\tilde{J}_1, \dots, \tilde{J}_{N-1}$ at x , then we can also see that:

$$\begin{aligned} \sum_{i=1}^{N-1} \langle L_g \tilde{J}_i(x)^\top, L_g \tilde{J}_N(x)^\top \rangle &= 0 \\ \sum_{i=1}^{N-1} L_g \tilde{J}_i(x) \left(L_g \tilde{J}_N(x) \right)^\top &= 0 \\ \sum_{i=1}^{N-1} \left(L_g \tilde{J}_i(x) \left(L_g \tilde{J}_N(x) \right)^\top \right)^2 &= 0 \\ \sum_{i=1}^{N-1} \left(L_g \tilde{J}_i(x) \left(-\frac{1}{2} L_g \tilde{J}_N(x) \right)^\top \right)^2 &= 0 \end{aligned} \quad (18)$$

We also know that $-\frac{1}{2}(L_g \tilde{J}_N(x))^\top$ minimizes the function $\|u\|^2 + L_g \tilde{J}_N(x)u$. Therefore, if $\tilde{J}_N$ is orthogonal to each of $\tilde{J}_1, \dots, \tilde{J}_{N-1}$ at x , then the optimal input at x is $-\frac{1}{2}(L_g \tilde{J}_N(x))^\top$.

($\Leftarrow$) Assume that $u^* = -\frac{1}{2}(L_g \tilde{J}_N(x))^\top$ is the optimal input. As mentioned previously, we know that the optimal input must solve the following optimization problem:

$$\min_{u \in \mathcal{U}} \left\{ \|u\|^2 + \sum_{i=1}^{N-1} (L_g \tilde{J}_i(x)u)^2 + L_g \tilde{J}_N(x)u \right\} \quad (19)$$

which means it must also solve the following equation when we take the gradient and set it equal to 0:

$$2u + 2g(x)^\top \left(\sum_{i=1}^{N-1} \left(\frac{\partial \tilde{J}_i}{\partial x} \right)^\top \frac{\partial \tilde{J}_i}{\partial x} \right) g(x)u + (L_g \tilde{J}_N(x))^\top = 0 \quad (20)$$

Substituting in $u^* = -\frac{1}{2}(L_g \tilde{J}_N(x))^\top$, we see that:

$$\begin{aligned}
-L_g \tilde{J}_N(x)^\top - g(x)^\top \left(\sum_{i=1}^{N-1} \left(\frac{\partial \tilde{J}_i}{\partial x} \right)^\top \frac{\partial \tilde{J}_i}{\partial x} \right) g(x) (L_g \tilde{J}_N(x))^\top + L_g \tilde{J}_N(x)^\top &= 0 \quad (21) \\
-g(x)^\top \left(\sum_{i=1}^{N-1} \left(\frac{\partial \tilde{J}_i}{\partial x} \right)^\top \frac{\partial \tilde{J}_i}{\partial x} \right) g(x) (L_g \tilde{J}_N(x))^\top &= 0 \\
\sum_{i=1}^{N-1} \left(g(x)^\top \frac{\partial \tilde{J}_i}{\partial x} \right)^\top \left(\frac{\partial \tilde{J}_i}{\partial x} g(x) \right) (L_g \tilde{J}_N(x))^\top &= 0 \\
\sum_{i=1}^{N-1} (L_g \tilde{J}_i(x))^\top \langle (L_g \tilde{J}_i(x))^\top, L_g \tilde{J}_N(x)^\top \rangle &= 0
\end{aligned}$$

We assumed that each of $\tilde{J}_1, \dots, \tilde{J}_{N-1}$ are independent at x . It is possible that $L_g \tilde{J}_1(x), \dots, L_g \tilde{J}_{N-1}(x)$ are all zero, but in that case $\tilde{J}$ would be orthogonal to each of $\tilde{J}_1, \dots, \tilde{J}_{N-1}$ at x . If at least one of $L_g \tilde{J}_1(x), \dots, L_g \tilde{J}_{N-1}(x)$ are non-zero, then we cannot find a set of non-zero constants that make this sum to 0, so it must be the case that $\langle (L_g \tilde{J}_i(x))^\top, (L_g \tilde{J}_N(x))^\top \rangle = 0$ for each $i \in \{1, \dots, N-1\}$.

□

If a task, $\tilde{J}_N$, can be trained such that it is orthogonal to a set of other independent tasks, $\tilde{J}_1, \dots, \tilde{J}_{N-1}$, then not only is it possible to execute each task concurrently, it is also possible to execute the optimal input for $\tilde{J}_N$ without having to calculate the gradients for tasks that $\tilde{J}_N$ was trained to be independent to.

In this subsection, we proposed the cost functional in (10) that can be used to train an N th value function to be independent to $N-1$ value functions that have already been trained. We show in Proposition 1 that we can make this N th value function independent to the previously trained $N-1$ value functions by picking large enough values of $\alpha_1, \dots, \alpha_{N-1}$. In Proposition 2, we show the relationship between orthogonality and the optimal input for a value function of the form in (10). In the next subsection, we will propose a variant of fitted value iteration for fitting cost functionals of the form in (10).

4.1.2 Variant of Continuous Fitted Value Iteration

We propose a form of fitted value iteration for learning cost-to-go functions of the form in (10). More generally, we will propose a form of fitted value iteration for learning cost-to-go

functions of the form in (2) and (3), which generalizes to the case in (10) as well as the corresponding discounted version.

The main idea of the CFVI [21] algorithm is to extend the fitted value iteration algorithm by solving the optimization problem in (7) analytically for the continuous control setting. We extend this approach in Proposition 3 for cost functionals of the form in (2). We used the result of this proposition in Propositions 1 and 2 by referring to equation (7) in [7]. However, we will provide more explicit details in the following proof.

Proposition 3. *Given a control-affine system and the k -th iteration of an estimate of the cost-to-go function described in (10), J^k , the optimal policy based on the fitted value iteration update step in the continuous setting is:*

$$u^*(t) = -\frac{1}{2}R(x)^{-1} (L_g J^k(x))^\top \quad (22)$$

Proof. This proof is very similar to Theorem 1 in [21]. We first define discretized dynamics for the system by using a Δt that is small enough to approximate the dynamics in (1). We define $\bar{f}(x, u)$ such that:

$$\bar{f}(x, u) = x + (f(x) + g(x)u)\Delta t \quad (23)$$

where $\bar{f}(x, u)$ approximates the new state after sending input $u \in \mathcal{U}$ at state $x \in \mathcal{X}$ for a duration of Δt . We then substitute $\bar{f}(x, u)$ and the instantaneous cost into the fitted value iteration update step in (8) with $l = 1$:

$$\begin{aligned} J_{\text{tar}}^k(x) &= \min_{u \in \mathcal{U}} \{c(x, u) + \gamma J(\bar{f}(x, u); \theta_k)\} \\ J_{\text{tar}}^k(x) &= \min_{u \in \mathcal{U}} \{(q(x) + u^\top R(x)u) \cdot \Delta t + \gamma J^k(\bar{f}(x, u))\}. \end{aligned} \quad (24)$$

We can then rewrite this using the Taylor approximation of J^k around x :

$$J_{\text{tar}}^k(x) = \min_{u \in \mathcal{U}} \left\{ (q(x) + u^\top R(x)u) \Delta t + \gamma \left(J^k(x) + \left(\frac{\partial J^k}{\partial x} \right) \bar{f}(x, u) + \mathcal{O}(\Delta t, x, u) \right) \right\} \quad (25)$$

where $\mathcal{O}(\Delta t, x, u)$ represents the higher order terms. In the continuous setting, Δt approaches 0, which means that the higher order terms, $\mathcal{O}(\Delta t, x, u)$, also approach 0. By the same logic in [21], γ also approaches 1 at the continuous time limit as Δt approaches 0.

To find the optimal input, we can then simplify, removing terms that do not rely on u , removing $\mathcal{O}(\Delta t, x, u)$ and substituting $\gamma = 1$ to get the following unconstrained optimization problem:

$$\begin{aligned}
& \min_{u \in \mathcal{U}} \left\{ (q(x) + u^\top R(x)u) \Delta t + \gamma \left(J^k(x) + \left(\frac{\partial J^k}{\partial x} \right) \bar{f}(x, u) + \mathcal{O}(\Delta t, x, u) \right) \right\} \quad (26) \\
&= \min_{u \in \mathcal{U}} \left\{ (q(x) + u^\top R(x)u) \Delta t + \gamma \left(J^k(x) + \left(\frac{\partial J^k}{\partial x} \right) (f(x) + g(x)u) \Delta t \right) \right\} \\
&= \min_{u \in \mathcal{U}} \left\{ (u^\top R(x)u) \Delta t + \gamma \frac{\partial J^k}{\partial x} g(x)u \Delta t \right\} \\
&= \min_{u \in \mathcal{U}} \left\{ u^\top R(x)u + \frac{\partial J^k}{\partial x} g(x)u \right\} \\
&= \min_{u \in \mathcal{U}} \{ u^\top R(x)u + L_g J^k(x)u \}
\end{aligned}$$

for which the solution is $u^* = -\frac{1}{2}R(x)^{-1} (L_g J^k(x))^\top$, as seen in (14) in Proposition 1. Note that $R(x)^{-1}$ is defined since $R(x)$ is positive definite. $\square$

We can then use this analytical solution for the optimal policy with respect to J^k to perform the fitted value iteration steps shown in (8). To avoid biased estimates for a particular choice of l , we use the forward view of TD(λ) [40].

This version of fitted value iteration is very similar to CFVI [21]. However, instead of assuming that the input cost is separable from the state and strictly convex, we instead assume that it is quadratic with respect to some function that maps the current state to a positive definite matrix. This differing assumption allows us to propose cost functionals, of a form different than those that the CFVI algorithm is able to learn, for training tasks that satisfy our definition of independence. Recall from the previous subsection that cost functionals of the form in (10) are an instance of those of the form in (3).

Our variant of CFVI is shown in Algorithm 2, with the new function that is referenced defined in Algorithm 3. It is very similar to the version of CFVI that we showed in Algorithm 1, with the main difference being the input used to calculate new value function estimates.

Algorithm 2 Continuous Fitted Value Iteration for Independent Value Functions

Input:

- 1: state cost function $q : \mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$; dynamics functions $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ and $g : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times m}$;
- 2: # of iterations $K \geq 1$; # of lookahead steps $l \geq 1$; time step $\Delta t > 0$;
- 3: discount factor $\gamma \in (0, 1]$; $\lambda \in (0, 1]$ for averaging n -step returns;
- 4: initial parameters $\theta_0 \in \mathbb{R}^d$ of neural network; neural network $\tilde{J}$ parameterized by θ_0 ;
- 5: dataset of states $\mathcal{D} \subset \mathcal{X}$; value functions to be independent to $\tilde{J}_1, \dots, \tilde{J}_N$;
- 6: constants for penalizing interference $\alpha_1, \dots, \alpha_N$

Output:

- 7: Value function approximator $\tilde{J}$ implemented with neural network parameterized by θ_K
 - 8:
 - 9: **function** GETNSTEPESTIMATE($x_0, n, \theta, g : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times m}, x, \tilde{J}_1, \dots, \tilde{J}_N, \alpha_1, \dots, \alpha_N$)
 - 10: $r := 0$
 - 11: $x := x_0$
 - 12: **for** $i \in \{1, \dots, n\}$ **do**
 - 13: $r := r + \gamma^{i-1} \cdot q(x) \cdot \Delta t$
 - 14: $R := \text{GETINPUTCOSTMATRIX}(g, x, \tilde{J}_1, \dots, \tilde{J}_N, \alpha_1, \dots, \alpha_N)$
 - 15: $u := -\frac{1}{2} R^{-1} g(x)^\top \nabla_x \tilde{J}(x; \theta)$
 - 16: $r := r + \gamma^{i-1} \cdot u^\top R u \cdot \Delta t$
 - 17: $x := x + (f(x) + g(x)u) \Delta t$
 - 18: **end for**
 - 19: $r := r + \gamma^n \cdot \tilde{J}(x)$
 - 20: **return** r
 - 21: **end function**
 - 22: **for** $k \in \{1, \dots, K\}$ **do**
 - 23: $J_{\text{tar}} \rightarrow$ Initialize array to store target value function estimates for each $x_0 \in \mathcal{D}$.
 - 24: **for** $x_0 \in \mathcal{D}$ **do**
 - 25: $J_{\text{tar}}[x_0] := 0$
 - 26: **for** $n \in \{1, \dots, l\}$ **do**
 - 27: $r := \text{GETNSTEPESTIMATE}(x_0, n, \theta_{k-1})$
 - 28: **if** $n = l$ **then**
 - 29: $J_{\text{tar}}[x_0] := J_{\text{tar}}[x_0] + \lambda^{n-1} \cdot r$
 - 30: **else**
 - 31: $J_{\text{tar}}[x_0] := J_{\text{tar}}[x_0] + (1 - \lambda) \cdot \lambda^{n-1} \cdot r$
 - 32: **end if**
 - 33: **end for**
 - 34: **end for**
 - 35: $\theta_k := \min_{\theta \in \mathbb{R}^d} \frac{1}{|\mathcal{D}|} \sum_{x_0 \in \mathcal{D}} \|\tilde{J}(x_0; \theta) - J_{\text{tar}}[x_0]\|^2$
 - 36: **end for**
-

Algorithm 3 Definition of Function Used in Algorithm 2

```
1: function GETINPUTCOSTMATRIX( $g : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times m}$ ,  $x$ ,  $\tilde{J}_1, \dots, \tilde{J}_N$ ,  $\alpha_1, \dots, \alpha_N$ )
2:    $M := 0 \in \mathbb{R}^{n \times n}$ 
3:   for  $i \in \{1, \dots, N\}$  do
4:      $M := M + \left( \nabla_x \tilde{J}_i \right) \left( \nabla_x \tilde{J}_i \right)^\top \cdot \alpha_i$ 
5:   end for
6:   return  $I + g(x)^\top M g(x)$ 
7: end function
```

4.1.3 Implementation Details for Our Variant of Continuous Fitted Value Iteration

Like in Algorithm 1, we omit some implementation details in order to more concisely and clearly explain the algorithm. Each of the implementation details presented in Section 3.4.1 for the original continuous fitted value iteration algorithm apply here as well. In addition to the details mentioned there, we also calculate the gradients for each of the value functions $\tilde{J}_1, \dots, \tilde{J}_N$ in mini-batches of states. Each of the matrix multiplications and calculation of the inverse of R are done in batches as well. These operations are supported by the mainstream frameworks for implementing neural networks such as PyTorch [31] and Tensorflow.[1]. Additional implementation details, both in general and specific to simulations presented later in Chapter 5, are mentioned in Appendix A.

4.2 Conditions for Concurrent Executability

Value functions do not have to be made independent to be concurrently executable. If the Lie derivatives of two or more value functions pointed in the same direction, then it would be possible to concurrently make progress on each task. Furthermore, it may not be possible to make some tasks concurrently executable, let alone independent. In Section 4.2.1, we present Propositions 4 and 5 that provide necessary conditions that a set of value functions must satisfy in order to be concurrently executable. In Section 4.2.2, we present Proposition 6 which provides sufficient conditions for a set of value functions to be concurrently executable in at least one subset of the state space.

4.2.1 Necessary Conditions for the Concurrent Executability of Learned Value Functions

Consider a set of value functions $\tilde{J}_1, \dots, \tilde{J}_N$ and a compact subset of the state space $A \subset \mathcal{X}$. For $\tilde{J}_1, \dots, \tilde{J}_N$ to be concurrently executable everywhere in A , it may make intuitive sense that there must exist at least one common “goal” state $x \in A$ that is an equilibrium point where $\tilde{J}_1(x) = \dots = \tilde{J}_N(x) = 0$. Furthermore, if $\tilde{J}_1, \dots, \tilde{J}_N$ were concurrently executable everywhere in A , we would also expect that we never leave the sublevel set for any particular value function $\tilde{J}_i \in \{\tilde{J}_1, \dots, \tilde{J}_N\}$ when executing each task simultaneously. We prove that these intuitions are true and formalize them in Propositions 5 and 6 using what we first prove in Lemma 1.

Lemma 1. *Let $\tilde{J}_1, \dots, \tilde{J}_N$ be a set of learned value functions as described in (3). Let $A \subset \mathcal{X}$ be a compact set. Assume that $\forall x \in A, \exists u \in \mathbb{R}^m$ such that $\dot{x} = f(x) + g(x)u \in T_A(x)$ and $L_f \tilde{J}_i(x) + L_g \tilde{J}_i(x)u \leq 0 \forall i \in \{1, \dots, N\}$ where $L_f \tilde{J}_i(x) + L_g \tilde{J}_i(x)u < 0$ when $\tilde{J}_i(x) > 0$. Then there exists some set of equilibrium states $G \subset A$ where $\forall x \in G, \tilde{J}_1(x) = \tilde{J}_2(x) = \dots = \tilde{J}_N(x) = 0$ and x is reachable from A by staying within A .*

Proof. Consider a min-norm controller of the form:

$$\begin{aligned} \min_{u \in \mathbb{R}^m} \quad & \|u\|^2 \\ \text{s.t.} \quad & L_f \tilde{J}_1(x) + L_g \tilde{J}_1(x)u \leq -\sigma_1(x) \\ & \vdots \\ & L_f \tilde{J}_N(x) + L_g \tilde{J}_N(x)u \leq -\sigma_N(x) \\ & f(x) + g(x)u \in T_A(x) \end{aligned} \tag{27}$$

The controller in (27) is the same as the one in (4) with the additional constraint that the input chosen keeps the time derivative of the state within the tangent cone to A at x . Each of these constraints can be satisfied for all $x \in A$ under our assumptions. In other words, for any $x \in A$, there is some feasible input $u \in \mathbb{R}^m$ that stays in A and also makes progress on each task $\tilde{J}_1, \dots, \tilde{J}_N$. Let $\bar{V}(x) = \sum_{i=1}^N \tilde{J}_i(x)$. Let $\bar{f}(x) = f(x) + g(x)u$ be the dynamics of this autonomous system resulting from the specified controller.

A would be positively invariant and $L_{\bar{f}} \bar{V}(x) \leq 0$ for each $x \in A$ along the resulting trajectory. Referring to [18], we can then apply Theorem 4.4 (LaSalle’s theorem). We can see that $L_{\bar{f}} \bar{V}(x) = 0$ if and only if $\tilde{J}_1(x) = \tilde{J}_2(x) = \dots = \tilde{J}_N(x) = 0$ and x is an equilibrium state. Let $G = \{x \in A : \tilde{J}_1(x) = \dots = \tilde{J}_N(x) = 0\}$. By the logic in the proof of LaSalle’s

theorem, G is non-empty. Therefore, there exists some set of equilibrium states where $\tilde{J}_1(x) = \tilde{J}_2(x) = \dots \tilde{J}_N(x) = 0$ that is reachable from A and by staying within A . $\square$

Proposition 4. *Let $\tilde{J}_1, \dots, \tilde{J}_N$ be a set of learned value functions as described in (3). For any compact set $A \subset \mathcal{X}$, if there does not exist a set of common equilibrium states $G \subseteq A$ where $\forall x \in G, \tilde{J}_1(x) = \dots = \tilde{J}_N(x) = 0$, then there exists some $x \in A$ where either it is not possible to pick an input that causes the system to stay in A or the tasks are not concurrently executable in A .*

Proof. Lemma 1 shows that if for every state in a compact set, $A \subset \mathcal{X}$, there is an input that keeps the system in A and concurrently makes progress on each task, then there exists some set of equilibrium states $G \subset A$ where $\forall x \in G, \tilde{J}_1(x) = \dots = \tilde{J}_N(x) = 0$. By the contrapositive of this, if such a set G does not exist within any compact set, A , then for any compact set, A , there is some state $x \in A$ where it is either not possible to pick an input $u \in \mathcal{U}$ that keeps the system within A or the value functions $\tilde{J}_1, \dots, \tilde{J}_N$ are not concurrently executable at x . $\square$

Proposition 5. *Let $\tilde{J}_1, \dots, \tilde{J}_N$ be a set of learned value functions as described in (3). Let $A \subset \mathcal{X}$ be a compact set of states. Let G be the common set of equilibrium states where $\forall x \in G, \tilde{J}_1(x) = \dots = \tilde{J}_N(x) = 0$. For $i \in \{1, \dots, N\}$ and $c \geq 0$, let $S_{i,c} := \{x \in A : \tilde{J}_i(x) \leq c\}$. If there exists some $i \in \{1, \dots, N\}$ and $c \geq 0$ where no state in G is reachable from $S_{i,c}$ without leaving $S_{i,c}$, then either $\tilde{J}_1, \dots, \tilde{J}_N$ are not concurrently executable at every state in A or there exists some state $x \in A$ where it is not possible to pick an input that keeps the system in A .*

Proof. We can see this by applying the contrapositive of Lemma 1 to the compact set $S_{i,c}$ and value functions $\tilde{J}_1, \dots, \tilde{J}_{i-1}, \tilde{J}_{i+1}, \tilde{J}_N$. If there does not exist a set of equilibrium states $\bar{G} \subset S_{i,c}$ where $\forall x \in \bar{G}, \tilde{J}_1(x) = \dots = \tilde{J}_{i-1}(x) = \tilde{J}_{i+1} = \dots = \tilde{J}_N = 0$ and it is possible to reach $\bar{G}$ without leaving $S_{i,c}$, then there is some $x \in S_{i,c}$ where it is either not possible to execute each of the tasks $\tilde{J}_1, \dots, \tilde{J}_{i-1}, \tilde{J}_{i+1}, \tilde{J}_N$ or it is not possible to pick an input that keeps the state in $S_{i,c}$, meaning that we either cannot make progress on the task $\tilde{J}_i$ or we exit the larger set A . In either case, this means that $\tilde{J}_1, \dots, \tilde{J}_N$ is not concurrently executable at x , or it is not possible to pick an input that keeps the system in A . $\square$

The practical implications of the propositions in this subsection are as follows. For a set to be concurrently executable, there must be a set of “goal” states common to each of the learned tasks. Furthermore, it must be possible to go toward some common “goal” state without leaving the sublevel set of any learned value function. These results are intuitive, but are formalized by the propositions in this subsection.

4.2.2 Existence of a Concurrently Executable Set of States

If there does exist at least one common “goal” equilibrium state where $\tilde{J}_1 = \dots \tilde{J}_N = 0$, it would intuitively seem that $\tilde{J}_1, \dots \tilde{J}_N$ should at least be concurrently executable when close enough to such a “goal” state. We prove in Proposition 6 that this intuition is also true.

Proposition 6. *Let $\tilde{J}_1, \dots, \tilde{J}_N$ be a set of learned value functions of the form in (3) that are all smooth and not identically equal to 0. Assume there exists some non-empty set $G \subset \mathcal{X}$ where $\forall x \in G$, x is an equilibrium point and $\tilde{J}_1(x) = \dots = \tilde{J}_N(x) = 0$. Referring to the system dynamics in (1), assume that $g(x)$ has full row rank. Then there exists some set F , where $G \subset F \subseteq \mathcal{X}$, and for every $x \in F$, $\tilde{J}_1, \dots, \tilde{J}_N$ is concurrently controllable, and therefore also concurrently executable, at x .*

Proof. We can see that if $g(x)$ has full row rank, then $\tilde{J}_1, \dots, \tilde{J}_N$ being concurrently controllable or not relies solely on the gradients of each value function. We can see that if there exists some $P, Q \subset \{1, \dots, N\}$, $x \in \mathcal{X}$ and $k > 0$ such that $\sum_{i \in P} L_g \tilde{J}_i(x) = -k \sum_{i \in Q} L_g \tilde{J}_i(x)$ where neither summation of Lie derivatives are zero, then:

$$\begin{aligned} \sum_{i \in P} L_g \tilde{J}_i(x) &= -k \sum_{i \in Q} L_g \tilde{J}_i(x) \tag{28} \\ \left(\sum_{i \in P} \frac{\partial \tilde{J}_i}{\partial x}(x) \right) g(x) &= \left(-k \sum_{i \in Q} \frac{\partial \tilde{J}_i}{\partial x}(x) \right) g(x) \\ \left(\sum_{i \in P} \frac{\partial \tilde{J}_i}{\partial x}(x) \right) g(x) g(x)^\top \left(g(x) g(x)^\top \right)^{-1} &= \left(-k \sum_{i \in Q} \frac{\partial \tilde{J}_i}{\partial x}(x) \right) g(x) g(x)^\top \left(g(x) g(x)^\top \right)^{-1} \\ \sum_{i \in P} \frac{\partial \tilde{J}_i}{\partial x}(x) &= -k \sum_{i \in Q} \frac{\partial \tilde{J}_i}{\partial x}(x) \\ \sum_{i \in P} \nabla_x \tilde{J}_i &= -k \sum_{i \in Q} \nabla_x \tilde{J}_i \end{aligned}$$

Now we must see if there exists some set $F \supset G$ such that no state exists within F where for some $P, Q \subset \{1, \dots, N\}$ the gradients of $\sum_{i \in P} \tilde{J}_i$ and $\sum_{i \in Q} \tilde{J}_i$ are in opposing directions.

Let $B_\delta := \{x \in \mathcal{X} : \exists g \in G, \|x - g\| < \delta\}$. For a suitably small $\delta > 0$, let $a \in B_\delta$ and

$x \in G$ such that:

$$\begin{aligned}
\tilde{J}_1(x) = 0 &= \tilde{J}_1(a) + \nabla_a \tilde{J}_1^\top (x - a) + \mathcal{O}(x, a) \\
\tilde{J}_2(x) = 0 &= \tilde{J}_2(a) + \nabla_a \tilde{J}_2^\top (x - a) + \mathcal{O}(x, a) \\
&\vdots \\
\tilde{J}_N(x) = 0 &= \tilde{J}_N(a) + \nabla_a \tilde{J}_N^\top (x - a) + \mathcal{O}(x, a)
\end{aligned} \tag{29}$$

where $\mathcal{O}(x, a)$ represents the higher order terms in each Taylor expansion. As the δ approaches 0, it is reasonable to ignore the higher order terms to get:

$$\begin{aligned}
\tilde{J}_1(x) = 0 &= \tilde{J}_1(a) + \nabla_a \tilde{J}_1^\top (x - a) \\
\tilde{J}_2(x) = 0 &= \tilde{J}_2(a) + \nabla_a \tilde{J}_2^\top (x - a) \\
&\vdots \\
\tilde{J}_N(x) = 0 &= \tilde{J}_N(a) + \nabla_a \tilde{J}_N^\top (x - a)
\end{aligned} \tag{30}$$

If $\nabla_a \tilde{J}_i = 0$ for each $i \in \{1, \dots, N\}$, then $\tilde{J}_1, \dots, \tilde{J}_N$ are clearly, by our definition in 3.3.2, concurrently controllable at a . Let $P, Q \subset \{1, \dots, N\}$. Let $p = \sum_{i \in P} \tilde{J}_i(a)$ and let $q = \sum_{i \in Q} \tilde{J}_i(a)$. Without loss of generality, assume that $p, q > 0$. We can then see that:

$$\begin{aligned}
0 &= \sum_{i \in P} \tilde{J}_i(a) + \sum_{i \in P} \nabla_a \tilde{J}_i^\top (x - a) = p + \langle \sum_{i \in P} \nabla_a \tilde{J}_i, x - a \rangle \\
0 &= \sum_{i \in Q} \tilde{J}_i(a) + \sum_{i \in Q} \nabla_a \tilde{J}_i^\top (x - a) = q + \langle \sum_{i \in Q} \nabla_a \tilde{J}_i, x - a \rangle
\end{aligned} \tag{31}$$

Let $k = \frac{p}{q}$. Multiplying the second of the previous equations by k , we see that:

$$\begin{aligned}
-p &= \langle \sum_{i \in P} \nabla_a \tilde{J}_i, x - a \rangle \\
-p &= k \langle \sum_{i \in Q} \nabla_a \tilde{J}_i, x - a \rangle
\end{aligned} \tag{32}$$

We can then see that:

$$\langle \sum_{i \in P} \nabla_a \tilde{J}_i, x - a \rangle = k \langle \sum_{i \in Q} \nabla_a \tilde{J}_i, x - a \rangle \tag{33}$$

This means that P, Q cannot be such that $\sum_{i \in P} \nabla_a \tilde{J}_i$ and $\sum_{i \in Q} \nabla_a \tilde{J}_i$ point in opposite directions which means that $\tilde{J}_1, \dots, \tilde{J}_N$ must be concurrently controllable at a . Therefore, for a small enough value of δ , $\tilde{J}_1, \dots, \tilde{J}_N$ are concurrently controllable at the states within B_δ . Since none of $\tilde{J}_1, \dots, \tilde{J}_N$ are identically equal to 0, and each of $\tilde{J}_1, \dots, \tilde{J}_N$ are continuous, it must be the case that B_δ is non-empty. $\square$

In Proposition 6, we show that when a set of value functions, $\tilde{J}_1, \dots, \tilde{J}_N$, have a common “goal” state, an equilibrium state where each value function is equal to 0, there is at least some non-empty set $F \subset \mathcal{X}$ where each of the value functions are concurrently executable. In the proof, we assume that $g(x)$ has full row rank which may seem like a strong assumption. However, for a system to have the redundancy necessary to concurrently execute multiple control tasks, it must be overactuated, making this assumption not unreasonable in this context.

4.3 Summary of Results

In this section, we will summarize the results shown in this chapter and their significance.

In Chapter 3, we specified the class of systems for which we wish to train tasks for as well as the cost functionals for which we wish to train value functions, using RL, to approximate. We then introduce the pointwise, min-norm, optimization-based controller, proposed in previous work in [26], which can be used to execute a set of learned value functions together concurrently. Using this controller, we see that the ability to concurrently execute learned value functions together relies on the relationships between the Lie derivatives, $L_g \tilde{J}_1, \dots, L_g \tilde{J}_N$. This leads us to define notions of *independence*, *orthogonality*, *concurrent executability* and *concurrent controllability*, which can classify the relationships between learned value functions based on their Lie derivatives.

In 4.1.1, we introduce an “interference” cost, as part of the cost functional proposed in (10). We prove in Proposition 1 that with large enough values of $\alpha_1, \dots, \alpha_{N-1}$, we can train an N th value function to be independent to a set of previously trained $N - 1$ value functions, allowing us to train a set of value functions to be independent. We can use this idea to train a set of value functions to be concurrently executable that may have otherwise had conflicting objectives. The resulting behaviours may be less optimal with respect to the cost on the state for a particular task, but are less likely to interfere with the execution of higher priority tasks when executed as part of the pointwise min-norm controller in (5). We also show in Proposition 2 the relationship between the optimal input of a value function and whether or not it is orthogonal to a set of other value functions.

The cost functional proposed in (10) in 4.1.1 can be approximated with various RL methods such as actor-critic methods which learn a value function in addition to a policy, or from evaluating a learned policy. However, in 4.1.2, we propose a model-based approach – a version of fitted value iteration in the continuous case that extends previous work in [21] and [7]. We show the algorithm in Algorithm 2 after justifying the approach using Proposition 3. While mainstream RL approaches can be used to approximate our proposed cost functional, this variant of value iteration may do so more efficiently given a reasonable model of the system and access to a GPU.

In 4.2, we propose necessary and sufficient conditions for it to be possible to concurrently execute learned value functions $\tilde{J}_1, \dots, \tilde{J}_N$ within a subset of the state space. The results here can be used to analyze value functions to determine which ones are possible to make independent to each other using the ideas in 4.1, which are not possible to do so and which may already be inherently compatible – at least within some subset of the state space.

In the next chapter we will demonstrate the ideas proposed in this chapter on various simulations of robotic systems learning to concurrently execute sets of learned tasks.

Chapter 5

Simulations and Experiments

In Sections 5.1, 5.2 and 5.3 we demonstrate the ideas presented in 4.1 by training robotic control tasks using our variant of CFVI to be concurrently executable that may have otherwise had conflicting objectives. In Section 5.5 we demonstrate the propositions from Section 4.2 in an illustrative example of a single mobile robot. Unless otherwise stated, a Δt of 0.05 was used to train each value function described in the sections in this chapter. While most relevant details are presented in these sections, we have also published code here: https://github.com/erablab/Value_Iteration_for_Learning_Concurrently_Executable_Robotic_Control_Tasks to help further guide practitioners in using many of the ideas presented in this thesis, particular Section 4.1. This code was initially published to supplement the work in [41] which encapsulates much of the work presented in this thesis. Additional implementation details, both in general and specific to the following sections, are provided in Appendix A.

5.1 Going to a Point and Avoiding a Square Region

We first demonstrate our proposed method by training a mobile robot to move to a point while avoiding a square-shaped region. The state space $\mathcal{X} = \mathbb{R}^2$ is the robot position and the input space $\mathcal{U} = \mathbb{R}^2$ is the velocity. Datasets sampled from a uniform distribution from the set $\{x \in \mathbb{R}^2 : \|x\|_1 \leq 3.1\} \subset \mathcal{X}$ were used in each iteration of training.

We first use the version of CFVI described in 3.4 to approximate a cost functional of

the form:

$$\min_{u(\cdot)} \int_0^t (q_1(x) + \|u\|^2) d\tau \quad (13)$$

where $q_1(x) = 60$ when the robot is within the region and 0 otherwise. We call this approximated cost-to-go function $\tilde{J}_1$ and plotted a heatmap of it in Fig. 1a.

Next, we train two versions of the task of moving the robot to the point $[-2 \ 0]^\top$. Both versions are trained to approximate a cost functional of the form:

$$\min_{u(\cdot)} \int_0^t (q_2(x) + u^\top R(x)u) d\tau$$

where $q_2(x)$ is the distance from the current position $x \in \mathcal{X}$ to $[-2 \ 0]^\top$ multiplied by 5. The first version, our “baseline” version in this case, is trained using the original CFVI method to approximate the cost functional with an instantaneous cost of $q_2(x) + \|u\|^2$. Here, we simply have that $R(x) = I$ in (13). The second version is trained, using our proposed method, with an instantaneous cost of $q_2(x) + \|u\|^2 + 10^4(L_g\tilde{J}_1(x)u)^2$. Here, we have that $R(x) = I + 10^4(L_g\tilde{J}_1(x))^\top(L_g\tilde{J}_1(x))$ in (13). We will refer to the resulting functions, both plotted in Fig. 1, as $\tilde{J}_{\text{base}}$ and $\tilde{J}_{\text{ind}}$ respectively. We can see that there is an imprint of the region in the case of $\tilde{J}_{\text{ind}}$.

In order to compare $\tilde{J}_{\text{base}}$ and $\tilde{J}_{\text{ind}}$, we formed two pointwise, min-norm controllers as described in (5) – one combining $\tilde{J}_1$ with $\tilde{J}_{\text{base}}$ and another combining $\tilde{J}_1$ with $\tilde{J}_{\text{ind}}$. The trajectories of the resulting controllers are shown in Fig. 2. We can see that since the gradients of $\tilde{J}_1$ and $\tilde{J}_{\text{base}}$ go in opposing directions, the robot is not able to move to the goal point from certain states. Meanwhile, our proposed approach influenced the gradients of $\tilde{J}_{\text{ind}}$ and $\tilde{J}_1$ to be linearly independent, such that when they are combined with the min-norm controller from Section 3.2, the resulting controller moves the robot to the goal point while successfully avoiding the region.

This illustrative example demonstrates how our approach can very visibly influence the gradients of a learned cost-to-go function to be independent to other cost-to-go functions, resulting in tasks that can be executed concurrently.

5.2 Forming a Shape and Avoiding a Square Region

In this next example, we train a team of three mobile robots to avoid a square-shaped region in the centre of the environment and form a triangle simultaneously. The state space,

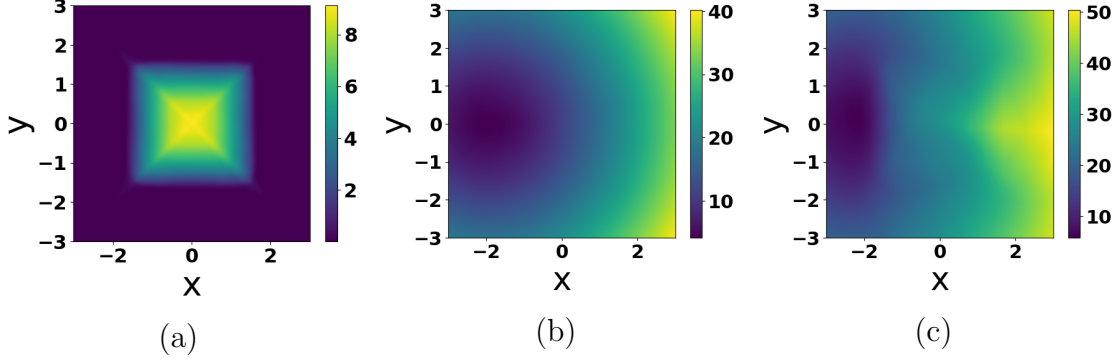


Figure 5.1: (a): Heatmap of function, $\tilde{J}_1$, trained for avoidance task. (b): Heatmap of function, $\tilde{J}_{\text{base}}$, trained using CFVI. (c): Heatmap of function, $\tilde{J}_{\text{ind}}$, trained using proposed method to be independent to avoidance task. Note that gradients of $\tilde{J}_{\text{ind}}$ appear to be linearly independent to those of $\tilde{J}_1$ as values in (c) appear to warp around the square region.

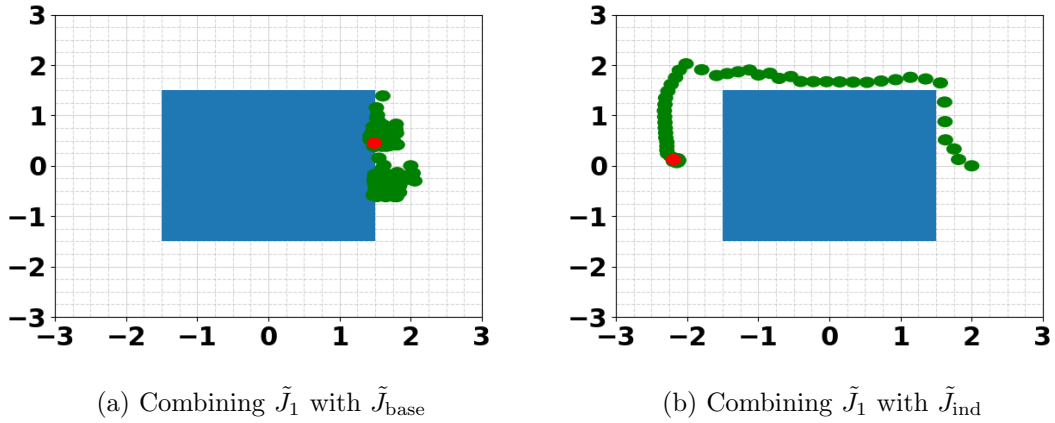


Figure 5.2: (a): Trajectory when combining avoidance task, $\tilde{J}_1$, with baseline go-to-point task, $\tilde{J}_{\text{base}}$. (b): Trajectory when combining $\tilde{J}_1$ with go-to-point task trained using proposed approach, $\tilde{J}_{\text{ind}}$. Green: Positions at each time step. Red: Final position for each trajectory. Blue: Region to avoid.

$\mathcal{X} = \mathbb{R}^6$, is the position of all robots where $\forall x \in \mathcal{X}, x = [p_1 \ p_2 \ p_3]^\top$ and $p_1, p_2, p_3 \in \mathbb{R}^2$ each represent the positions of robots 1, 2 and 3 respectively. The input space $\mathcal{U} = \mathbb{R}^6$ is the velocities of all robots where $\forall u \in \mathcal{U}, u = [v_1 \ v_2 \ v_3]^\top$ and $v_1, v_2, v_3 \in \mathbb{R}^2$ each represent the velocities of robots 1, 2 and 3 respectively.

We first train the task of avoiding the region by using CFVI to approximate a cost functional of the form:

$$\min_{u(\cdot)} \int_0^t (q_1(x) + \|u\|^2) d\tau.$$

We have that $q_1(x) = a_1(x) + a_2(x) + a_3(x)$ where $a_i(x) = 35$ if robot i is inside the 1x1 region and $a_i(x) = 0$ otherwise. We train this task as a single-robot task and convert it to a multi-robot task as described in Remark 5. When training the task for robot i , the dataset $\mathcal{D}$ is sampled from a uniform distribution of $\{p_i \in \mathbb{R}^2 : \|p_i\|_1\}$. After combining each trained value function into a single value function, we will denote it as $\tilde{J}_1$.

Remark 5. Consider a system comprising of N robots where each robot has a state space $\mathcal{X} = \mathcal{R}^n$. The state space for the whole system is $\mathcal{X} = \mathcal{R}^{Nn}$. Consider a learned cost-to-go function of the form $\tilde{j}(\bar{x})$ where $\bar{x} \in \bar{\mathcal{X}}$. This can be turned into a multi-robot task by defining $\tilde{J}(x) = \sum_{i \in T} \tilde{j}(\bar{x}_i)$, for $x \in \mathcal{X}$ and $x_i \in \bar{\mathcal{X}}$, where T is the set of robots to assign the single-robot task, $\tilde{j}$, to and $\bar{x}_i$ is the subset of state variables in x used to represent the state of robot i . When calculating $L_f \tilde{J}$ and $L_g \tilde{J}$ using back-propagation, every robot not assigned task $\tilde{j}$ will simply have gradients of zero. This is similar to what is described in Remark 5 in [26].

We then train the triangle formation task. We train two baseline trials using CFVI to approximate a cost functional:

$$\min_{u(\cdot)} \int_0^t (q_2(x) + \|u\|^2) d\tau.$$

We then train two trials using our proposed method to approximate a cost functional:

$$\min_{u(\cdot)} \int_0^t \left(q_2(x) + \|u\|^2 + (L_g \tilde{J}_1(x)u)^2 \alpha \right) d\tau$$

where $\alpha = 5 \cdot 10^4$. $q_2(x)$ is equal to:

$$(|\|p_1 - p_2\| - 0.75| + |\|p_1 - p_3\| - 0.75| + |\|p_2 - p_3\| - 0.75|) \cdot 1.5.$$

Table 5.1: Success Rates of Avoiding Region and Forming a Triangle

Training Method for Formation Task	Success Rate
CFVI - Trial 1	0.4
CFVI - Trial 2	0.34
Ours - Trial 1	0.74
Ours - Trial 2	0.92

$q_2(x) = 0$ when the robots form an equilateral triangle with each side of length 0.75. For both versions of the formation task, each iteration is trained from the dataset $\mathcal{D}$ where $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \mathcal{D}_3$. $\mathcal{D}_1$ is uniformly sampled from the set $\{x \in \mathcal{X} = \mathbb{R}^6 : \|x\|_1 \leq 2.1\}$. $\mathcal{D}_2$ is uniformly sampled from the set $\{x \in \mathcal{X} = \mathbb{R}^6 : p_1 = p_2 = p_3, \|p_1\|_1 \leq 2.1\}$. $\mathcal{D}_3$ was sampled by first uniformly sampling from states $x \in \mathcal{X}$ where $q(x) = 0$ and the robots form an equilateral triangle. Random noise was then added to the states in $\mathcal{D}_3$ to perturb them. This strategy for sampling $\mathcal{D}$ was done to train the formation task more efficiently.

After training all versions of the formation task, we combine each trained value function with $\tilde{J}_1$, using the min-norm controller from Section 3.2, and test on the same 50 initial states. The initial states were randomly generated from a uniform distribution of a 2x2 area encapsulating the region that $\tilde{J}_1$ was trained to avoid. We considered the team of robots to have successfully completed both the avoidance and formation tasks if $q_1(x) = 0$ and $q_2(x) < 0.4$. The results of this experiment are shown in Table 5.1. We can see that our proposed method allowed the team of robots to successfully avoid the region and form a triangle simultaneously much more frequently than when we simply trained the formation task using CFVI.

5.3 Forming a Shape and Going to a Point while Avoiding a Square-Shaped Region

We extend the previous example to train a team of three mobile robots to perform the following three tasks simultaneously: avoid three square-shaped regions in the environment, send one robot to a specified point and form the shape of a triangle. Like in the previous example, the state space, $\mathcal{X} = \mathbb{R}^6$, is the position of all robots where $\forall x \in \mathcal{X}, x = [p_1 \ p_2 \ p_3]^\top$ and $p_1, p_2, p_3 \in \mathbb{R}^2$ each represent the positions of robots 1, 2 and 3 respectively. The input space $\mathcal{U} = \mathbb{R}^6$ is the velocities of all robots where

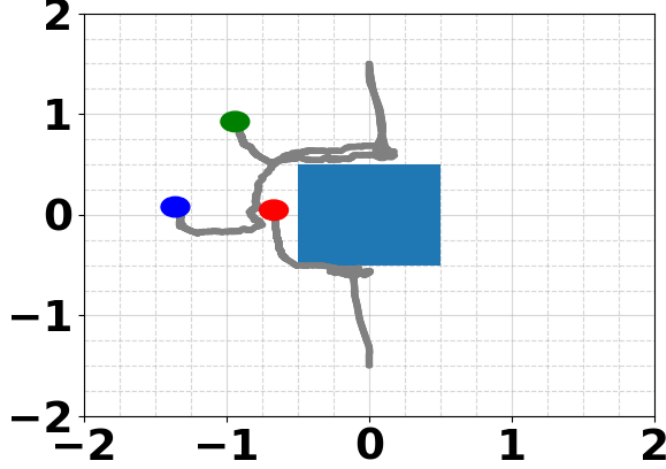


Figure 5.3: Robot team forming triangle while avoiding region. Coloured Dots: Robots 1-3. Lighter Blue: Region to avoid. Grey: Trajectories of each robot.

$\forall u \in \mathcal{U}, u = [v_1 \ v_2 \ v_3]^\top$ and $v_1, v_2, v_3 \in \mathbb{R}^2$ each represent the velocities of robots 1, 2 and 3 respectively. Each of the tasks we train in this simulation are similar to a task trained in the previous two sections. For each such task, we use the same process for sampling the dataset $\mathcal{D}$ during each iteration of training.

We first train the task of avoiding three obstacles placed in the environment. Like in the previous examples, we use CFVI to approximate a cost functional:

$$\min_{u(\cdot)} \int_0^t (q_1(x) + \|u\|^2) d\tau.$$

where $q_1(x) = b_1(x) + b_2(x) + b_3(x)$, $b_i(x) = 25$ if robot i is inside any of the regions and $b_i(x) = 0$ otherwise. Similar to the previous section, this task is trained for each robot and then combined into a value function for the whole system as described in Remark 5. We will denote the combined, learned, cost-to-go function for this task as $\tilde{J}_1$.

We then train the triangle formation task. As in the previous section, we use CFVI to approximate a cost functional:

$$\min_{u(\cdot)} \int_0^t (q_2(x) + \|u\|^2) d\tau$$

Table 5.2: Success Rates of Avoiding Region, Going to Origin and Forming a Triangle

Training Method for Formation Task	Training Method for Go-to-Point Task	Success Rate
CFVI - Trial 1	CFVI	0.24
CFVI - Trial 1	Ours	0.18
CFVI - Trial 2	CFVI	0.16
CFVI - Trial 2	Ours	0.78
CFVI - Trial 3	CFVI	0.2
CFVI - Trial 3	Ours	0.7
Ours - Trial 1	CFVI	0.32
Ours - Trial 1	Ours	0.82
Ours - Trial 2	CFVI	0.18
Ours - Trial 2	Ours	0.54
Ours - Trial 3	CFVI	0.32
Ours - Trial 3	Ours	0.72

for the baseline trials. For the trials testing our proposed method, we use our variant of CFVI to approximate the cost functional:

$$\min_{u(\cdot)} \int_0^t \left(q_2(x) + \|u\|^2 + (L_g \tilde{J}_1(x)u)^2 \alpha \right) d\tau$$

where $\alpha = 50000$. $q_2(x)$ is as defined previously in 5.2 where $q_2(x) = 0$ when the robots have formed an equilateral triangle of length 0.75. We train three baseline trials and three trials testing our proposed method.

Next, we train the single-robot task of going to the origin. We use an instantaneous cost of $q_3(\bar{x}) + \|\bar{u}\|^2$ for the baseline task trained with CFVI and an instantaneous cost of $q_3(\bar{x}_i) + \|\bar{u}_i\|^2 + (L_g \tilde{j}_i(\bar{x}_i)\bar{u}_i)^2$ for the task trained with our proposed method where $\bar{x}_i \in \mathcal{R}^2$ is the state for a single robot $i \in \{1, 2, 3\}$, $\bar{u}_i \in \mathbb{R}^2$ is the input velocity for a single robot and $q_3(\bar{x}_i)$ is the distance that robot is away from the origin multiplied by 12. $\tilde{j}_i$ is the component of $\tilde{J}_1$ for robot i . This single-robot task is turned into a multi-robot task in the way described in Remark 5. Note that we did not train the go-to-point task to explicitly be independent to the formation task as it should already be independent by construction.

Finally, we test each combination of trained models, combined using the min-norm controller from Section 3.2, on 50 randomly generated initial states and test the ability

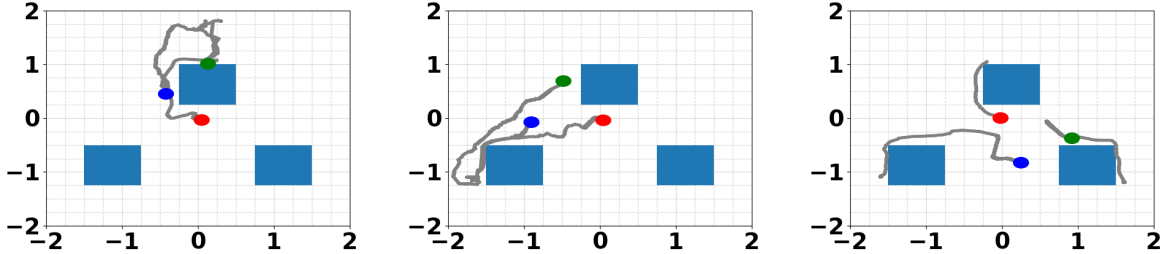


Figure 5.4: Robot team successfully forming triangle, sending first robot to origin and avoiding regions. Coloured Dots: Robots 1-3. Lighter Blue: Regions to avoid. Grey: Trajectories of each robot.

of each composed controller to achieve all three tasks. To randomly generate the initial states, we selected a position from a uniform distribution from space in the environment not covered by any of the three regions. We then added Gaussian noise to this position, to generate initial states where each robot was approximately close to one another. This was done to simulate the team of robots transporting objects collaboratively – a possible situation where one might want to decompose the tasks and train them in the way we did in this section. We consider the system to have successfully completed all three tasks when $q_1(x) = 0$, $q_2(x) < 0.75$ and $q_3(x) < 1.8$. The avoidance task was given the highest priority and the formation task was given the lowest.

The results of this experiment are shown in Table 5.2. We can see that the best performing controller is composed of a formation task and go-to-point task trained using our proposed method and that the worst performing controller is one where the formation task and go-to-point task were both trained using just CFVI and not to be independent to the avoidance task. The formation task seemed to benefit less from being trained to be independent to the avoidance task, compared to the previous example in 5.2, which may be due to how the initial states were generated and because the regions were smaller in this scenario. However, our proposed method still appeared to have made all three tasks more likely to be able to execute together concurrently. This simulation is an example of how a relatively complicated task such as “collaborative transporation” can be decomposed into several simpler tasks with each task then trained using our proposed methods to be compatible.

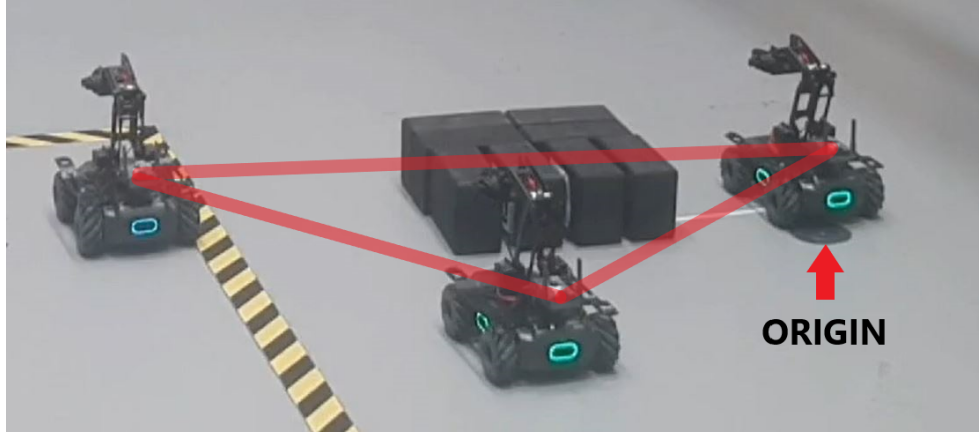


Figure 5.5: Physical team of robots forming a shape and sending one robot to a point while avoiding an obstacle

Flexibility in Constraint-Based Control Design

This example also demonstrates the benefit of our overall approach to multi-objective RL. The set of constraints and their priorities in the min-norm controller from (5) can be changed to solve some other task or even vary over time as needed. If we want to train our system to go to a different point or perform some other task while still avoiding the regions and forming a triangle, we can simply train a new third task that is independent to these two tasks.

5.4 Experiment with Physical Robots

We tested the best-performing controller from Section 5.3 on physical robots with a physical square-shaped obstacle, the results of which can be viewed here: <https://youtu.be/8FSjqIr-pgY>. We used a Vicon motion capture system to track the robots for feedback and sent velocities using the API for the DJI RoboMaster EP Core robots. The dimensions of the physical region to avoid were adjusted to account for the size of the robots. The physical obstacle seen in Figure 5.5 corresponds to the topmost region seen in Figure 5.4. We can see from Figure 5.5 that the team of mobile robots were able to send a designated robot to the origin while forming the shape of a triangle. This shows that the ideas presented in this thesis can be transferred and executed on real robots.

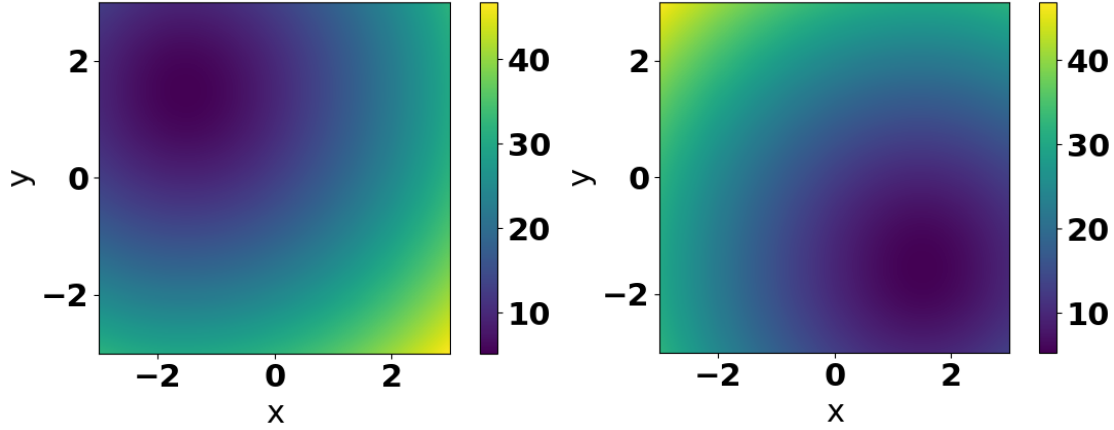


Figure 5.6: Heatmaps of $\tilde{J}_1$ and $\tilde{J}_2$ respectively

5.5 Illustrating Necessary and Sufficient Conditions for Concurrent Executability

In Section 4.2 of Chapter 4, we provided necessary and sufficient conditions for value functions to be concurrently executable within subsets of the state space. In this section, we will provide illustrative examples demonstrating those propositions. Like in Section 5.1, we will simply look at robotic tasks for a single mobile robot. We have that the state space $\mathcal{X} = \mathbb{R}^2$ is the robot position and the input space $\mathcal{U} = \mathbb{R}^2$ is the velocity.

First we will demonstrate the implications of Proposition 4 which tells us that if for a set of value functions $\tilde{J}_1, \dots, \tilde{J}_N$ there is no $x \in \mathcal{X}$ where $\tilde{J}_1(x) = \dots = \tilde{J}_N(x) = 0$, then there is no compact set $A \subset \mathcal{X}$ where it is possible to concurrently execute each task without leaving the set A . Consider the case where we train two value functions, each encoding the task of moving to a distinct point in the state space. $\tilde{J}_1$ is trained to go to the point $[-1.5 \ 1.5]^\top$ with an instantaneous cost of $q_1(x) + \|u\|^2$ and $\tilde{J}_2$ is trained to go to the point $[1.5 \ -1.5]^\top$ with an instantaneous cost of $q_2(x) + \|u\|^2$. $q_1(x)$ and $q_2(x)$ are the distances from $[-1.5 \ 1.5]^\top$ and $[1.5 \ -1.5]^\top$, respectively, multiplied by 5. Heatmaps of the two trained value functions can be found in Figure 5.6.

To numerically evaluate the relationship between the Lie derivatives, which simplifies to the gradients in the single integrator case, between $\tilde{J}_1$ and $\tilde{J}_2$, we plotted a heatmap of the angle between the gradients of $\tilde{J}_1$ and $\tilde{J}_2$ in Figure 5.7. Points that are closer to red are where $\tilde{J}_1$ and $\tilde{J}_2$ are not concurrently executable, as the angle between the gradients is

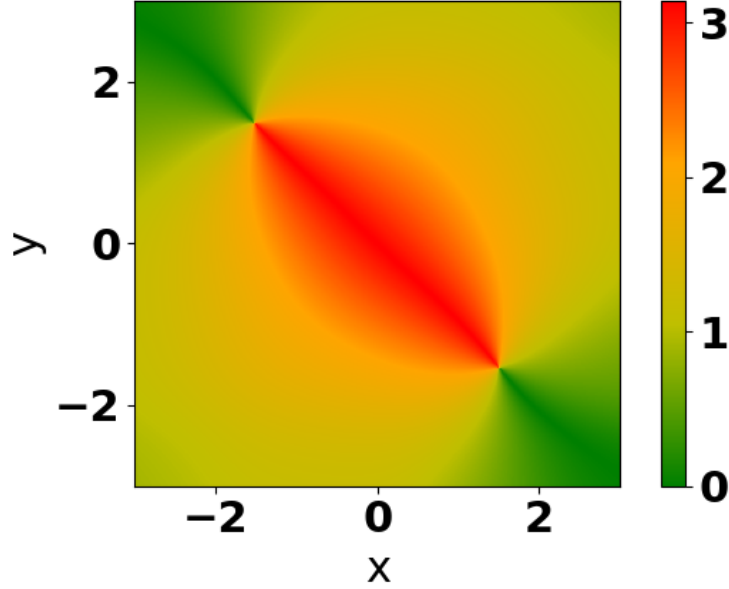


Figure 5.7: Heatmap of angles between the gradients of $\tilde{J}_1$ and $\tilde{J}_2$. Points that are closer to red are points where $\tilde{J}_1$ and $\tilde{J}_2$ are not concurrently executable.

around π , meaning that they point in opposite directions. Points that are closer to green point in close to the same direction as the angle between the two gradients is close to 0. As expected, we can see a line of states where $\tilde{J}_1$ and $\tilde{J}_2$ are not concurrently executable. We claim that it would not be possible to find a compact set $A \subset \mathcal{X}$ where it would be possible to concurrently execute both $\tilde{J}_1$ and $\tilde{J}_2$ without leaving A .

Next, we will demonstrate the implications of Propositions 5 and 6. $\tilde{J}_3$ is trained to be able to go to the nearest of four points among $[-1.5 \ 1.5]^\top$, $[1.5 \ 1.5]^\top$, $[-1.5 \ -1.5]^\top$ and $[1.5 \ -1.5]^\top$. It is trained using an instantaneous cost $q_3(x) + \|u\|^2$ where $q_3(x)$ is the distance to the nearest of the four points multiplied by 5. A heatmap of $\tilde{J}_3$ is plotted in Figure 5.8

Recall that we previously trained $\tilde{J}_2$ to go to the point $[1.5 \ -1.5]^\top$. To analyze the relationship between $\tilde{J}_2$ and $\tilde{J}_3$, we again plotted a heatmap in Figure 5.9 with the same colour code as before. Since $\tilde{J}_2$ and $\tilde{J}_3$ share a “goal” state, we would expect, based on Proposition 6, that there is some subset of the state space around the point $[1.5 \ -1.5]^\top$ where both $\tilde{J}_2$ and $\tilde{J}_3$ are concurrently executable. We can see that this is the case and it would be possible to draw a ball around the point where both $\tilde{J}_2$ and $\tilde{J}_3$ are concurrently

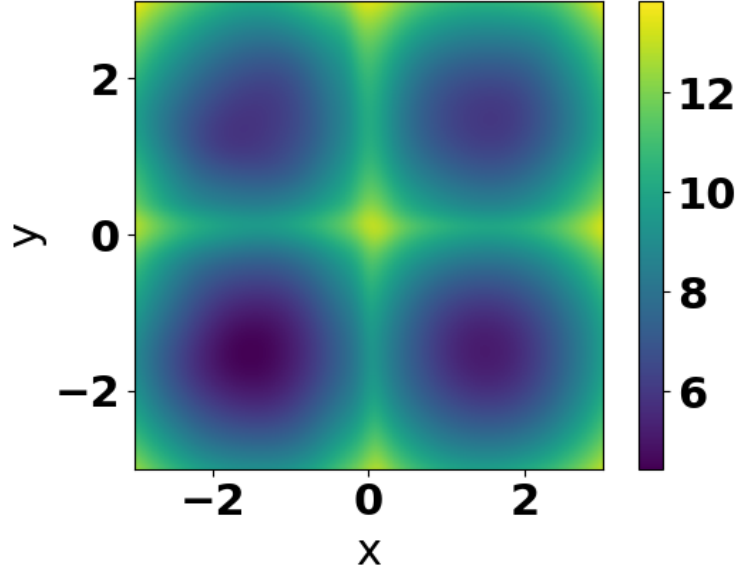


Figure 5.8: Heatmap of $\tilde{J}_3$

executable. However, as we expand the ball, we can see that it would eventually encompass states where $\tilde{J}_2$ and $\tilde{J}_3$ are not concurrently executable. Near these states, we can see that it would not be possible to make progress on one of the tasks without leaving the sublevel set of the other value function. This is an implication of Proposition 5.

The examples presented in this section are simple. However, by picking a state space that is in two dimensions, we were able to illustrate the implications of Propositions 4, 5 and 6 which provide insight into when it is possible for trained value functions to be concurrently executable.

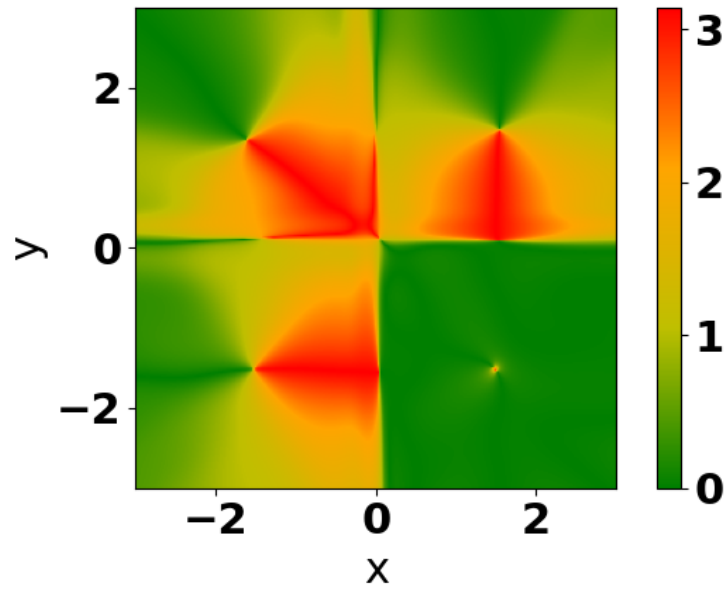


Figure 5.9: Heatmap of angles between the gradients of $\tilde{J}_2$ and $\tilde{J}_3$. Points that are closer to red are points where $\tilde{J}_2$ and $\tilde{J}_3$ are not concurrently executable.

Chapter 6

Conclusion and Future Work

This thesis built upon work in [26] to further develop a framework for training robotic control tasks using RL and executing them simultaneously. Based on our assumptions and chosen problem settings, we defined notions of independence, orthogonality, concurrent executability and concurrent controllability to characterize the relationship between learned value functions and the ability to execute them together concurrently. The main theoretical results were presented in Chapter 4. In Section 4.1, we presented ideas which can be used to train a set of value functions to be independent. An “interference” cost is introduced which can be used to make a value function independent to previously trained, higher priority value functions. To efficiently train such value functions given a known model, we introduce a variant of continuous fitted value iteration, building upon work in [21] which extended the approximate dynamic programming technique of fitted value iteration to systems with continuous input spaces. In Section 4.2, we presented propositions which provide necessary and sufficient conditions for a set of value functions to be concurrently executable within subsets of the state space. These propositions can guide practitioners in deciding which control tasks can be trained to be able to be concurrently executed together. We demonstrate the theoretical ideas presented in Chapter 4 in simulations and a physical robot experiment involving systems of one or more mobile robots executing multiple tasks concurrently.

While this thesis has presented extensive theoretical results and demonstrated the ideas in multiple scenarios, more work can be done to further develop this framework of training robotic control tasks to be concurrently executable. The following are possible future additions to the work:

- Deploy the techniques on systems with nonlinear dynamics.

- Demonstrate the techniques on more complex tasks involving more value functions and larger state and input spaces.
- Develop theory relating the gradient to the optimal input for a value function with an “interference” cost. In the pointwise min-norm controller, we select inputs that “go down” the gradient for each learned value function. As explained in 3.2.3, this does recover inputs that are “optimal” in some sense. However, when value functions incorporate the “interference” cost seen in (10), then the optimal inputs are no longer recovered, but picking inputs that “go down” the gradient does effectively accomplish the task. Therefore, developing theory about how “going down” the gradient for a value function relates to its optimal input would be helpful.
- Generalize definitions and related propositions concerning independence, orthogonality, concurrent executability and concurrent controllability to other types of functions used in the constraint-based control paradigm such as Control Barrier Functions.

There are other possible additions that can be made by further exploring the proposed ideas alongside other existing work within RL. Such additions include:

- Test the idea of the “interference” cost with other deep RL methods which may be better at exploring the policy space than simple value iteration.
- Further develop model-based methods for approximating value functions that may be more reliable and better at balancing between exploration and exploitation. The theory developed in this thesis assumes that optimal control problems are approximately solved using RL. If value functions can be reliably approximated, then these theorems become more valid.
- Incorporate learning to be independent to previous value functions with learning a model of the system.

Robots that exhibit redundancy, such as multi-robot systems, multi-joint arms and humanoids are systems which the ideas presented in this thesis can be useful for. A good application for the ideas presented in this thesis is environmental monitoring with a team of robots. Multi-robot systems are redundant, so exploiting their ability to execute multiple tasks is beneficial. Environmental monitoring involves performing tasks over long durations, so representing tasks as constraints, as is done in the pointwise min-norm controller used in this thesis, is also useful and has been done in previous works [29], [27], [2]. Since multi-robot systems have large state and input spaces and are not differentially

flat in general, training control tasks using RL may be useful. The idea of training value functions to be independent to other value functions may be useful in training tasks to be more “energy-aware”. For example, in environmental monitoring over long durations, a high priority task may be for robots to maintain their energy levels in order to survive, while another task may be for the team of robots to efficiently monitor the environment and collect data. While this was not tried yet, we suspect that training the latter task to be independent to the task of maintaining energy levels may result in a learned policy whose performance is less affected by robots having to recharge themselves. This scenario and other applications to long duration environmental monitoring serve as possible applications of the ideas presented in this thesis as well as extensions to this work.

References

- [1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [2] Aaron D. Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. Control barrier functions: Theory and applications. In *2019 18th European Control Conference (ECC)*, pages 3420–3431, 2019.
- [3] Gianluca Antonelli. Stability analysis for prioritized closed-loop inverse kinematic algorithms for redundant robotic systems. *IEEE Transactions on Robotics*, 25(5):985–994, 2009.
- [4] P. Baerlocher and R. Boulic. Task-priority formulations for the kinematic control of highly redundant articulated structures. In *Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications (Cat. No.98CH36190)*, volume 1, pages 323–329 vol.1, 1998.
- [5] D. Bertsekas. *Reinforcement Learning and Optimal Control*. Athena Scientific optimization and computation series. Athena Scientific, 2019.
- [6] Jorge Cortes and Magnus Egerstedt. Coordinated control of multi-robot systems: A survey. *SICE Journal of Control, Measurement, and System Integration*, 10(6):495–503, 2017.

- [7] Kenji Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12(1):219–245, 2000.
- [8] M. Egerstedt. *Robot Ecology: Constraint-Based Design for Long-Duration Autonomy*. Princeton University Press, 2021.
- [9] Adrien Escande, Nicolas Mansard, and Pierre-Brice Wieber. Hierarchical quadratic programming: Fast online humanoid-robot motion generation. *The International Journal of Robotics Research*, 33(7):1006–1028, 2014.
- [10] Felix Faber, Maren Bennewitz, and Sven Behnke. Controlling the gaze direction of a humanoid robot with redundant joints. In *RO-MAN 2008 - The 17th IEEE International Symposium on Robot and Human Interactive Communication*, pages 413–418, 2008.
- [11] R.A. Freeman and J.A. Primbs. Control lyapunov functions: new ideas from an old source. In *Proceedings of 35th IEEE Conference on Decision and Control*, volume 4, pages 3926–3931 vol.4, 1996.
- [12] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.
- [13] Ivo Grondman, Lucian Busoniu, Gabriel A. D. Lopes, and Robert Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307, 2012.
- [14] Tuomas Haarnoja, Vitchyr Pong, Aurick Zhou, Murtaza Dalal, Pieter Abbeel, and Sergey Levine. Composable deep reinforcement learning for robotic manipulation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6244–6251, 2018.
- [15] Conor F. Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M. Zintgraf, Richard Dazeley, Fredrik Heintz, Enda Howley, Athirai A. Irissappane, Patrick Mannion, Ann Nowé, Gabriel Ramos, Marcello Restelli, Peter Vamplew, and Diederik M. Roijers. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems*, 36(1):26, Apr 2022.

- [16] N. Hazon and G.A. Kaminka. Redundancy, efficiency and robustness in multi-robot coverage. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 735–741, 2005.
- [17] Xiangkun He, Zhongxu Hu, Haohan Yang, and Chen Lv. Personalized robotic control via constrained multi-objective reinforcement learning. *Neurocomputing*, 565:126986, 2024.
- [18] Hassan K Khalil. *Nonlinear systems; 3rd ed.* Prentice-Hall, Upper Saddle River, NJ, 2002. The book can be consulted by contacting: PH-AID: Wallet, Lionel.
- [19] Boqian Li, Shiping Wen, Zheng Yan, Guanghui Wen, and Tingwen Huang. A survey on the control lyapunov function and control barrier function for nonlinear-affine control systems. *IEEE/CAA Journal of Automatica Sinica*, 10(3):584–602, 2023.
- [20] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.
- [21] Michael Lutter, Shie Mannor, Jan Peters, Dieter Fox, and Animesh Garg. Value iteration in continuous actions, states and time. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 7224–7234. PMLR, 18–24 Jul 2021.
- [22] Kai-Chieh Ma, Zhibei Ma, Lantao Liu, and Gaurav S. Sukhatme. *Multi-robot Informative and Adaptive Planning for Persistent Environmental Monitoring*, pages 285–298. Springer International Publishing, Cham, 2018.
- [23] Dejan Milutinovic and Jacob Rosen. *Redundancy in Robot Manipulators and Multi-Robot Systems*. Springer Publishing Company, Incorporated, 2012.
- [24] Signe Moe, Gianluca Antonelli, Andrew R. Teel, Kristin Y. Pettersen, and Johannes Schrimpf. Set-based tasks within the singularity-robust multiple task-priority inverse kinematics framework: General formulation, stability analysis, and experimental results. *Frontiers in Robotics and AI*, 3, 2016.
- [25] Yoshihiko Nakamura, Hideo Hanafusa, and Tsuneo Yoshikawa. Task-priority based redundancy control of robot manipulators. *The International Journal of Robotics Research*, 6(2):3–15, 1987.

- [26] Gennaro Notomista. A constrained-optimization approach to the execution of prioritized stacks of learned multi-robot tasks. In Julien Bourgeois, Jamie Paik, Benoît Piranda, Justin Werfel, Sabine Hauert, Alyssa Pierson, Heiko Hamann, Tin Lun Lam, Fumitoshi Matsuno, Negar Mehr, and Abdallah Makhoul, editors, *Distributed Autonomous Robotic Systems*, pages 479–493, Cham, 2024. Springer Nature Switzerland.
- [27] Gennaro Notomista, Siddharth Mayya, Seth Hutchinson, and Magnus Egerstedt. An optimal task allocation strategy for heterogeneous multi-robot systems. In *2019 18th European Control Conference (ECC)*, pages 2071–2076, 2019.
- [28] Gennaro Notomista, Siddharth Mayya, Mario Selvaggio, María Santos, and Cristian Secchi. A set-theoretic approach to multi-task execution and prioritization. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9873–9879, 2020.
- [29] Gennaro Notomista, Claudio Pacchierotti, and Paolo Robuffo Giordano. Multi-robot persistent environmental monitoring based on constraint-driven execution of learned robot tasks. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 6853–6859, 2022.
- [30] Gennaro Notomista, Mario Selvaggio, María Santos, Siddharth Mayya, Francesca Pagano, Vincenzo Lippiello, and Cristian Secchi. Beyond jacobian-based tasks: Extended set-based tasks for multi-task execution and prioritization, 2023.
- [31] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. *PyTorch: an imperative style, high-performance deep learning library*. Curran Associates Inc., Red Hook, NY, USA, 2019.
- [32] Xue Bin Peng, Michael Chang, Grace Zhang, Pieter Abbeel, and Sergey Levine. Mcp: Learning composable hierarchical control with multiplicative compositional policies. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alche-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.
- [33] Athanasios S. Polydoros and Lazaros Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, May 2017.

- [34] James A. Primbs, Vesna Nevistić, and John C. Doyle. Nonlinear optimal control: A control lyapunov function and receding horizon perspective. *Asian Journal of Control*, 1(1):14–24, 1999.
- [35] Thomas M. Roehr. Active exploitation of redundancies in reconfigurable multirobot systems. *IEEE Transactions on Robotics*, 38(1):180–196, 2022.
- [36] Diederik M. Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *J. Artif. Int. Res.*, 48(1):67–113, oct 2013.
- [37] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.
- [38] Bruno Siciliano. Kinematic control of redundant robot manipulators: A tutorial. *Journal of Intelligent and Robotic Systems*, 3(3):201–212, Sep 1990.
- [39] Bruno Siciliano and J.-J.E. Slotine. A general framework for managing multiple tasks in highly redundant robotic systems. in fifth international conference on advanced robotics (vol. 2. pages 1211 – 1216 vol.2, 07 1991.
- [40] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.
- [41] Sheikh A. Tahmid and Gennaro Notomista. Value iteration for learning concurrently executable robotic control tasks. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems*, AAMAS ’25, page 2006–2014, Richland, SC, 2025. International Foundation for Autonomous Agents and Multiagent Systems.
- [42] Chen Tang, Ben Abbatematteo, Jiaheng Hu, Rohan Chandra, Roberto Martín-Martín, and Peter Stone. Deep reinforcement learning for robotics: A survey of real-world successes. *Annual Review of Control, Robotics, and Autonomous Systems*, 8(Volume 8, 2025):153–188, 2025.
- [43] Emanuel Todorov. Compositionality of optimal control laws. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc., 2009.

- [44] Benjamin Van Niekerk, Steven James, Adam Earle, and Benjamin Rosman. Composing value functions in reinforcement learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6401–6409. PMLR, 09–15 Jun 2019.
- [45] Jie Xu, Yunsheng Tian, Pingchuan Ma, Daniela Rus, Shinjiro Sueda, and Wojciech Matusik. Prediction-guided multi-objective reinforcement learning for continuous robot control. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 10607–10616. PMLR, 13–18 Jul 2020.
- [46] Chengxu Zhou, Xin Wang, Zhibin Li, Darwin Caldwell, and Nikos Tsagarakis. Exploiting the redundancy for humanoid robots to dynamically step over a large obstacle. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1599–1604, 2015.

APPENDICES

Appendix A

Implementation Details

The following are some miscellaneous implementation details that practitioners may find useful.

- Large values of α used in the simulations in Chapter 5 are not necessary and practitioners should be able to recreate similar results using much smaller values of α .
- Furthermore, values of α that are too large may lead to unstable training using fitted value iteration and other deep RL methods.
- Practitioners may find it helpful to bound the $\sum_{i=1}^N (L_g \tilde{J}_i(x)u)^2 \alpha_i$ term in the instantaneous cost. While we do not provide a rigorous study, we have also found that using the Softplus activation function [12], a smooth approximation of the standard ReLU activation function, within the network can lead to more stable training.
- The value functions described in Chapter 5 did not contain discount factors. However, each of them were, in practice, trained with a slight discount factor, a value between 0.97 and 0.99 in the discrete formulation, which were not strictly necessary but helped with convergence during training.