LEARNING MOBILE ROBOT MOTION CONTROL FROM DEMONSTRATION AND CORRECTIVE FEEDBACK

Brenna D. Argall

CMU-RI-TR-09-13

Submitted in partial fulfilment of
the requirements for the degree of
Doctor of Philosophy

Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

March 2009

Thesis Committee:
Brett Browning, Co-Chair
Manuela Veloso, Co-Chair
J. Andrew Bagnell
Chuck T. Thorpe
Maja J. Matarić, University of Southern California

ABSTRACT

FUNDAMENTAL to the successful, autonomous operation of mobile robots are robust motion control algorithms. Motion control algorithms determine an appropriate action to take based on the current state of the world. A robot observes the world through sensors, and executes physical actions through actuation mechanisms. Sensors are noisy and can mislead, however, and actions are non-deterministic and thus execute with uncertainty. Furthermore, the trajectories produced by the physical motion devices of mobile robots are complex, which make them difficult to model and treat with traditional control approaches. Thus, to develop motion control algorithms for mobile robots poses a significant challenge, even for simple motion behaviors. As behaviors become more complex, the generation of appropriate control algorithms only becomes more challenging. To develop sophisticated motion behaviors for a dynamically balancing differential drive mobile robot is one target application for this thesis work. Not only are the desired behaviors complex, but prior experiences developing motion behaviors through traditional means for this robot proved to be tedious and demand a high level of expertise.

One approach that mitigates many of these challenges is to develop motion control algorithms within a *Learning from Demonstration (LfD)* paradigm. Here, a behavior is represented as pairs of states and actions; more specifically, the states encountered and actions executed by a teacher during demonstration of the motion behavior. The control algorithm is generated from the robot learning a *policy*, or mapping from world observations to robot actions, that is able to reproduce the demonstrated motion behavior. Robot executions with any policy, including those learned from demonstration, may at times exhibit poor performance; for example, when encountering areas of the state-space unseen during demonstration. Execution experience of this sort can be used by a teacher to correct and update a policy, and thus improve performance and robustness.

The approaches for motion control algorithm development introduced in this thesis pair demonstration learning with *human feedback* on execution experience. The contributed feedback framework does *not* require revisiting areas of the execution space in order to provide feedback, a key advantage for mobile robot behaviors, for which revisiting an exact state can be expensive and often impossible. The types of feedback this thesis introduces range from binary indications of performance quality to execution corrections. In particular, *advice-operators* are a mechanism through which *continuous*-valued corrections are provided for *multiple* execution points. The advice-operator formulation is thus appropriate for low-level motion control, which operates in continuous-valued action spaces sampled at high frequency.

This thesis contributes multiple algorithms that develop motion control policies for mobile robot behaviors, and incorporate feedback in various ways. Our algorithms use feedback to refine demonstrated policies, as well as to build new policies through the scaffolding of simple motion behaviors learned from demonstration. We evaluate our algorithms empirically, both within simulated motion control domains and on a real robot. We show that feedback *improves* policy performance on simple behaviors, and *enables* policy execution of more complex behaviors. Results with the Segway RMP robot confirm the effectiveness of the algorithms in developing and correcting motion control policies on a mobile robot.

FUNDING SOURCES

We gratefully acknowledge the sponsors of this research, without whom this thesis would not have been possible:

- The Boeing Corporation, under Grant No. CMU-BA-GTA-1.
- The Qatar Foundation for Education, Science and Community Development.
- The Department of the Interior, under Grant No. NBCH1040007.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

ACKNOWLEDGEMENTS

There are many to thank in connection with this dissertation, and to begin I must acknowledge my advisors. Throughout the years they have critically assessed my research while supporting both the work and the person, extracting strengths and weaknesses alike, and have taught me to do the same. In Brett I have been fortunate to have an advisor who provided an extraordinary level of attention to, and hands on guidance in, the building of my knowledge and skill base, who ever encouraged me to think bigger and strive for greater achievements. I am grateful to Manuela for her insights and big-picture guidance, for the excitement she continues to show for and build around my thesis research, and most importantly for reminding me what we are and are not in the business of doing. Thank you also to my committee members, Drew Bagnell, Chuck Thorpe and Maja Matarić for the time and thought they put into the direction and assessment of this thesis.

The Carnegie Mellon robotics community as a whole has been a wonderful and supportive group. I am grateful to all past and present co-collaborators on the various Segway RMP projects (Jeremy, Yang, Kristina, Matt, Hatem), and other Treasure Hunt team members (Gil, Mark, Balajee, Freddie, Thomas, Matt), for companionship in countless hours of robot testing and finding humor in the frustration of broken networks and broken robots. I acknowledge Gil especially, for invariably providing his balanced perspective. Thanks also to Sonia, for all of the quick questions and reference checks. Thank you to all of the Robotics Institute and School of Computer Science staff, who do such a great job supporting us students.

I owe a huge debt to all of my teachers throughout the years, who shared with me their knowledge and skills, supported inquisitive and critical thought, and underlined that the ability to learn is more useful than facts; thank you for making me a better thinker. I am especially grateful to my childhood piano teachers, whose approach to instruction that combines demonstration and iterative corrections provided much inspiration for this work.

To the compound and all of its visitors, for contributing so instrumentally to the richness of my life beyond campus, I am forever thankful. Especially to Lauren, for being the best backyard neighbor, and to Pete, for being a great partner with which to explore the neighborhood and world, not to mention for providing invaluable support through the highs and lows of this dissertation development. To the city of Pittsburgh, home of many wonderful organizations in which I have been honored to take part, and host to many forgotten, fascinating spots which I have been delighted to discover. Graduate school was much more fun that it was supposed to be.

Last, but very certainly not least, I am grateful to my family, for their love and support throughout all phases of my life that have led to this point. I thank my siblings - Alyssa, Evan, Lacey, Claire, Jonathan and Brigitte - for enduring my childhood bossiness, for their continuing friendship, and for pairing nearly every situation with laughter, however snarky. I thank my parents, Amy and Dan, for so actively encouraging my learning and education: from answering annoyingly exhaustive childhood questions and sending me to great schools, through to showing interest in foreign topics like robotics research. I am lucky to have you all, and this dissertation could not have happened without you.

DEDICATION

To my grandparents

Frances, Big Jim, Dell and Clancy; for building the family of which I am so fortunate to be a part, and in memory of those who saw me begin this degree but will not see its completion.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOTATION

S : the set of world states, consisting of individual states $s \in S$

Z : the set of observations of world state, consisting of individual observations $\mathbf{z} \in Z$

A : the set of robot actions, consisting of individual actions $\mathbf{a} \in A$

$T(s'|s,a)$: a probabilistic mapping between states by way of actions

D : a set of teacher demonstrations, consisting of individual demonstrations $d \in D$

Π : a set of policies, consisting of individual policies $\pi \in \Pi$

Φ : indication of an execution trace segment

d_Φ : the subset of datapoints within segment Φ of execution d, `s.t.` $d_\Phi \subseteq d$

$\digamma$: general teacher feedback

c : specific teacher feedback of the performance credit type

op : specific teacher feedback of the advice-operator type

$\phi(:)$: a kernel distance function for regression computations

Σ^{-1} : a diagonal parameter matrix for regression computations

m : a scaling factor associated with each point in D under the BC algorithm

ξ : label for a demonstration set, `s.t.` policy π_ξ derives from dataset D_ξ; used to annotate behavior primitives under algorithm FPS and data sources under algorithm DWL

Ξ : a set of demonstration set labels, consisting of individual labels $\xi \in \Xi$

τ : an indication of dataset support for a policy prediction under the algorithm FPS

$\mathbf{w}$: the set of data source weights under the algorithm DWL, consisting of individual weights $w_\xi \in \mathbf{w}$

r : reward (execution or state)

λ : parameter of the Poisson distribution modeling *1*-NN distances between dataset points

μ **:** mean of a statistical distribution

σ **:** standard deviation of a statistical distribution

$\alpha, \beta, \delta, \epsilon, \kappa$ **:** implementation-specific parameters

(x, y, θ) **:** robot position and heading

(ν, ω) **:** robot translational and rotational speeds, respectively

$g_R\,(z, a)$ **:** LfD record mapping

$g_E\,(z, a)$ **:** LfD embodiment mapping

CHAPTER 1

Introduction

ROBUST motion control algorithms are fundamental to the successful, autonomous operation of mobile robots. Robot movement is enacted through a spectrum of mechanisms, from wheel speeds to joint actuation. Even the simplest of movements can produce complex motion trajectories, and consequently robot motion control is known to be a difficult problem. Existing approaches that develop motion control algorithms range from model-based control to machine learning techniques, and all require a high level of expertise and effort to implement. One approach that addresses many of these challenges is to teach motion control through *demonstration*.

In this thesis, we contribute approaches for the development of motion control algorithms for mobile robots, that build on the demonstration learning framework with the incorporation of human *feedback*. The types of feedback considered range from binary indications of performance quality to execution corrections. In particular, one key contribution is a mechanism through which continuous-valued corrections are provided for motion control tasks. The use of feedback spans from refining low-level motion control to building algorithms from simple motion behaviors. In our contributed algorithms, teacher feedback augments demonstration to improve control algorithm performance and enable new motion behaviors, and does so more effectively than demonstration alone.

1.1. Practical Approaches to Robot Motion Control

Whether an exploration rover in space or recreational robot for the home, successful autonomous mobile robot operation requires a motion control algorithm. A *policy* is one such control algorithm form, that maps observations of the world to actions available on the robot. This mapping is fundamental to many robotics applications, yet in general is complex to develop.

The development of control policies requires a significant measure of effort and expertise. To implement existing techniques for policy development frequently requires extensive prior knowledge and parameter tuning. The required prior knowledge ranges from details of the robot and its movement mechanisms, to details of the execution domain and how to implement a given control algorithm. Any successful application typically has the algorithm highly tuned for operation with a particular robot in a specific domain. Furthermore, existing approaches are often applicable only to simple tasks due to computation or task representation constraints.

1.1.1. Policy Development and Low-Level Motion Control

The state-action mapping represented by a motion policy is typically complex to develop. One reason for this complexity is that the target observation-action mapping is unknown. What *is* known is the desired robot motion behavior, and this behavior must somehow be represented through an unknown observation-action mapping. How accurately the policy derivation techniques then reproduce the mapping is a separate and additional challenge. A second reason for this complexity are the complications of motion policy execution in real world environments. In particular:

1. The world is observed through sensors, which are typically noisy and thus may provide inconsistent or misleading information.

2. Models of world dynamics are an approximation to the true dynamics, and are often further simplified due to computational or memory constraints. These models thus may inaccurately predict motion effects.

3. Actions are motions executed with real hardware, which depends on many physical considerations such as calibration accuracy and necessarily executes actions with some level of imprecision.

All of these considerations contribute to the inherent uncertainty of policy execution in the real world. The net result is a difference between the expected and actual policy execution.

Traditional approaches to robot control model the domain dynamics and derive policies using these mathematical models. Though theoretically well-founded, these approaches depend heavily upon the accuracy of the model. Not only does this model require considerable expertise to develop, but approximations such as linearization are often introduced for computational tractability, thereby degrading performance. Other approaches, such as *Reinforcement Learning*, guide policy learning by providing reward feedback about the desirability of visiting particular states. To define a function to provide these rewards, however, is known to be a difficult problem that also requires considerable expertise to address. Furthermore, building the policy requires gathering information by visiting states to receive rewards, which is non-trivial for a mobile robot learner executing actual actions in the real world.

Motion control policy mappings are able to represent actions at a variety of control levels.

Low-level actions: Low-level actions directly control the movement mechanisms of the robot. These actions are in general continuous-valued and of short time duration, and a low-level motion policy is sampled at a high frequency.

High-level actions: High-level actions encode a more abstract action representation, which is then translated through other means to affect the movement mechanisms of the robot; for example, through another controller. These actions are in general discrete-valued and of longer time duration, and their associated control policies are sampled at a low frequency.

In this thesis, we focus on low-level motion control policies. The continuous action-space and high sampling rate of low-level control are all key considerations during policy development.

The particular robot platform used to validate the approaches of this thesis is a Segway Robot Mobility Platform (Segway RMP), pictured in Figure 1.1. The Segway RMP is a two-wheeled dynamically-balancing differential drive robot (Nguyen et al., 2004). The robot balancing mechanism is founded on inverse pendulum dynamics, the details of which are proprietary information of the Segway LLC company and are essentially unknown to us. The absence of details fundamental to the robot motion mechanism, like the balancing controller, complicates the development of motion behaviors for this robot, and in particular the application of dynamics-model-based motion control techniques. Furthermore, this robot operates in complex environments that demand sophisticated motion behaviors. Our developed behavior architecture for this robot functions as a finite state machine, where high-level behaviors are built on a hierarchy of other behaviors. In our previous work, each low-level motion behavior was developed and extensively tuned by hand.

FIGURE 1.1. The Segway RMP robot.

The experience of personally developing numerous motion behaviors by hand for this robot (Argall et al., 2006, 2007b), and subsequent desire for more straightforward policy development techniques, was a strong motivating factor in this thesis. Similar frustrations have been observed in other roboticists, further underlining the value of approaches that ease the policy development process. Another, more hypothetical, motivating factor is that as familiarity with robots within general society becomes more prevalent, it is expected that future robot operators will include those who are *not* robotics experts. We anticipate a future requirement for policy development approaches that not only ease the development process for experts, but are accessible to non-experts as well. This thesis represents a first step towards this goal.

1.1.2. Learning from Demonstration

Learning from Demonstration (LfD) is a policy development technique with the potential for both application to non-trivial tasks and straightforward use by robotics-experts and non-experts

alike. Under the LfD paradigm, a teacher first demonstrates a desired behavior to the robot, producing an example state-action trace. The robot then generalizes from these examples to learn a state-action mapping and thus derive a policy.

LfD has many attractive points for both learner and teacher. LfD formulations typically do not require expert knowledge of the domain dynamics, which removes performance brittleness resulting from model simplifications. The relaxation of the expert knowledge requirement also opens policy development to non-robotics-experts, satisfying a need which we expect will increase as robots become more commonplace. Furthermore, demonstration has the attractive feature of being an intuitive medium for communication from humans, who already use demonstration to teach other humans.

More concretely, the application LfD to motion control has a variety of advantages:

Implicit behavior to mapping translation: By demonstrating a desired motion behavior, and recording the encountered states and actions, the translation of a behavior into a representative state-action mapping is immediate and implicit. This translation therefore does not need to be explicitly identified and defined by the policy developer.

Robustness under real world uncertainty: The uncertainty of the real world means that multiple demonstrations of the same behavior will not execute identically. Generalization over examples therefore produces a policy that does not depend on a strictly deterministic world, and thus will execute more robustly under real world uncertainty.

Focused policies: Demonstration has the practical feature of focusing the dataset of examples to areas of the state-action space actually encountered during behavior execution. This is particularly useful in continuous-valued action domains, with an infinite number of state-action combinations.

LfD has enabled successful policy development for a variety of robot platforms and applications. This approach is not without its limitations, however. Common sources of LfD limitations include:

1. Suboptimal or ambiguous teacher demonstrations.
2. Uncovered areas of the state space, absent from the demonstration dataset.
3. Poor translation from teacher to learner, due to differences in sensing or actuation.

This last source relates to the broad issue of *correspondence* between the teacher and learner, who may differ in sensing or motion capabilities. In this thesis, we focus on demonstration techniques that do not exhibit strong correspondence issues.

1.1.3. Control Policy Refinement

A robot will likely encounter many states during execution, and to develop a policy that appropriately responds to all world conditions is difficult. Such a policy would require that the policy developer had prior knowledge of which world states would be visited, which is unlikely in real-world

domains, in addition to knowing the correct action to take from each of these states. An approach that circumvents this requirement is to refine a policy in response to robot execution experience. Policy refinement from execution experience requires both a mechanism for evaluating an execution, as well as a framework through which execution experience may be used to update the policy.

Executing a policy provides information about the task, domain and the effects of policy execution of this task in the given domain. Unless a policy is already optimal for every state in the world, this execution information can be used to refine and improve the policy. Policy refinement from execution experience requires both detecting the relevant information, for example observing a failure state, and also incorporating the information into a policy update, for example producing a policy that avoids the failure state. *Learning from Experience*, where execution experience is used to update the learner policy, allows for increased policy robustness and improved policy performance. A variety of mechanisms may be employed to learn from experience, including the incorporation of performance feedback, policy corrections or new examples of good behavior.

One approach to learning from experience has an external source offer *performance feedback* on the policy execution. For example, feedback could indicate specific areas of poor or good policy performance, which is one of the feedback approaches considered in this thesis. Another feedback formulation could draw attention to elements of the environment that are important for task execution.

Within the broad topic of machine learning, performance feedback provided to a learner typically takes the form of state reward, as in *Reinforcement Learning*. State rewards provide the learner with an indication of the desirability of visiting a particular state. To determine whether a different state would have been more desirable to visit instead, alternate states must be visited, which can be unfocused and intractable to optimize when working on real robot systems in motion control domains with an infinite number of world state-action combinations. State rewards are generally provided automatically by the system and tend to be sparse, for example zero for all states except those near the goal. One challenge to operating in worlds with sparse reward functions is the issue of reward back-propagation; that is, of crediting key early states in the execution for *leading* to a particular reward state.

An alternative to overall performance feedback is to provide a *correction* on the policy execution, which is another feedback form considered in this thesis. Given a current state, such information could indicate a preferred action to take, or a preferred state into which to transition, for example. Determining which correction to provide, however, is typically a task sufficiently complex to preclude a simple sparse function from providing a correction. The complexity of a correction formulation grows with the size of the state-action space, and becomes particularly challenging in continuous state-action domains.

Another policy refinement technique, particular to LfD, provides the learner with *more teacher demonstrations*, or more examples of good behavior executions. The goal of this approach is to provide examples that clarify ambiguous teacher demonstrations or visit previously undemonstrated areas of the state-space. Having the teacher provide more examples, however, is unable to address all sources of LfD error, for example correspondence issues or suboptimal teacher performance. The more-demonstrations approach also requires revisiting the target state in order to provide a

demonstration, which can be non-trivial for large state-spaces such as motion control domains. The target state may be difficult, dangerous or even impossible to access. Furthermore, the motion path taken to visit the state can constitute a poor example of the desired policy behavior.

1.2. Approach

This thesis contributes an effective LfD framework to address common limitations within LfD that cannot improve through further demonstration alone. Our techniques build and refine motion control policies using a combination of demonstration and human feedback, which takes a variety of forms. Of particular note is the contributed formulation of advice-operators, which correct policy executions within continuous-valued, motion control domains. Our feedback techniques build and refine individual policies, as well as facilitate the incorporation of multiple policies into the execution of more complex tasks. The thesis validates the introduced policy development techniques in both simulated and real robot domains.

1.2.1. Algorithms Overview

This work introduces algorithms that build policies through a combination of demonstration and teacher feedback. The document first presents algorithms that are novel in the *type* of feedback provided; these are the *Binary Critiquing* and *Advice-Operator Policy Improvement* algorithms. This presentation is followed with algorithms that are novel in their *incorporation* of feedback into a complex behavior policy; these are the *Feedback for Policy Scaffolding* and *Demonstration Weight Learning* algorithms.

In the first algorithm, *Binary Critiquing (BC)*, the human teacher flags poorly performing areas of learner executions. The learner uses this information to modify its policy, by penalizing the underlying demonstration data that supported the flagged areas. The penalization technique addresses the issue of suboptimal or ambiguous teacher demonstrations. This sort of feedback is arguably well-suited for human teachers, as humans are generally good at assigning basic performance credit to executions.

In the second algorithm, *Advice-Operator Policy Improvement* (A-OPI), a richer feedback is provided by having the human teacher provide *corrections* on the learner executions. This is in contrast to BC, where poor performance is only flagged and the correct action to take is not indicated. In A-OPI the learner uses corrections to *synthesize* new data based on its own executions, and incorporates this data into its policy. Data synthesis can address the LfD limitation of dataset sparsity, and the A-OPI synthesis technique provides an alternate source for data - a key novel feature of the A-OPI algorithm - that does not derive from teacher demonstrations. Providing an alternative to teacher demonstration addresses the LfD limitation of teacher-learner correspondence, as well as suboptimal teacher demonstrations. To provide corrective feedback the teacher selects from a finite predefined list of corrections, named *advice-operators*. This feedback is translated by the learner into *continuous-valued* corrections suitable for modifying low-level motion control actions, which is the target application domain for this work.

The third algorithm, *Feedback for Policy Scaffolding (FPS)*, incorporates feedback into a policy built from simple behavior policies. Both the built-up policy and the simple policies incorporate

teacher feedback that consists of good performance flags and corrective advice. To begin, the simple behavior policies, or *motion primitives*, are built under a slightly modified version of the A-OPI algorithm. The policy for a more complex, undemonstrated task is then developed, that operates by selecting between novel and motion primitive policies. More specifically, the teacher provides feedback on executions with the complex policy. Data resulting from teacher feedback is then used in two ways. The first updates the underlying primitive policies. The second builds novel policies, exclusively from data generated as a result of feedback.

The fourth algorithm, *Demonstration Weight Learning (DWL)*, incorporates feedback by treating different types of teacher feedback as distinct data sources, with the two feedback types empirically considered being good performance flags and corrective advice. Different teachers, or teaching styles, are additionally treated as separate data sources. A policy is derived from each data source, and the larger policy selects between these sources at execution time. DWL additionally associates a performance-based weight with each source. The weights are learned and automatically updated under an expert learning inspired paradigm, and are considered during policy selection.

1.2.2. Results Overview

The above algorithms build motion control policies through demonstration and human feedback, and are validated within both simulated and real-world implementations.

In particular, BC is implemented on a realistic simulation of a differential drive robot, modeled on the Segway RMP, performing a motion interception task. The presented results show that human teacher critiquing does improve task performance, measured by interception success and efficiency. A-OPI is implemented on a real Segway RMP robot performing spatial positioning tasks. The A-OPI algorithm enables similar or superior performance when compared to the more typical LfD approach to behavior correction that provides more teacher demonstrations. Furthermore, by concentrating new data exclusively to the areas visited by the robot and needing improvement, A-OPI produces noticeably smaller datasets.

Both algorithms FPS and DWL are implemented within a simulated motion control domain, where a differential drive robot performs a racetrack driving task. The domain is again modeled on the Segway RMP robot. Under the FPS framework, motion control primitives are successfully learned from demonstration and teacher feedback, and a policy built from these primitives and further teacher feedback is able to perform a more complex task. Performance improvements in success, speed and efficiency are observed, and all FPS policies far outperform policies built from extensive teacher demonstration. In the DWL implementation, a policy built from multiple weighted demonstration sources successfully learns the racetrack driving task. Data sources are confirmed to be unequally reliable in the experimental domain, and data source weighting is shown to impact policy performance. The weights automatically learned by the DWL algorithm are further demonstrated to accurately reflect data source reliability.

A framework for providing teacher feedback, named *Focused Feedback for Mobile Robot Policies (F3MRP)*, is additionally contributed and evaluated in this work. In particular, an in-depth look at the design decisions required in the development of a feedback framework is provided. Extensive details of the F3MRP framework are presented, as well as an analysis of data produced under

this framework and in particular resulting from corrective advice. Within the presentation of this corrective feedback technique, named *advice-operators*, a principled approach to their development is additionally contributed.

1.3. Thesis Contributions

This thesis considers the following research questions:

> *How might teacher feedback be used to address and correct common Learning from Demonstration limitations in low-level motion control policies?*
>
> *In what ways might the resulting feedback techniques be incorporated into more complex policy behaviors?*

To address common limitations of LfD, this thesis contributes mechanisms for providing human feedback in the form of performance flags and corrective advice, and algorithms that incorporate these feedback techniques. For the incorporation into more complex policies, human feedback is used in the following ways: to build and correct demonstrated policy primitives; to link the execution of policy primitives and correct these linkages; and to produce policies that are considered along with demonstrated policies during the complex motion behavior execution.

The contributions of this thesis are the following.

Advice-Operators: A feedback formulation that enables the correction of *continuous*-valued policies. An in-depth analysis of correcting policies with in continuous action spaces, and the data produced by our technique, is also provided.

Framework *Focused Feedback for Mobile Robot Policies*: A policy improvement framework for the incorporation of teacher feedback on learner executions, that allows for the application of a *single* piece of feedback over *multiple* execution points.

Algorithm *Binary Critiquing*: An algorithm that uses teacher feedback in the form of binary *performance flags* to refine motion control policies within a demonstration learning framework.

Algorithm *Advice-Operator Policy Improvement*: An algorithm that uses teacher feedback in the form of *corrective advice* to refine motion control policies within a demonstration learning framework.

Algorithm *Feedback for Policy Scaffolding*: An algorithm that uses multiple forms of teacher feedback to *scaffold* primitive behavior policies, learned through demonstration, into a policy that exhibits a more complex behavior.

Algorithm *Demonstration Weight Learning*: An algorithm that considers multiple forms of teacher feedback as individual *data sources*, along with multiple demonstrators, and learns to select between sources based on reliability.

Empirical Validation: The algorithms of this thesis are all empirically implemented and evaluated, within both real world - using a Segway RMP robot - and simulated motion control domains.

Learning from Demonstration Categorization: A framework for the categorization of techniques typically used in robot Learning from Demonstration.

1.4. Document Outline

The work of this thesis is organized into the following chapters.

- Chapter 2 overviews the LfD formulation of this thesis, identifies the design decisions involved in building a feedback framework, and details the contributed Focused Feedback for Mobile Robot Policies framework along with our baseline feedback algorithm.
- Chapter 3 introduces the Binary Critiquing algorithm, and presents empirical results along with a discussion of binary puntative feedback.
- Chapter 4 presents the contributed advice-operator technique along with an empirical comparison to an exclusively demonstration technique, and introduces an approach for the principled development of advice-operators.
- Chapter 5 introduces the Advice-Operator Policy Improvement algorithm, presents the results from an empirical case study as well as a full task implementation, and provides a discussion of corrective feedback in continuous-action domains.
- Chapter 6 introduces the Feedback for Policy Scaffolding algorithm, presents an empirical validation from building both motion primitive and complex policy behaviors, and provides a discussion of the use of teacher feedback to build complex motion behaviors.
- Chapter 7 introduces the Demonstration Weight Learning algorithm, and presents empirical results along with a discussion of the performance differences between multiple demonstration sources.
- Chapter 8 presents our contributed LfD categorization framework, along with a placement of relevant literature within this categorization.
- Chapter 9 presents published literature relevant to the topics addressed, and techniques developed, in this thesis.
- Chapter 10 overviews the conclusions of this work.

CHAPTER 2

Policy Development and Execution Feedback

POLICY development is typically a complex proccess that requires a large investment in time and expertise on the part of the policy designer. Even with a carefully crafted policy, a robot often will not behave as the designer expects or intends in all areas of the execution space. One way to address behavior shortcomings is to update a policy based on execution experience, which can increase policy robustness and overall performance. For example, such an update may expand the state-space in which the policy operates, or increase the likelihood of successful task completion. Many policy updates depend on *evaluations* of execution performance. Human teacher feedback is one approach for providing a policy with performance evaluations.

This chapter identifies many of the design decisions involved in the development of a feedback framework. We contribute the framework *Focused Feedback for Mobile Robot Policies (F3MRP)* as a mechanism through which a teacher provides feedback on mobile robot motion control executions. Through the F3MRP framework, human teacher feedback updates the motion control policy of a mobile robot. The F3MRP framework is distinguished by operating at the stage of low-level motion control, where actions are continuous-valued and sampled at high frequency. Some noteworthy characteristics of the F3MRP framework are the following. A visual presentation of the 2-D ground path of the mobile robot execution serves as an interface through which the teacher selects the segments of an execution that are to receive feedback, which simplifies the challenge of providing feedback to policies sampled at a high frequency. Visual indications of data support during an execution assist the teacher in the selection of execution segments and feedback type. An interactive tagging mechanism enables close association between teacher feedback and the learner execution.

Our feedback techniques build on a *Learning from Demonstration (LfD)* framework. Under LfD, examples of behavior execution by a teacher are provided to a student. In our work, the student derives a *policy*, or state-action mapping, from the dataset of these examples. This mapping enables the learner to select an action to execute based on the current world state, and thus provides a control algorithm for the target behavior. Though LfD has been successfully employed for a variety of robotics applications (see Ch. 8), the approach is not without its limitations. This thesis aims to address limitations common to LfD, and in particular those that do not improve with more teacher demonstration. Our approach for addressing LfD limitations is to provide human teacher feedback on learner policy executions.

The following section provides a brief overview of policy development under LfD, including a delineation of the specific form LfD takes in this thesis. Section 2.2 identifies key design decisions that define a feedback framework. Our feedback framework, F3MRP, is then described in Section 2.3. We present our general feedback algorithm in Section 2.4.1, which provides a base for all algorithms presented later in the document.

2.1. Policy Development

Successful autonomous robot operation requires a control algorithm to select actions based on the current state of the world. Traditional approaches to robot control model world dynamics, and derive a mathematically-based policy (Stefani et al., 2001). Though theoretically well-founded, these approaches depend heavily upon the accuracy of the dynamics model. Not only does the model require considerable expertise to develop, but approximations such as linearization are often introduced for computational tractability, thereby degrading performance. In other approaches the robot *learns* the control algorithm, through the use of machine learning techniques. One such approach learns control from executions of the target behavior, as demonstrated by a teacher.

2.1.1. Learning from Demonstration

Learning from Demonstration (LfD) is a technique for control algorithm development that learns a behavior from *examples*, or demonstrations, provided by a teacher. For our purposes, these examples are sequences of state-action pairs recorded during the teacher's demonstration of a desired robot behavior. Algorithms then utilize this dataset of examples to derive a *policy*, or mapping from world states to robot actions, that reproduces the demonstrated behavior. The learned policy constitutes a control algorithm for the behavior, and the robot uses this policy to select an action based on the observed world state.

Demonstration has the attractive feature of being an intuitive medium for human communication, as well as focusing the dataset to areas of the state-space actually encountered during behavior execution. Since it does not require expert knowledge of the system dynamics, demonstration also opens policy development to non-robotics-experts. Here we present a breif overview of LfD and its implementation within this thesis; a more thorough review of LfD is provided in Chapter 8.

2.1.1.1. Problem Statement. Formally, we define the world to consist of states S and actions A, with the mapping between states by way of actions being defined by the probabilistic transition function $T(s'|s,a) : S \times A \times S \rightarrow [0,1]$. We assume state to not be fully observable. The learner instead has access to observed state Z, through a mapping $S \rightarrow Z$. A teacher demonstration $d_j \in D$ is represented as n_j pairs of observations and actions such that $d_j = \{(\mathbf{z}_j^i, \mathbf{a}_j^i)\} \in D, \mathbf{z}_j^i \in Z, \mathbf{a}_j^i \in A, i = 0 \cdots n_j$. Within the typical LfD paradigm, the set D of these demonstrations is then provided to the learner. No distinction is made within D between the individual teacher executions however, and so for succinctness we adopt the notation $(\mathbf{z}_k, \mathbf{a}_k) \equiv \left(\mathbf{z}_j^i, \mathbf{a}_j^i\right)$. A policy $\pi : Z \rightarrow A$, that selects actions based on an observation of the current world state, or *query point*, is then derived from the dataset D.

A schematic showing demonstrated teacher executions, followed policy derivation from the resultant dataset D and subsequent learner policy executions, is shown in Figure 2.1. Dashed lines indicate repetitive flow, and therefore execution cycles performed multiple times.

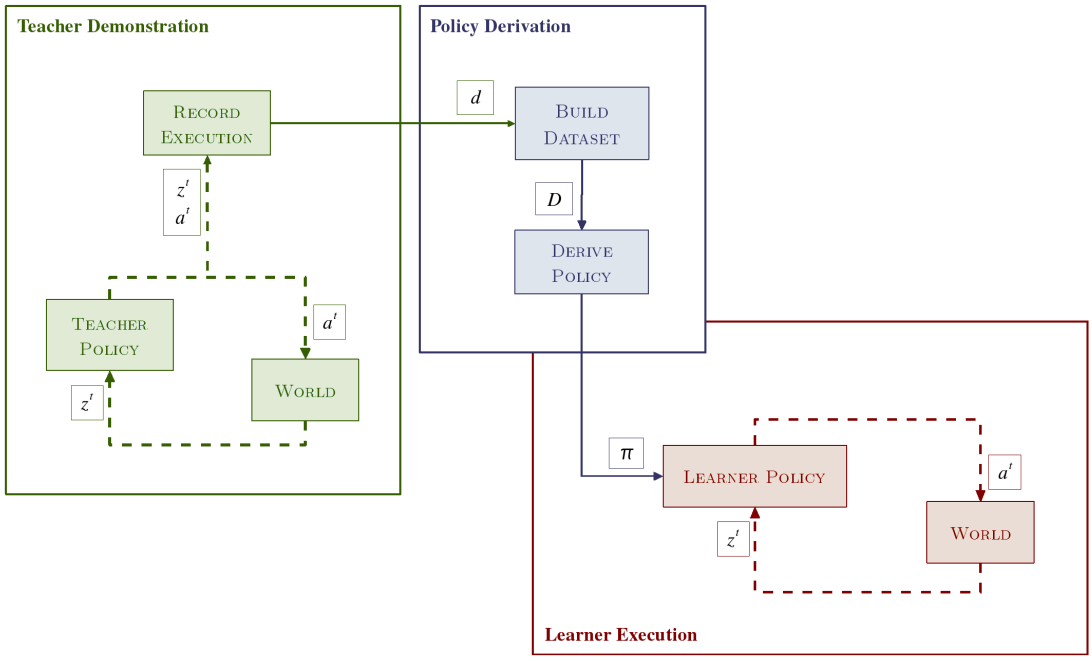

FIGURE 2.1. Learning from Demonstration control policy derivation and execution.

The LfD approach to obtaining a policy is in contrast to other techniques in which a policy is learned from *experience*, for example building a policy based on data acquired through exploration, as in *Reinforcement Learning (RL)* (Sutton and Barto, 1998). Also note that a policy derived under LfD is necessarily defined only in those states encountered, and for those actions taken, during the example executions.

2.1.1.2. Learning from Demonstration in this Thesis. The algorithms we contribute in this thesis learn policies within an LfD framework. There are many design decisions involved in the development of a LfD system, ranging from who executes a demonstration to how a policy is derived from the dataset examples. We discuss LfD design decisions in depth within Chapter 8. Here however we summarize the primary decisions made for the algorithms and empirical implementations of this thesis:

- A teleoperation demonstration approach is taken, as this minimizes correspondence issues and is reasonable on our robot platform.[1]

[1]Teleoperation is not necessarily reasonable for all robot platforms, for example those with high degrees of control freedom; it is reasonable, however, for the wheeled motion of our robot platform, the Segway RMP.

- In nearly all cases, the demonstration teacher is human.[2]
- The action space for all experimental domains is continuous, since the target application of this work is low-level motion control.
- Policies are derived via regression techniques, that use function approximation to reproduce the continuous-valued state-action mappings present in the demonstration dataset.

During teleoperation, a passive robot platform records from its sensors while being controlled by the demonstration teacher. Within our LfD implementations, therefore, the platform executing the demonstration is the passive robot learner, the teacher controlling the demonstration is human and the method of recording the demonstration data is to record directly off of the learner platform sensors. The issue of *correspondence* refers to differences in *embodiment*, i.e. sensing or motion capabilities between the teacher and learner. Correspondence issues complicate the transfer of teacher demonstrations to the robot learner, and therefore are a common source for limitations with LfD.

2.1.1.3. Policy Derivation with Regression. The empirical algorithm implementations of the following chapters accomplish policy derivation via function approximation, using regression techniques. A wealth of regression approaches exist, independently of the field of LfD, and many are compatible with the algorithms of this thesis. The reader is referred to Hastie et al. (2001) for a full review of regression.

Throughout this thesis, the regression technique we employ most frequently is a form of Locally Weighted Learning (Atkeson et al., 1997). Given observation $\mathbf{z}^t$, we predict action $\mathbf{a}^t$ through an averaging of datapoints in D. More specifically, the actions of the datapoints within D are weighted by a kernelized distance $\phi(\mathbf{z}^t, :)$ between their associated datapoint observations and the current observation $\mathbf{z}^t$. Thus,

$$\mathbf{a}^t = \sum_{(\mathbf{z}_i, \mathbf{a}_i) \in D} \phi\left(\mathbf{z}^t, \mathbf{z}_i\right) \cdot \mathbf{a}_i \tag{2.1}$$

$$\phi\left(\mathbf{z}^t, \mathbf{z}_i\right) = \frac{e^{\left(\mathbf{z}_i - \mathbf{z}^t\right)^T \Sigma^{-1} \left(\mathbf{z}_i - \mathbf{z}^t\right)}}{\sum_{\mathbf{z}_j \in D} e^{(\mathbf{z}_j - \mathbf{z}^t)^T \Sigma^{-1} (\mathbf{z}_j - \mathbf{z}^t)}}, \; \Sigma^{-1} = \begin{bmatrix} \sigma_0^2 & & \\ & \sigma_1^2 & \\ & & \ddots \\ & & & \sigma_m^2 \end{bmatrix} \tag{2.2}$$

where the weights $\phi(\mathbf{z}^t, :)$ are normalized over i and m is the dimensionality of the observation-space. In this work the distance computation is always Euclidean and the kernel Gaussian. The parameter Σ^{-1} is a constant diagonal matrix that scales each observation dimension and furthermore embeds the bandwidth of the Gaussian kernel. Details particular to the tuning of this parameter, in addition to any other regression techniques employed, will be noted throughout the document.

[2]For every experimental implementation in this thesis, save one, the teacher controlling the demonstration is a human; in the single exception (Ch. 3), the teacher is hand-written controller.

2.1.2. Dataset Limitations and Corrective Feedback

LfD systems are inherently linked to the information provided in the demonstration dataset. As a result, learner performance is heavily limited by the quality of this information. One common cause for poor learner performance is dataset sparsity, or the existence of areas of the state space in which no demonstration has been provided. A second cause is poor quality of the dataset examples, which can result from a teacher's inability to perform the task optimally or from poor correspondence between the teacher and learner.

A primary focus of this thesis is to develop policy refinement techniques that address common LfD dataset limitations, while being suitable for mobile robot motion control domains. The mobility of the robot expands the state space, making more prevalent the issue of dataset sparsity. Low-level motion control implies domains with continuous-valued actions, sampled at a high frequency. Furthermore, we are particularly interested in refinement techniques that provide *corrections* within these domains.

Our contributed LfD policy correction techniques do *not* rely on teacher demonstration to exhibit the corrected behavior. Some strengths of this approach include the following:

No need to recreate state: This is especially useful if the world states where demonstration is needed are dangerous (e.g. lead to a collision), or difficult to access (e.g. in the middle of a motion trajectory).

Not limited by the demonstrator: Corrections are not limited to the execution abilities of the demonstration teacher, who may be suboptimal.

Unconstrained by correspondence: Corrections are not constrained by physical differences between the teacher and learner.

Possible when demonstration is not: Further demonstration may actually be impossible (e.g. rover teleoperation over a 40 minute Earth-Mars communications lag).

Other novel feedback forms also are contributed in this thesis, in addition to corrections. These feedback forms also do not require state revisitation.

We formulate corrective feedback as a predefined list of corrections, termed *advice-operators*. Advice-operators enable the translation of a statically-defined high-level correction into a continuous-valued, execution-dependent, low-level correction; Chapter 4 presents advice-operators in detail. Furthermore, when combined with our techniques for *providing* feedback (presented in Section 2.3.2), a single piece of advice corrects multiple execution points. The selection of a *single advice-operator* thus translates into *multiple continuous-valued corrections*, and therefore is suitable for modifying low-level motion control actions sampled at high frequency.

2.2. Design Decisions for a Feedback Framework

There are many design decisions to consider in the development of a framework for providing teacher feedback. These range from the sort of information that is encoded in the feedback, to

how the feedback is incorporated into a policy update. More specifically, we identify the following considerations as key to the development of a feedback framework:

Feedback type: Defined by the amount of information feedback encodes and the level of granularity at which it is provided.

Feedback interface: Controls how feedback is provided, including the means of evaluating, and associating feedback with, a learner execution.

Feedback incorporation: Determines how feedback is incorporated into a policy update.

The follow sections discuss each of these considerations in greater detail.

2.2.1. Feedback Type

This section discusses the design decisions associated with the feedback type. Feedback is crucially defined both by the amount of information it encodes and the granularity at which it is provided.

2.2.1.1. Level of Detail. The purpose of feedback is to be a mechanism through which evaluations of learner performance translate into a policy update. Refinement that improves policy performance is the target result. Within the feedback details, some, none or all of this translation, from policy evaluation to policy update, can be encoded. The information-level of detail contained within the feedback thus spans a continuum, defined by two extremes.

At one extreme, the feedback provides very minimal information. In this case the majority of the work of policy refinement lies with the learner, who must translate this feedback (i) into a policy update (ii) that improves policy performance. For example, if the learner receives a single reward upon reaching a failure state, to translate this feedback into a policy update the learner must employ techniques like RL to incorporate the reward, and possibly additional techniques to penalize prior states for leading to the failure state. The complete translation of this feedback into a policy update that results in *improved* behavior is not attained until the learner determines through exploration an alternate, non-failure, state to visit instead.

At the opposite extreme, feedback provides very detailed information. In this case, the majority of the work of policy refinement is encoded in the feedback details, and virtually no translation is required for this to be meaningful as a policy update that also improves performance. For example, consider that the learner receives the value for an action-space gradient, along which a more desired action selection may be found. For a policy derived under a mapping function approximation approach, adjusting the function approximation to reproduce this gradient then constitutes both the feedback incorporation as well as the policy update that will produce a modified and improved behavior.

2.2.1.2. Feedback Forms. The potential forms taken by teacher feedback may differ according to many axes. Some examples of these axes include the source that provides the feedback, what triggers feedback being provided, and whether the feedback relates to states or actions. We consider one of the most important axes to be feedback *granularity*, defined here as the continuity

and frequency of the feedback. By *continuity*, we refer to whether discrete- versus continuous-valued feedback is given. By *frequency*, we refer to how frequently feedback is provided, which is determined by whether feedback is provided for entire executions or individual decision points, and the corresponding time duration between decision points.

In practice, policy execution consists of multiple phases, beginning with sensor readings being processed into state observations and ending with execution of a predicted action. We identify the key determining factor of feedback granularity as the policy *phase* at which the feedback will be applied, and the corresponding granularity of that phase.

Many different policy phases are candidates to receive performance feedback. For example, feedback could influence state observations by drawing attention to particular elements in the world, such as an object to be grasped or an obstacle to avoid. Another option could have feedback influence action selection, such as indicating an alternate action from a discrete set. As a higher-level example, feedback could indicate an alternate *policy* from a discrete set, if behavior execution consists of selecting between a hierarchy of underlying sub-policies.

The granularity of the policy phase receiving feedback determines the feedback granularity. For example, consider providing action corrections as feedback. Low-level actions for motion control tend to be continuous-valued and of short time duration (e.g. tenths or hundredths of a second). Correcting policy behavior in this case requires providing a continuous-valued action, or equivalently selecting an alternate action from an *infinite* set. Furthermore, since actions are sampled at high frequency, correcting an observed behavior likely translates to correcting *multiple* sequenced actions, and thus to multiple selections from this infinite set. By contrast, basic high-level actions and complex behavioral actions both generally derive from discrete sets and execute with longer time durations (e.g. tens of seconds or minutes). Correcting an observed behavior in this case requires selecting a *single* alternate action from a *discrete* set.

2.2.2. Feedback Interface

For teacher evaluations to translate into meaningful information for the learner to use in a policy update, some sort of teacher-student feedback interface must exist. The first consideration when developing such an interface is how to provide feedback; that is, the means through which the learner execution is evaluated and these evaluations are passed onto the learner. The second consideration is how the learner then associates these evaluations with the executed behavior.

2.2.2.1. How to Provide Feedback. The first step in providing feedback is to evaluate the learner execution, and from this to determine appropriate feedback. Many options are available for the evaluation of a learner execution, ranging from rewards automatically computed based on performance metrics, to corrections provided by a task expert. The options for evaluation are distinguished by the *source* of the evaluation, e.g. automatic computation or task expert, as well as the *information* required by the source to perform the evaluation. Different evaluation sources require varying amounts and types of information. For example, the automatic computation may require performance statistics like task success or efficiency, while the task expert may require

observing the full learner execution. The information required by the source additionally depends on the form of the feedback, e.g. reward or corrections, as discussed in Section 2.2.1.2.

After evaluation, the second step is to transfer the feedback to the learner. The transfer may or may not be immediate, for example if the learner itself directly observes a failure state versus if the correction of a teacher is passed over a network to the robot. How frequently feedback is incorporated and the policy updated is an additional consideration. Algorithms may receive and incorporate feedback online or in batch mode. At one extreme, streaming feedback is provided as the learner executes and immediately updates the policy. At the opposite extreme, feedback is provided post-execution, possibly after multiple executions, and updates the policy offline.

2.2.2.2. How to Associate with the Execution. A final design decision for the feedback interface is how to associate feedback with the underlying, now evaluated, execution. The mechanism of association varies, based on both the type and granularity of the feedback.

Some feedback forms need only be loosely tied to the execution data. For example, an *overall performance* measure is associated with the entire execution, and thus links to the data at a very coarse scale. This scale becomes finer, and association with the underlying data trickier, if this single performance measure is intended to be somehow distributed across only a portion of the execution states, rather than the execution as a whole; similar to the RL issue of reward back-propagation.

Other feedback forms are closely tied to the execution data. For example, an *action correction* must be strictly associated with the execution point that produced the action. Feedback and data that are closely tied are necessarily influenced by the sampling frequency of the policy. For example, actions that have significant time durations are easier to isolate as responsible for particular policy behavior, and thus as recipients of a correction. Such actions are therefore straightforward to properly associate with the underlying execution data. By contrast, actions sampled at high frequency are more difficult to isolate as responsible for policy behavior, and thus more difficult to properly associate with the execution data.

An additional consideration when associating with the execution data is whether feedback is offered online or post-execution. For feedback offered online, potential response lag from the feedback source must be accounted for. This lag may or may not be an issue, depending on the sampling rate of the policy. For example, consider a human feedback teacher who takes up to a second to provide action corrections. A one-second delay will not impact association with actions that last on the order of tens of second or minutes, but could result in incorrect association for actions that last fractions of a second. For feedback offered post-execution, sampling rate becomes less of an issue. However, now the execution may need to be somehow replayed or re-represented to the feedback provider, if the feedback is offered at a finer level than overall execution performance.

2.2.3. Feedback Incorporation

After determining the feedback type and interface, the final step in the development of a feedback framework is how to incorporate the feedback into a policy update. Incorporation depends both

on the type of feedback received, as well as the approach for policy derivation. How frequently feedback incorporation and policy updating occurs depends on whether the policy derivation approach is online or offline, as well as the frequency at which the evaluation source provides feedback.

Consider the following examples. For a policy that directly approximates the function mapping states to actions, corrective feedback could provide a gradient along which to adjust the function approximation. Incorporation then consists of modifying the function approximation to reproduce this gradient. For a policy that combines a state-action transition model with RL, state-crediting feedback could be used to change the state values, that are taken into account by the RL technique. For a policy represented as a plan, corrective feedback could modify an incorrectly learned association rule defined between an action and pre-condition, and correspondingly also a policy produced from a planner that uses these rules.

2.3. Our Feedback Framework

This thesis contributes *Focused Feedback for Mobile Robot Policies (F3MRP)* as a framework for providing feedback for the purposes of building and improving LfD motion control policies on a mobile robot. The F3MRP framework was developed within the GNU Octave scientific language (Eaton, 2002). In summary, F3MRP framework makes the following design decisions:

Feedback type: The types of feedback considered include binary performance flags and policy corrections.

Feedback interface: Evaluations are performed by a human teacher, who selects segments of the visually displayed learner execution for the purposes of data association.

Feedback incorporation: Feedback incorporation varies based on the feedback type. Techniques for the incorporation of feedback into more complex policies are also discussed.

Each of these design decisions are described in depth within the following sections.

2.3.1. Feedback Types

This section discusses the feedback types currently implemented within the F3MRP framework. First presented is the binary performance flag type, followed by corrective advice. Note that the F3MRP framework is flexible to many feedback forms, and is not restricted to those presented here.

2.3.1.1. Binary Performance Flags. The first type of feedback considered is a binary performance flag. This feedback provides a *binary* indication of whether policy performance in a particular area of the state-space is preferred or not. In the BC algorithm (Ch. 3), binary flags will provide an indication of *poor* policy performance. In the FPS (Ch. 6) and DWL (Ch. 7) algorithms, binary flags will provide an indication of *good* policy performance.

The level of detail provided by this simple feedback form is minimal; only an indication of poor/good performance quality is provided. Since the flags are binary, a notion of relative amount, or to what *extent* the performance is poor or good, is not provided. Regarding feedback granularity,

the space-continuity of this feedback is binary and therefore coarse, but the frequency at which the feedback is provided is high, since feedback is provided for a policy sampled at high frequency.

2.3.1.2. Corrective Advice. The second type of feedback considered is corrective advice. This feedback provides a *correction* on the state-action policy mapping. Corrections are provided via our contributed *advice-operator* interface. Advice-operators will be presented in depth within Chapter 4, and employed within the A-OPI (Ch. 5), FPS (Ch. 6) and DWL (Ch. 7) algorithms.

A higher level of detail is provided by this corrective feedback form. Beyond providing an indication of performance quality, an indication of the *preferred* state-action mapping is provided. Since advice-operators perform mathematical computations on the learner executed state/action values, the correction amounts are not static and do depend on the underlying execution data. Furthermore, advice-operators may be designed to impart a notion of relative amount. Such considerations are the choice of the designer, and advice-operator development is flexible. Details of the principled development approach to the design of advice-operators taken in this thesis will be provided in Chapter 4 (Sec. 4.2).

2.3.2. Feedback Interface

This section presents the teacher-student feedback interface of the F3MRP framework. Key to this interface is the anchoring of feedback to a visual presentation of the 2-D ground path taken during the mobile robot execution. This visual presentation and feedback anchoring is the mechanism for associating feedback with the execution under evaluation.

Performance evaluations under the F3MRP framework are performed by a human teacher. To perform the evaluation, the teacher observes the learner execution. The teacher then decides whether to provide feedback. If the teacher elects to provide feedback, he must indicate the type of feedback, i.e. binary performance flags or corrective advice, as well as an execution segment over which to apply the feedback.

2.3.2.1. Visual Presentation. The F3MRP framework graphically presents the 2-D path physically taken by the robot on the ground. This presentation furthermore provides a visual indication of data support during the execution. In detail, the presentation of the 2-D path employs a color scheme that indicates areas of weak and strong dataset support. Support is determined by how close a query point is to the demonstration data producing the action prediction; more specifically, by the distance between the query point and the single *nearest* dataset point contributing to the regression prediction.

Plot colors are set based on thresholds on dataset support, determined in the following manner. For a given dataset, the 1-NN Euclidean distance between points of the set are modelled as a Poisson distribution, parameterized by λ, with mean $\mu = \lambda$ and standard deviation $\sigma = \sqrt{\lambda}$. An example histogram of 1-NN distances within one of our datasets and the Poisson model approximating the distribution is shown in Figure 2.2. This distribution formulation was chosen since the distance calculations never fall below, and often cluster near, zero; behavior which is better modelled by a Poisson rather than Gaussian distribution.

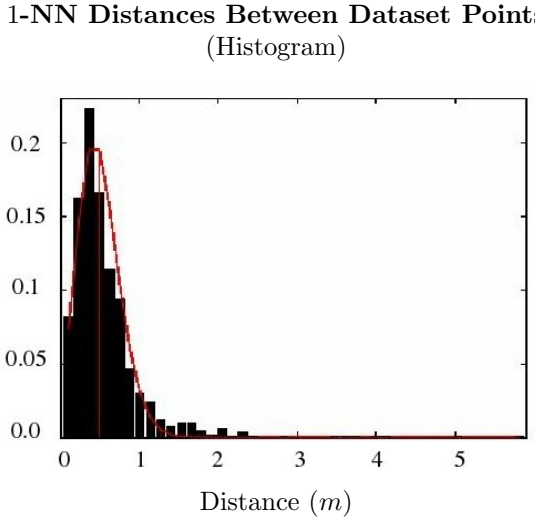

FIGURE 2.2. Example distribution of 1-NN distances within a demonstration dataset (black bars), and the Poisson model approximation (red curve).

The data support thresholds are then determined by the distribution standard deviation σ. For example, in Figure 2.3, given an execution query point q with 1-NN Euclidean distance ℓ_q to the demonstration set, plotted in black are the points for which $\ell_q < \mu + \sigma$, in dark blue those within $\mu + \sigma \leq \ell_q < \mu + 3\sigma$ and in light blue those for which $\mu + 3\sigma \leq \ell_q$.

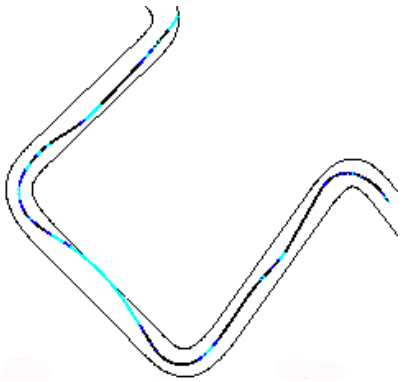

FIGURE 2.3. Example plot of the 2-D ground path of a learner execution, with color indications of dataset support (see text for details).

The data support information is used by the teacher as he sees fit. When the teacher uses the information to determine areas that are to receive the positive credit feedback, this technique effectively reinforces good learner behavior in areas of the state-space where there is a lack of data support. Predictions in such areas rely on the generalization ability of the regression technique used for policy derivation. Since the regression function approximation is constantly changing, due to the addition of new data as well as parameter tuning, there is no guarantee that the regression will continue to generalize in the same manner in an unsupported area. This is why adding examples of

good learner performance in these areas can be so important. To select these areas, the teacher relies on the visual presentation provided by the feedback interface. Without the visual depiction of data support, the teacher would have no way to distinguish unsupported from supported execution areas, and thus no way to isolate well-performing execution points lacking in data support and dependent on the regression generalization.

Note that early implementations of the F3MRP framework, employed in Chapters 3 and 5, relied exclusively on the teacher's visual observation of the learner performance. The full F3MRP framework, employed in Chapters 6 and 7, utilizes the data support feedback scheme just described.

2.3.2.2. Anchoring Feedback to the Execution. The types of feedback provided under the F3MRP framework associate closely with the underlying learner execution, since the feedback either credits or corrects specific execution points. To accomplish this close feedback-execution association, the teacher selects segments of the graphically displayed ground path taken by the mobile robot during execution. The F3MRP framework then associates this ground path segment with the corresponding segment of the observation-action trace from the learner execution. Segment sizes are determined dynamically by the teacher, and may range from a single point to all points in the trajectory. observation-action pairings for modification, by selecting segments of the displayed ground path, which the framework then associates with the observation-action trace of the policy.

A further challenge to address in motion control domains is the high frequency at which the policy is sampled. Feedback under the F3MRP framework requires the isolation of the execution points responsible for the behavior receiving feedback, and a high sampling rate makes this isolation more difficult and prone to inaccuracies. High sampling frequency thus complicates the above requirement of a tight association between feedback and execution data, which depends on the accurate isolation of points receiving feedback.

To address this complication, the F3MRP framework provides an interactive tagging mechanism, that allows the teacher to mark execution points as they display on the graphical depiction of the 2-D path. The tagging mechanism enables more accurate syncing between the teacher feedback and the learner execution points. For experiments with a simulated robot, the 2-D path is represented in real-time as the robot executes. For experiments with a real robot, the 2-D path is played back after the learner execution completes, to mitigate inaccuracies due to network lag.

2.3.3. Feedback Incorporation

The final step in the development of a feedback framework is how to incorporate feedback into a policy update. Feedback incorporation under the F3MRP framework varies based on feedback type. This section also discusses how feedback may be used to build more complex policies.

2.3.3.1. Incorporation Techniques. Feedback incorporation into the policy depends on the type of feedback provided. The feedback incorporation technique employed most commonly in this thesis proceeds as follows. The application of feedback over the selected learner observation-action execution points produces *new* data, which is added to the demonstration dataset. Incorporation into the policy is then as simple as rederiving the policy. For lazy learning techniques, like the Locally Weighted Averaging regression employed in this thesis, the policy is derived at execution

time based on a current query point; adding the data to the demonstration set thus constitutes the entire policy update.

Alternative approaches for feedback incorporation within the F3MRP framework are also possible. Our negative credit feedback is incorporated into the policy by modifying the treatment of demonstration data by the regression technique (Ch. 3); no new data in this case is produced. Consider also the similarity between producing a corrected datapoint via advice-operators and providing a gradient that corrects the function approximation at that point. For example, some regression techniques, such as margin-maximization approaches, employ a loss-function during the development of an approximating function. The difference between an executed and corrected datapoint could define the loss-function at that point, and then this loss value would be used to adjust the function approximation and thus update the policy.

2.3.3.2. Broadening the Scope of Policy Development. We view the incorporation of performance feedback to be an additional dimension along which policy development can occur. For example, *behavior shaping* may be accomplished through the use a variety of popular feedback forms. Simple feedback that credits performance, like state reward, provides a policy with a notion of how appropriately it is behaving in particular areas of the state-space. By encouraging or discouraging particular states or actions, the feedback shapes policy behavior. This shaping ability increases with richer feedback forms, that may influence behavior more strongly with more informative and directed feedback.

Feedback thus broadens the scope of policy development. In this thesis, we further investigate whether *novel* or *more complex* policy behavior may be produced as a result of feedback. In this case feedback enhances the *scalability* of a policy: feedback enables the policy to build into one that produces more complex behavior or accomplishes more complex tasks, that perhaps were difficult to develop using more traditional means such as hand-coding or demonstration alone. How to build feedback into policies so that they accomplish more complex tasks is by and large an area of open research.

Chapter 6 presents our algorithm (FPS) for explicit policy scaffolding with feedback. The algorithm operates by first developing multiple simple motion control policies, or behavior *primitives*, through demonstration and corrective feedback. The algorithm next builds a policy for a more complex task that has *not* been demonstrated. This policy is built on top of the motion primitives. The demonstration datasets of the primitive policies are assumed to occupy relatively distinct areas of the state-space. The more complex policy is not assumed to restrict itself to any state-space area. Given these assumptions, the algorithm automatically determines when to select and execute each primitive policy, and thus how to scaffold the primitives into the behavior of the more complex policy. Teacher feedback additionally is used to assist with this scaffolding. In the case of this approach, feedback is used both to develop the motion primitives and to assist their scaffolding into a more complex task.

Chapter 7 presents our algorithm (DWL) that derives a separate policy for each feedback type and also for distinct demonstration teachers, and employs a performance-based weighting scheme to select between policies at execution time. As a result, the more complex policy selects between

the multiple smaller policies, including all feedback-derived policies as well as any demonstration-derived policies. Unlike the policy scaffolding algorithm, in this formulation the different policy datasets are *not* assumed to occupy distinct areas of the state space. Policy selection must therefore be accomplished through other means, and the algorithm weights the multiple policies for selection based on their relative past performance. In the case of this approach, feedback is used to populate novel datasets and thus produce multiple policies distinct from the demonstration policy. All policies are intended to accomplish the full task, in contrast to the multiple primitive behavior policies of the scaffolding algorithm.

2.3.4. Future Directions

Here we identify future directions for the development of the F3MRP framework. In particular, we consider the adaptation of this framework to other domains, such as those with non-mobile robots or long-enduring actions.

While the F3MRP framework was designed specifically for mobile robot applications, the techniques of the framework could apply to non-mobile robots through the development of an alternative to the visual representation of the 2-D ground path. Some other mechanism would be required for the selection of execution points by the teacher, necessary for the association between feedback and the learner execution. For example, to visually display the 3-D spatial path taken by the end effector of a robotic arm could serve as a suitable alternative; other options might forgo a visual representation altogether.

Teacher feedback is offered in this framework in a semi-online, or multi-batch, approach. In the case of the work in this thesis, since the policy is sampled at such a high frequency ($0.0\bar{3}3s$), the teacher does not have time to provide feedback before the next action executes. Feedback is therefore provided at the conclusion of a learner execution. In theory, feedback could be provided fully online, during the execution. In domains where actions have a longer duration, such online feedback would be possible. Feedback *provided* during execution also could be *incorporated* during execution, as long as the chosen policy derivation technique were an online approach.

2.4. Our Baseline Feedback Algorithm

We now detail the baseline feedback algorithm of this thesis. This algorithm is incorporated, in some form, into each of the algorithms presented later in the document.

2.4.1. Algorithm Overview

Our feedback algorithm operates in two phases. During the *demonstration phase*, a set of teacher demonstrations is provided to the learner. From this the learner generalizes an initial policy. During the *feedback phase*, the learner executes with this initial policy. Feedback on the learner execution is offered by a human teacher, and is used by the learner to update its policy. The learner then executes with the updated policy, and the *execute-feedback-update* cycle continues to the satisfaction of the teacher.

Figure 2.4 presents a schematic of this approach. Within this schematic, dashed lines indicate repetitive flow and therefore execution cycles that are performed multiple times. In comparison to

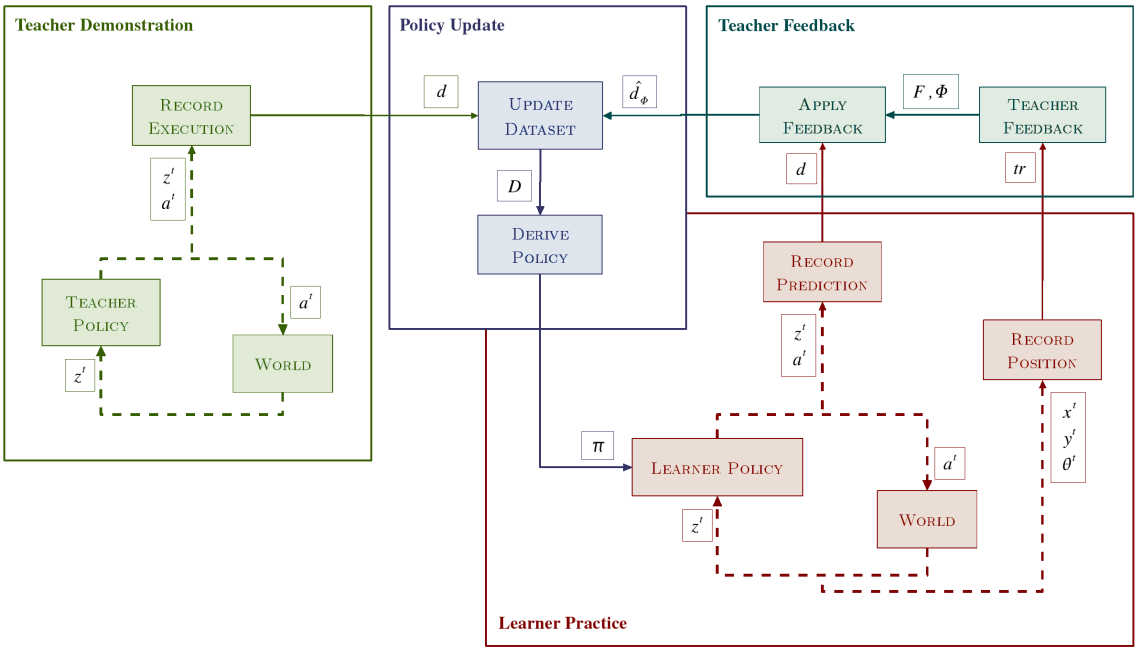

FIGURE 2.4. Policy derivation and execution under the general teacher feedback algorithm.

the generic LfD schematic of Figure 2.1, a box for *Teacher Feedback* has now been added. The demonstration phase of the feedback algorithm is represented by the *Teacher Demonstration* box, and the feedback phase by the *Learner Execution*, *Teacher Feedback* and *Policy Update* boxes. Similar schematics will be presented for each algorithm throughout the document; note however that to mitigate repetition and clutter the teacher demonstration phase will be omitted from all future schematics (and pseudo-code).

2.4.2. Algorithm Execution

Psuedo-code for the baseline feedback algorithm provided in Algorithm 1. The first phase of this algorithm consists of teacher demonstration (lines 1-8), during which example observation-action pairs are recorded. At each timestep, the teacher selects an action $\mathbf{a}^t$ (lines 4). This action is executed, and recorded along with the observed state of the world at this timestep $\mathbf{z}^t$, into the demonstration dataset D (lines 5-6). This process continues until the teacher has completed the demonstration of the target behavior. The teacher may choose to provide multiple demonstrations, should he desire. The full set D of these demonstrations are provided to the learner.

The second phase of policy development consists of learner practice. To begin, an initial policy π is derived from the set of teacher demonstrations (line 9). A single practice run (lines 10-21) consists of a single *execution-feedback-update* cycle; that is, of learner execution followed by teacher feedback and a policy update. A subsequent practice run is then initiated, during which the learner will execute with this new, updated policy. Practice runs continue until the teacher is satisfied with learner performance, and consequently with the developed policy.

Algorithm 1 Baseline Feedback Algorithm

1: *initialize* $D \leftarrow \{\}$
2: **while** *demonstrating* **do**
3: **repeat**
4: *select* $\mathbf{a}^t$
5: *execute* $\mathbf{a}^t$
6: *record* $D \leftarrow D \cup (\mathbf{z}^t, \mathbf{a}^t)$
7: **until** done
8: **end while**
9: *initialize* $\pi \leftarrow$ `policyDerivation`(D)
10: **while** *practicing* **do**
11: *initialize* $d \leftarrow \{\}$, $tr \leftarrow \{\}$
12: **repeat**
13: *predict* $\mathbf{a}^t \leftarrow \pi(\mathbf{z}^t)$
14: *execute* $\mathbf{a}^t$
15: *record* $d \leftarrow d \cup (\mathbf{z}^t, \mathbf{a}^t)$, $tr \leftarrow tr \cup (x^t, y^t, \theta^t)$
16: **until** done
17: *advise* $\{ F, \Phi \} \leftarrow$ `teacherFeedback`$(\ tr\ )$
18: *apply* $\hat{d}_\Phi \leftarrow$ `applyFeedback`$(\ F, \Phi, d\ )$
19: *update* $D \leftarrow$ `datasetUpdate`$(\ D, \hat{d}_\Phi\ )$
20: *rederive* $\pi \leftarrow$ `policyDerivation`$(\ D\ )$
21: **end while**
22: **return** π

During the learner execution portion of a practice run (lines 12-16), the learner first executes the task. At each timestep the learner observes the world, and predicts action $\mathbf{a}^t$ according to policy π (line 13). This action is executed and recorded in the prediction trace d, along with observation $\mathbf{z}^t$ (line 15). The information recorded in the trace d will be incorporated into the policy update. The global position x^t, y^t and heading θ^t of the mobile robot is additionally recorded, into the position trace tr. Information recorded into tr will be used by the F3MRP framework, when visually presenting the path taken by the robot on the ground during execution.

During the teacher feedback and policy update portion of a practice run (lines 17-20), the teacher provides feedback based on her own observations of the learner execution performance. Using the visual presentation of tr provided by the F3MRP feedback interface, the teacher indicates a segment Φ of the learner execution, along with feedback F for that segment. The learner applies the information from the teacher (F, Φ) to the recorded prediction trace d, producing data $\hat{d}_\Phi$ (line 18). This data is used to update the dataset D (line 19). The exact details of how the teacher information is applied to the prediction trace d, and how the subsequent update to dataset D occurs, are particular to each feedback type and will be discussed within the presentation of each algorithm individually. The final step is the learner update of its policy π, by rederiving the policy from the feedback-modified dataset D (line 20).

2.4.3. Requirements and Optimization

Optimization under our feedback algorithm happens according to metrics used by the teacher when providing feedback. These metrics reflect characteristics of the behavior execution that the

teacher deems important to have present in the final policy. The metrics may be concrete and straightforward to evaluate, for example successful task completion, or more intuitive, such as smooth motion trajectories. The algorithm does not require explicit, formal, delineation of the metrics by the teacher, which has the benefit of allowing for evaluations guided by human intuition. In contrast to traditional reward as in RL, the metrics may allow for evaluations that are sufficiently complex or subtle to preclude modeling with a simple function. A drawback to such intuitive metrics, however, and that the optimization criteria are never explicitly defined. Since the criteria exist solely in the mind of the teacher, their satisfaction similarly is determined exclusively according to her discretion.

The teacher-employed metrics direct policy development, by guiding the teacher in his determination of (i) which portions of a learner execution represent aspects of the policy in need of improvement and (ii) what sort of feedback will bring about the intended improvement. It is a requirement that the teacher be able to identify poor policy performance. Presumably a teacher able to demonstrate the target behavior has at the very least an idea of what successful behavior looks like when *he* executes the task. It is not required that the feedback and demonstration teachers be the same person (indeed, the demonstration teacher does not even need to be a person), or even that the feedback teacher be able to execute the task. All that is required is that the feedback teacher is able to identify areas of an execution that correspond to suboptimal behavior. The actual association of these areas with the underlying policy is accomplished through the feedback association paradigm of the F3MRP framework.

The next requirement is that the teacher be able to identify a feedback type that will improve the poor policy performance. This step depends on the existence of a set of defined feedback types, that are effective at modifying the policy according to those execution characteristics deemed by the feedback teacher to be important. The identification of effective feedback types is largely up to the skill and creativity of the designer. This thesis introduces multiple novel feedback forms, and there is no limit on the number of other formulations possible.

The primary requirement for any feedback type is a clear definition of how it affects a policy update. The feedback forms of this thesis update the policy either by modifying the use of the existing demonstration data, or by adding new data to the demonstration set. The first approach requires a regression technique that is transparent with regards to which datapoints were responsible for a given regression prediction. The second approach has no requirements on the employed regression technique. Policy updates need not be restricted to these two manners of feedback incorporation, however; in fact one example of an alternative formulation for feedback incorporation will be proposed in Chapter 4 (Sec. 4.4.2.1). A specific feedback form may or may not have further requirements for its development. For example, in Chapter 4 the additional requirements specific to the development of advice-operators are detailed (Sec. 4.1.2).

A common denominator to the quality of any policies derived under our approach is the soundness of the regression technique employed for policy derivation. The majority of our feedback types add new data to the demonstration dataset. The behavior encoded within this data will be represented by the policy only assuming accurate regression techniques. This dependence is present for any policies derived under LfD techniques in general, however, and is no stronger for our feedback-derived data than for teacher-demonstrated data.

2.5. Summary

The Learning from Demonstration (LfD) framework of this thesis has been introduced in this chapter. In particular, the example gathering approach of teleoperation, and the policy derivation approach of approximating the mapping function, will be employed in all implementations of LfD throughout this work. Potential limitations present in the LfD dataset were detailed, along with motivation for the most noteworthy of our contributed feedback types. Two key benefits of corrective feedback were identified as the following: states do not need to be revisited to receive feedback, which is useful when states are difficult or dangerous to reach, and feedback is not provided through more teacher demonstrations, which is useful when the teacher is a suboptimal demonstrator or the teacher-to-learner correspondence is poor.

Human feedback, that builds and refines motion control policies for mobile robots, is the cornerstone of this thesis. An in-depth evaluation of the requirements associated with providing feedback has been presented in this chapter. Key design decisions were identified to include the type of feedback provided, the feedback interface and the manner of feedback incorporation into a policy update.

Our feedback framework, Focused Feedback for Mobile Robot Policies (F3MRP), was contributed in this chapter. The F3MRP framework enables information from teacher evaluations of a learner execution to be transferred to the learner, and incorporated into a policy update. The feedback types implemented in F3MRP include binary indications of good performance and corrective advice provided through advice-operators. To associate feedback with the execution, the teacher selects segments of a graphically represented execution trace. The feedback interface provides the human teacher with an indication of dataset support, which the teacher uses when selecting execution segments to receive good performance flags. The manner of feedback incorporation varies, based on the feedback type.

In conclusion, our baseline feedback algorithm was presented. The algorithm provides a base for all algorithms contributed and introduced throughout this work. The execution of this algorithm was detailed, along with a discussion of its optimization and implementation requirements.

CHAPTER 3

Policy Improvement with Binary Feedback

Most LfD approaches place the majority of the policy learning burden with the robot. One way the teacher can help shoulder some of this burden is by commenting on learner performance. While humans generally have less intuition about assigning credit to an underlying algorithm, they are good at assigning credit to overall performance. In the *Binary Critiquing (BC)* algorithm, a human teacher provides flags that indicate areas of poor performance in learner executions (Argall et al., 2007a). The teacher critiques learner performance of the task, by indicating areas of poor performance. Since the teacher critiques performance exclusively, she does not need to guess at what appropriate feedback for the underlying *causes* of the poor performance, such as suboptimal teacher demonstrations or ill-suited policy derivation techniques.

In this chapter we contribute Binary Critiquing as an algorithm for learning a robot control policy in which the teacher provides both task demonstrations and performance feedback. The approach is validated in a realistic simulation of a differential drive robot performing a motion interception task. The results show that this form of human teacher feedback does improve task performance, as measured by interception success and efficiency. Furthermore, through critiquing the robot performance comes to *exceed* the performance of the demonstration teacher.

The following section presents the BC algorithm, including the details of binary crediting feedback and overall algorithm execution. Section 3.2 presents the experimental implementation of BC. The motion interception task and domain are presented, and empirical results are discussed. The conclusions of this work are presented in Section 3.3, along with directions for future research with this algorithm.

3.1. Algorithm: Binary Critiquing

We present in this section the Binary Critiquing algorithm. A discussion of how the negative feedback is incorporated into the policy derivation, followed by details of algorithm execution under this paradigm, are provided.

3.1.1. Crediting Behavior with Binary Feedback

Teacher feedback under the BC algorithm consists of poor performance flags. This feedback is used by the algorithm to *credit* the underlying demonstration data. The credit attached to each datapoint is then considered during the policy derivation.

To credit the underlying demonstration data requires knowledge of which datapoints were involved in a particular prediction. The incorporation of teacher feedback, and its influence on the policy update, therefore is dependent on the regression approach used. For example, an approach which no longer directly uses the underlying demonstration data at execution time, such as *Neural Networks*, would not be appropriate. By contrast, *Lazy Learning* techniques (Atkeson et al., 1997) are particularly appropriate, as they keep around all of the training data and perform function approximation only at execution time, in response to a current observation query point. The simplest of these is *k-Nearest Neighbors (k-NN)*, which is here employed ($k = 1$) due to how straightforward it is to determine which datapoints were involved in a given prediction. At execution time, 1-NN determines the closest datapoint within the dataset, according to some distance metric, and its associated action is selected for execution.

Feedback is incorporated into the policy through an internal data representation. This representation is constructed by associating a scaling factor m_i with each training set observation-action pair $(\mathbf{z}_i, \mathbf{a}_i) \in D$. This factor scales the distance computation during the k-NN prediction. Formally, given a current observation $\mathbf{z}^t$, the scaled distance to each observation point $\mathbf{z}_i \in D$ is computed, according to some metric. The minimum is determined, and the action in D associated with this minimum

$$\mathbf{a}^t \;=\; \mathbf{a}_{\arg\min_i (\mathbf{z}^t - \mathbf{z}_i)^T \Sigma^{-1} (\mathbf{z}^t - \mathbf{z}_i) m_i}, \;\; (\mathbf{z}_i, \mathbf{a}_i) \in D \tag{3.1}$$

is executed. In our implementation the distance computation is Euclidean and observation dimensions are scaled inversely with dataset range within the diagonal matrix Σ^{-1}. The values m_i are initially set to 1.

Figure 3.1 presents a schematic of this approach. Unlike the other algorithms of this thesis, including the baseline feedback algorithm (Fig. 2.4), here only metrics used to compute the policy prediction are recorded, and not the executed observation-action pairs. The reason is due to the manner of feeedback incorporation into the policy. To incorporate feedback, the scaling factor associated with the dataset point that provided the action prediction is updated, as shown within the *Update Dataset* box within the schematic. Only the scaling factor, and not the observation-action pair, therefore needs to be recorded.

3.1.2. Algorithm Execution

Algorithm 2 presents pseudo-code for the BC algorithm. This algorithm is founded on the baseline feedback algorithm of Chapter 2 (Alg. 1). The distinguishing details of BC lie in the type and application of teacher feedback (Alg. 1, lines 17-18).

The teacher demonstration phase produces an initial dataset D, and since this is identical to the demonstration phase of the baseline feedback algorithm (Alg. 1, lines 1-8), for brevity the details of teacher execution are omitted here. The learner execution portion of the practice phase also proceeds

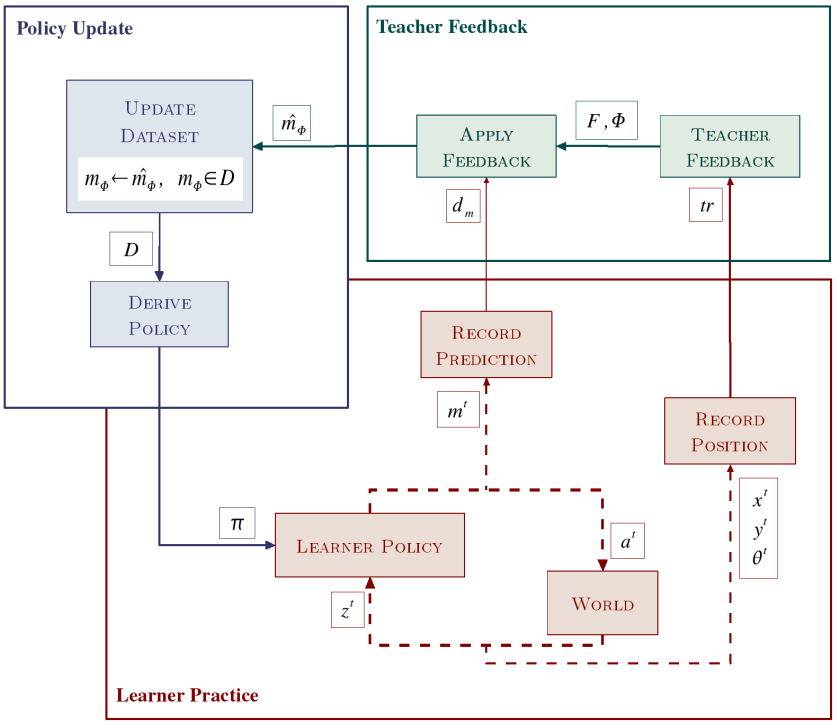

FIGURE 3.1. Policy derivation and execution under the Binary Critiquing algorithm.

similarly the baseline algorithm. The difference lies in what is recorded into the prediction trace d_m; here the scaling factor $m^t = m_i$ of the dataset point that provided the prediction is recorded, along with its distance to the query point $\ell^t \equiv (\mathbf{z}^t - \mathbf{z}_i)^T \Sigma^{-1} (\mathbf{z}^t - \mathbf{z}_i)$ (line 8).

Teacher feedback (line 10) under the BC algorithm proceeds as follows. For each learner execution, the teacher selects segment $\Phi, d_\Phi \subseteq d$, of the trajectory to flag as poorly performing. The teacher feedback is a negative credit $c \equiv \digamma$ (Sec. 2.4.2), a binary flag indicating poor performance. This credit will be associated with all selected points.

The teacher feedback is then applied across all data recorded in d_m and within the indicated subset Φ (line 12). For each selected execution point in d_m, indexed as $\varphi \in \Phi$, the value m^φ of this point is increased according to line 12, where $\kappa > 0$ is some empirically determined constant (here $\kappa = 0.1$). The amount by which m^φ is increased is scaled inversely with distance ℓ^φ so that points are not unjustly penalized if, due to sparsity in D, they provide recommendations in distant areas of the observation space which they do not support. To update m^φ in this manner means that datapoints whose recommendations gave poor results (according to the critique of the teacher) will be seen as further away during subsequent k-NN distance calculations.

These details of the dataset update are the real key to critique incorporation. To then incorporate the teacher feedback into the learner policy simply consists of rederiving the policy from the dataset D, now containing the updated values of m (line 15).

Algorithm 2 *Binary Critiquing*

Given **D**
initialize $\pi \leftarrow$ `policyDerivation`(D)
while *practicing* **do**
  initialize $d \leftarrow \{\},\ tr \leftarrow \{\}$
  repeat
    predict $\{\ \mathbf{a}^t, \ell^t, m^t\ \} \leftarrow \pi\left(\mathbf{z}^t\right)$
    execute $\mathbf{a}^t$
    record $d_m \leftarrow d_m \cup \left(\ell^t, m^t\right),\ tr \leftarrow tr \cup \left(x^t, y^t, \theta^t\right)$
  until done
  advise $\{\ c, \Phi\ \} \leftarrow$ `teacherFeedback`$(\ tr\ )$
  for all $\varphi \in \Phi,\ (\ell^\varphi, m^\varphi) \in d_m$ **do**
    apply $\hat{m}^\varphi \leftarrow m^\varphi + \frac{\kappa}{\ell^\varphi}$
    replace $m^\varphi \leftarrow \hat{m}^\varphi,\ m^\varphi \in D$
  end for
  rederive $\pi \leftarrow$ `policyDerivation`(D)
end while
return π

3.2. Empirical Implementation

This section presents the details and results of applying the BC algorithm to a simulated motion interception task. Policy performance was found to improve with critiquing, and furthermore to exceed the performance of the demonstration policy (Argall et al., 2007a).

3.2.1. Experimental Setup

Empirical validation of the BC algorithm is performed within a simulated motion interception domain. First presented are the task and domain, followed by the details of demonstration and critiquing, and then the evaluation measures.

3.2.1.1. Simulated Motion Interception Task and Domain. Within these experiments, a simulated differential drive robot is tasked with intercepting a moving ball (Fig. 3.2). Since a differential drive robot may only drive forward or turn, and cannot go sideways, interception of a moving object becomes a difficult task. The task also depends heavily on the dynamics of the world, and so writing an appropriate control policy can be challenging in the absence of accurate predictive world motion models.

Care was taken to keep the simulated domain realistic to the real world domain of a Segway RMP robot (introduced in later chapters). The robot is assumed to use a differential-drive mechanism, with a low-level controller that takes desired robot speeds as input commands[1]. For the robot, motion is propagated by simple differential drive simulation of the global robot position

$$\begin{aligned} x^{t+1} &= x^t + v^t \cos(\theta^t) \\ y^{t+1} &= y^t + v^t \sin(\theta^t) \\ \theta^{t+1} &= \theta^t + \omega^t \cdot dt \end{aligned} \tag{3.2}$$

[1]While other arrangements are possible, this approach is common to many robot platforms.

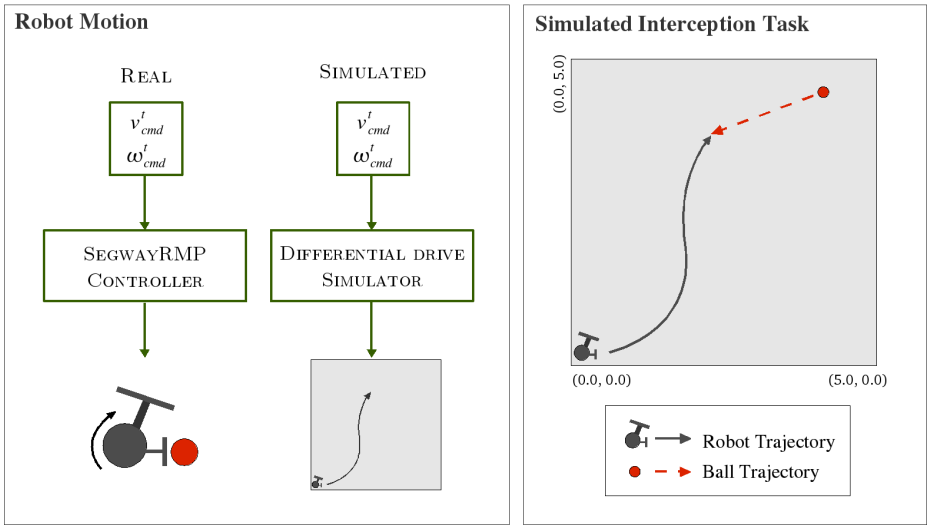

FIGURE 3.2. Simulated robot motion (left) and ball interception task (right).

where (x, y) is the global robot location and θ the global orientation. Dynamic limitations on the performance of the robot constrain its linear and rotational speeds ($v \in [0.0, 2.0]\frac{m}{s}$, $\omega \in [-3.0, 3.0]\frac{rad}{s}$) and accelerations ($\dot{v} \in [0.0, 3.0]\frac{m}{s}$, $\dot{\omega} \in [-7.0, 7.0]\frac{rad}{s}$). For the ball, motion is propagated by a constant velocity model with a simple exponential decay on the initial ball velocity to mimic frictional loss,

$$\begin{aligned} x_B^{t+1} &= x_B^t + \gamma^t \dot{x}_B^t dt \\ y_B^{t+1} &= y_B^t + \gamma^t \dot{y}_B^t dt \end{aligned} \tag{3.3}$$

where $\gamma \in [0, 1]$ is the decay constant (here $\gamma = 0.99$). Initial ball velocity is limited ($\|\dot{x}_B^0, \dot{y}_B^0\| \in [-2.5, 0.0]\frac{m}{s}$). The positions generated within the domain are bounded above and below ($x, y \in [0.0, 5.0]m$), constraining both the ball and the robot locations.

3.2.1.2. Demonstrations, Observations and Learner Practice. Teacher demonstrations are performed via teleoperation of the robot by a hand-written suboptimal controller able to select changes in rotational and translational speed. This teacher was chosen for two reasons: the first being to highlight the ability of this algorithm to improve upon teacher suboptimality, and the second because of the ease with which a large number of demonstrations could be provided (here 100). For teacher demonstrations, initial world configurations are uniformly sampled from the range of world locations $(x, y) \sim U(0.0, 5.0m)$ and ball speeds $(\dot{x}, \dot{y}) \sim U\left(-2.5, 0.0\frac{m}{s}\right)$.

Learner policy executions begin from an initial world configuration of robot-relative ball position and velocity. Execution ends when the robot either intercepts the ball or the ball travels out of bounds. During execution the robot may directly observe its own position and the ball position, as shown in Figure 3.3. Let (d_b^t, ϕ_b^t) be the distance and angle to the ball in the robot-relative frame, and (d^t, ϕ^t) the distance traveled by and heading of the robot within the global world frame. The observations computed by the robot are 6-dimensional: $[d_b^t, \phi_b^t, (d_b^t - d_b^{t-1}), (\phi_b^t - \phi_b^{t-1}), (d^t -$

$d^{t-1}), (\phi^t - \phi^{t-1})]$. The actions predicted by the robot are 2-dimensional: change in rotational speed (Δv) and change in translational speed ($\Delta \omega$).

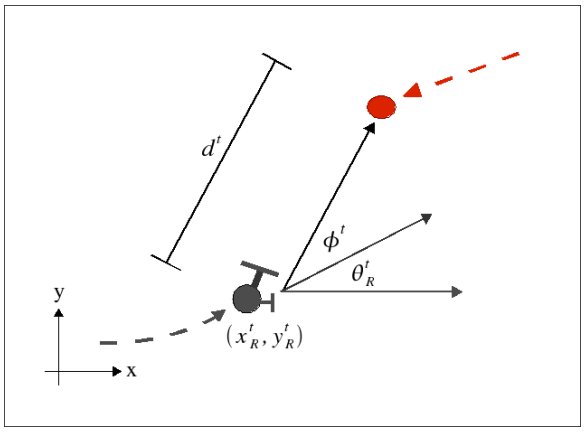

FIGURE 3.3. World state observations for the motion interception task.

An example cycle of learner execution and teacher critique, along with the subsequent improvement in learner execution, is demonstrated in Figure 3.4. Blue and red lines mark respectively the robot and ball execution paths on the ground plane, arrow heads indicate direction of travel, and the red circle is a distance threshold on successful interception. In this example the robot does initially intercept the ball, but the loop in its trajectory is inefficient (A). The teacher critiques this trajectory, flagging the loop as an area of poor performance (B). The robot repeats the execution, now without the inefficient loop (C).

3.2.1.3. Evaluation. To measure the performance of this algorithm, trajectory executions are evaluated for success and efficiency. A successful interception is defined by (a) the relative distance to the ball falling below some threshold ϵ and (b) the ball and robot both remaining within bounds ($\epsilon = 0.01m$). Efficiency is measured by execution duration, with trajectories of shorter duration being considered more efficient.

Performance evaluations occur on an independent test set containing n_t initial world conditions randomly sampled from a uniform distribution within the bounds of the training set ($n_t = 100$). A *practice round* is defined as execution from a single randomly sampled initial world condition (not found within the test set), followed by teacher feedback and optional re-execution, at the teacher's discretion. To mark learner progress, execution on the test set is carried out after every n_p practice rounds ($n_p = 20$, 120 rounds in total). Practice rounds conclude when no further performance improvement is observed on the test set. Note that during the test evaluations, the learner executes using its most recent policy π, and *no* teacher critiquing or policy updating occurs. For comparative purposes, the performance of the demonstration policy on this test set is evaluated as well.

3.2.2. Results

In this section we show learner performance to improve with critiquing. This performance improvement is demonstrated through an increase in interception successes, as well as more efficient

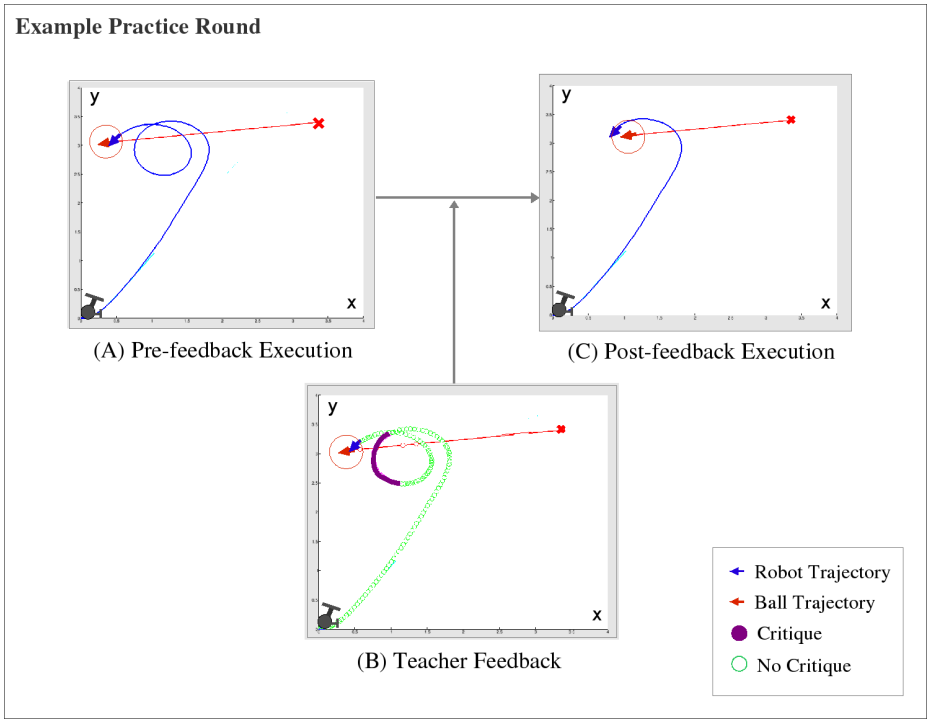

FIGURE 3.4. Example practice round, where execution efficiency improves with critiquing.

executions of successful trajectories. On both of these measures, learner performance not only improves, but comes to exceed the performance of its teacher.

3.2.2.1. Success Improvement. Improvement with teacher critiquing was seen under validation by an independent test set between practice rounds. Figure 3.5 shows learner improvement, where each datapoint represents the percent success result of executing from all test set initial conditions (solid line). For comparison, the performance of the demonstration policy on the test set is also provided (dashed line).

Learner percent success also improved *within* the individual practice rounds, shown within Table 3.1. The first column indicates the subset of practice rounds under consideration. The second column shows the result of executing from each initial world configuration in a given subset, using the policy derived *before* these executions receive any teacher feedback. The third column shows re-execution from the same initial world configurations, but now using the policy derived *after* teacher feedback on the practice round subset. The average of all rounds is shown in the bottom row. The higher average percent success post-feedback, compared to the pre-feedback average, verifies that performance improved within the practice rounds. Note that this post-feedback practice set average (74.2%) also exceeds the final test set performance (70.0%); this is equivalent to lower training versus test set error.

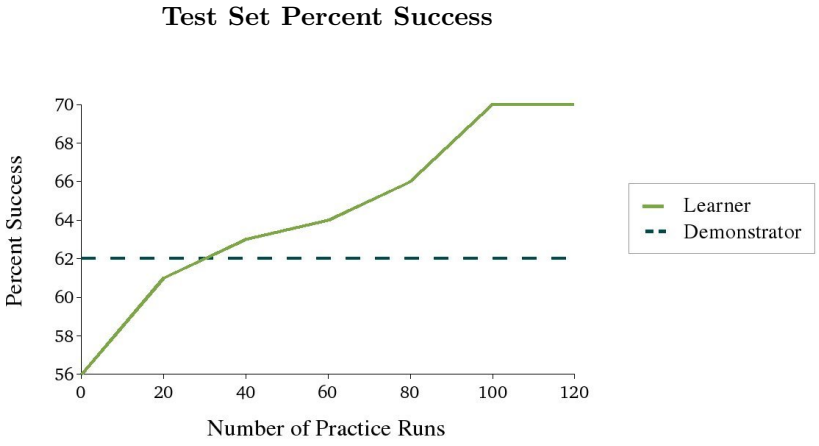

FIGURE 3.5. Improvement in successful interceptions, test set.

Practice Round	Success, Pre-feedback π (%)	Success, Post-feedback π (%)
1-20	65	80
21-40	60	75
41-60	50	55
61-80	50	60
81-100	65	90
101-120	75	85
All	**60.83**	**74.17**

TABLE 3.1. Pre- and post-feedback interception percent success, practice set.

3.2.2.2. Efficiency Improvement. Negative feedback was also found to improve learner execution efficiency. That is, the robot learned to intercept the ball faster, indicated by reduced execution times. Efficiency results on the independent test set, from the learner executing with its initial and final policies, are presented in Table 3.2. Note that to decouple this measure from success, only those runs in which *all* policies were successful are compared (since the out-of-bounds success measure otherwise taints the comparison, given that from identical starting conditions a successful interception is necessarily faster than a unsuccessful one). Again, for comparison, results from the demonstration policy test set executions are also provided.

	Learner Initial π	Learner Final π	Teacher
Success %	56	**70**	62
Execution Time (s)	2.73	**1.96**	2.87

TABLE 3.2. Interception task success and efficiency, test set.

3.3. Discussion

A discussion of the BC algorithm and the above empirical implementation is presented in this section. To begin, the conclusions of this work are detailed, followed by a discussion of future directions for the BC algorithm.

3.3.1. Conclusions

This section highlights some noteworthy conclusions which may be drawn from this empirical implementation of the BC algorithm. First discussed is the unexpected extent of policy improvement with critiquing, which enabled performance beyond that of the demonstration teacher. Next insights into the benefit of providing critique feedback with a human teacher are discussed.

3.3.1.1. Improvement Beyond Teacher. The empirical results show that the learner policy, through the incorporation of teacher feedback, was able to perform *better* than the demonstrator (Fig. 3.5, Tbl. 3.2). These results underline the benefits of using the BC algorithm in this experimental setup. The hand-coded demonstration controller was *not* optimal for the domain. By critiquing the robot's executions, the algorithm was able to correct for some demonstration error, thus improving the robot's performance beyond the capabilities of the demonstration teacher, and all in a simple and straightforward manner. The BC approach thus is shown to address the LfD limitation of suboptimal teacher demonstrations.

3.3.1.2. Providing Critique Feedback. Feedback provided in this implementation of the BC algorithm depended on human-expertise. The evaluation criteria used by the feedback teacher was a combination of ball interception success and execution efficiency. To define metrics that capture these criteria is straightforward, for example those defined in Section 3.2.1.3 for analytical purposes. Were only such simple metrics provided in the feedback, a critiquing teacher could easily be automated.

A critique does not simply provide an overall measure of execution performance, however. Critiques additionally indicate areas that *contributed* to the poor values of these evaluation metrics. For example, consider the execution of Figure 3.4. Being able to assess that an execution is on the whole inefficient, versus being able to determine which portions of the execution *caused* the inefficiency, are two very different tasks.

In contrast to overall task performance, to formally define a metric for *credit assignment*, that indicates source areas for poor performance, can be quite difficult. This challenge is similar to the much explored issue of *reward back-propagation* within the field of Reinforcement Learning. The BC algorithm circumvents this issue by exploiting a *human* for the task of credit assignment, thus underlining the worth in having a human feedback teacher in this algorithm.

3.3.2. Future Directions

Here two key areas for the extension of the BC algorithm are identified. The first addresses the potential for critique over-application and the option for real-valued feedback. The second discusses providing human feedback that contains more information than just a binary performance flag.

3.3.2.1. Directional Critiques. One weakness of this algorithm is that points in D might be unduly penalized by critiquing, since *where* query points are in relation to training datapoints is not considered when updating scaling factor m. Consider two query points located at identical distances, but orthogonal directions with respect, to a given training point. The training point's recommended action might be appropriate for one query point but not the other. If so, recommending its action for would incur different success for each point, and therefore also conflicting feedback. The incorporation of query point orientation into the update of m is thus a potential improvement for this algorithm. A similar approach is taken in the work of Bentivegna (2004), where state-action-*query* Q-values are considered in a kNN policy derivation, and are updated based on learner execution performance.

Another extension to the BC algorithm could incorporate real-valued feedback. Under the current formulation, critiques provide a binary poor performance flag. The incorporation of this feedback into the policy, through an update of the scaling factor m, does depend on the relative distance between the query and dataset points. However, this incorporation could further depend on the *value* of a critique, if the feedback were not binary. In this case, larger critique values could effect larger increases to the scaling factor m. For example, real-valued feedback could be a function of performance metrics, like the success and efficiency metrics defined for this domain (Sec. 3.2.1.3).

3.3.2.2. More Informative Feedback. Providing a binary critique, like providing RL reward, gives the learner an indication of where poor action selection occurred. It does not, however, provide any sort of indication about what should have occurred instead. The only way to determine which action would produce a superior performance is to revisit the state and execute a different action. Such an approach easily becomes intractable on real robot systems operating in worlds of infinite state-action combinations.

Furthermore, under a LfD paradigm, any exploratory actions taken by the learner are restricted to either those that were demonstrated by the teacher, or those that possibly result from regression generalization. Note that generalized actions only result from certain regression techniques, for example those that produce a soft average over dataset actions (e.g. kernelized averaging) but not those that strictly select a single dataset action (e.g. 1-NN). If a desired action was not demonstrated by the teacher, possible reasons are that during demonstration the teacher never visited a state from which taking this action was appropriate, or that the teacher *did* visit such a state but the teacher is suboptimal. Regardless of the reason, it is possible that the learner does not even have access to a better action for an execution state that received a poor performance critique.

We posit that more informative, *corrective*, feedback would prove useful to the learner. Furthermore, feedback that expands the set of available actions *beyond* those contained within the demonstration set should also improve policy performance. The development of our *advice-operator* feedback technique, introduced in the following chapter, is grounded on both of these considerations.

3.4. Summary

Binary Critiquing (BC) has been introduced as an algorithm that credits policy performance with teacher feedback in the form of binary performance flags. The teacher selects poor performance

portions of a learner execution to receive the binary credit, thus overcoming the issue of credit assignment common in many reward-based feedback algorithms. The learner uses the crediting feedback to update its policy; in particular, the feedback is used to modify how the underlying demonstration data is treated by the policy derivation technique. By thus weighting the demonstration datapoints according to their relative performance, the robot is able to improve its policy.

The BC algorithm was implemented within a simulated robot motion interception domain. Empirical results showed improvements in success and efficiency as a consequence of the crediting feedback. Interestingly, the performance of the final learned policy exceeded the performance of the demonstration policy, without modifying the contents of the demonstration dataset.

There are many benefits to using a human for credit assignment. Within the BC algorithm particularly, the human selects portions of an execution to flag as poorly performing. Rather than just crediting poor performance where it occurs, segment selection allows for behaviors that *lead* to poor performance to receive poor credit as well. Furthermore, since a human evaluates the execution, the robot is not required to be able to detect or measure the aspects of an execution that define poor performance. A human evaluator also allows for the option of evaluation metrics that are too subtle or complex to be detected, and represented computationally, within the learning system.

Future research directions for the BC algorithm were identified to included directional critiques that consider query point orientation, and real-valued critiques that impart a measure of relative rather than just binary performance. Feedback that is corrective was also identified as a promising direction for future work, motivating the development of corrective techniques in the A-OPI algorithm, introduced in the following chapter.

CHAPTER 4

Advice-Operators

To provide a correction on an executed learner behavior is a very direct manner by which to refine a policy. Corrections do not require any exploration on the part of the learner, since the preferred action to take, or state to enter, is explicitly indicated. When provided by a human, corrections additionally do not require that any automatic policy evaluations be performed by the robot learner or by the learning system as a whole.

The work of this thesis focuses on low-level robot motion control within continuous action-spaces. To indicate a preferred action or state within continuous state-action spaces requires providing a continuous-valued correction. While a human teacher may have a general idea of how to correct the behavior, expecting him to know the appropriate continuous *value* that corrects an execution point is neither reasonable nor efficient. For example, a human teacher may know that an execution should have been faster, but not that the speed should have been exactly $2.3\frac{m}{s}$ instead of $2.1\frac{m}{s}$. Furthermore, since our motion control policies are sampled rapidly, e.g. at $30Hz$, any executed robot behavior deemed by the human to need correcting likely endured over multiple execution points. Continuous-valued corrections therefore must be provided for *all* of these points, thus further increasing the burden on the teacher. With our contributed correction approach, we aim to circumvent both of these difficulties.

This chapter introduces *advice-operators* as a language through which a human teacher provides corrections to a robot student. Advice-operators perform mathematical computations on continuous-valued datapoints. To provide a correction, the teacher selects from a finite list of advice-operators. The robot learner applies the operator to an execution datapoint, modifying its value and producing a continuous-valued correction. Paired with the segment selection approach of our feedback framework, an operator is applied over all of the execution points within the segment. The selection of a *single* piece of advice and application segment therefore provides *continuous*-valued corrections on *multiple* execution points. In this manner, our approach enables correction-giving that is reasonable for a human to perform, even within our continuous-valued, rapidly sampled, domain.

While advice-operators significantly simplify the process of providing continuous-valued corrections, the operators themselves still must be defined. In this chapter we additionally contribute a principled approach to the development of action advice-operators, which we employ in later chapters of this thesis.

We also present in this chapter a comparison of the data produced through the techniques of more demonstration versus advice-operators, since one motivation for providing policy corrections within an LfD framework is to provide an alternative to teacher demonstrations. Common limitations to LfD datasets include poor teacher-learner correspondence and suboptimal teacher performance, neither of which may be addressed by providing more teacher demonstrations. We show the two techniques to produce data that exists in different areas of the state and action space, and furthermore validate advice-operators as an effective approach for policy improvement that produces smaller datasets in addition to superior performance.

This chapter begins with an overview of advice-operators. In Section 4.2 our principled approach to the development of action advice-operators is detailed. Section 4.3 then presents a comparison between data produced from teacher feedback and more demonstration. Directions for future research are identified in Section 4.4.

4.1. Overview

This section introduces our approach to correcting policies within continuous state-action spaces. We begin with a description of our correction technique, followed by a delineation of some requirements for the application of this technique.

4.1.1. Definition

To address the challenge of providing continuous-valued corrections, we introduce *advice-operators* as a language through which a human teacher provides policy corrections to a robot student. Concretely defined, an advice-operator is a mathematical computation performed on an observation input or action output. Key characteristics of advice-operators are that they:

- Perform mathematical computations on datapoints.
- Are defined commonly between the student and advisor.
- May be applied to observations or actions.

To illustrate with an example, consider a simple operator that modifies translational acceleration by a static amount δ. Suppose this operator is indicated, along with a segment of 15 execution data points. The translational speed a^0 of executed point 0 then updates to $\hat{a}^0 \leftarrow a^0 + \delta$, point 1 to $\hat{a}^1 \leftarrow a^1 + \delta$, and so forth until point 14 updates to $\hat{a}^{14} \leftarrow a^{14} + \delta$.

4.1.2. Requirements for Development

Advice-operators perform mathematical computations on the observations or actions of execution datapoints. To develop advice-operators requires that the designer be able to isolate specific elements of a datapoint formulation as responsible for producing the particular behavior that will be corrected by the operator. The majority of the operators developed and used in the work of this thesis are *action*-modifying operators; this consideration is discussed further in Section 4.4.2. We therefore restrict this discussion of advice-operator development to action-modifying operators,

while noting that a parallel, though not necessarily equivalent, discussion exists for the development of observation-modifying operators.

Within this action context, to isolate the necessary datapoint elements involves identification of the component actions that are responsible for producing the particular aspect of the net action that the operator is intended to correct. For example, if a net action involves both rotational and translational speeds, to develop an operator that increases linear velocity involves isolating the *translational speed* component of the net action. An appropriate modification for this isolated element, that produces the desired behavior change, is then additionally required; namely, that *increasing* the translational speed will result in the desired modification of increased linear velocity.

Advice-operator development depends generally on the capabilities of the robot, and specifically on the state-action space of the task at hand. The generality of an operator - that is, its applicability to more than one task - depends on design decisions made by the developer. Many valid formulations for *how* to develop advice-operators potentially exist, the most straightforward of which is to develop each operator by hand and without generality to other tasks. Advice-operator development under our approach, presented in the following section, produces operators that are as general as the defined actions. That is, for task-specific actions, task-specific operators will result. For actions general to the robot, general advice-operators will result. Actions common to multiple tasks on the robot therefore are able to share advice-operators developed under our technique.

4.2. A Principled Approach to Advice-Operator Development

This section contributes our principled approach to the development of action advice-operators. We first describe the baseline advice-operators defined under our development approach, followed by our technique for building new operators through the scaffolding of existing operators. The section concludes with a presentation of constraints imposed on the system to keep the synthesized data firmly grounded within the robot's capabilities.

4.2.1. Baseline Advice-Operators

Advice-operator development under our approach begins with the definition of a baseline set of advice-operators. For multi-dimensional action predictions, e.g. a 2-D prediction consisting of rotational speed and translational speed, the baseline operators are defined for each single action dimension, and function as first and second order modifications to the value of that action dimension. Concretely, given a recorded observation-action execution trace d and selection Φ, such that subset $d_\Phi \subseteq d$ contains executed actions $a_i \in d_\Phi$ (of a single action dimension), the baseline operators are:

- *Static*: A static modification amount δ, applied equally across all selected points, producing action $\hat{a}_i \leftarrow a_i + \delta$.
- *Fractional*: A fractional modification amount α, that operates on the executed action values $a_i \in d_\Phi$, producing action $\hat{a}_i \leftarrow a_i + \alpha a_i$.

- *Incremental Fractional*: Fractional modification amounts $\frac{i}{|\Phi|}\beta$, that operate as *linearly increasing* fractional modification amounts on the executed action values $a_i \in d_\Phi$, producing actions $\hat{a}_i \leftarrow a_i + \frac{i}{|\Phi|}\beta a_i$.

Here $|\Phi|$ is the size of the selected segment, $i = 1 \dots |\Phi|$ and $\delta > 0, \alpha > 0, \beta > 0$ are all constant parameters. Each operator furthermore takes a binary parameter, indicating a positive or negative modification amount.

The first operator *Static* allows for the static augmentation of an action value, particularly useful if the value was previously zero or of opposite sign to the desired value. The second operator *Fractional* enables augmentations that increase or decrease according to the size of the executed action value, which is useful for producing smooth changes as well as very large or very small modification amounts. The reasoning behind the final operator *Incremental Fractional* is to allow incremental easing into a modification. For example, in the case of ten selected actions $a_1, a_2 \dots a_{10} \in d_\Phi$, this operator adds $\frac{\beta}{10}a_1$ to the first action and $\frac{2\beta}{10}a_2$ to the second action, up to the addition of βa_{10} to the final action in the set. At first glance, this may appear to represent an "acceleration" across the executed points added to the dataset. Careful to note, however, is that points are treated individually by the regression techniques and no notion of execution trace or sequence is taken into account once points have been added to the demonstration set. The only difference between this operator (Incremental Fractional) and the second operator (Fractional) therefore is that smaller modifications are made to earlier points, and larger modifications to later points, along the selected execution segment.

For the experimental implementations of the following chapters, the parameter values are set as $\alpha = \frac{1}{3}$ and $\beta = \frac{1}{2}$. Section 4.2.2.1 describes how to set δ. The three baseline operators are developed for each of two single actions, translational and rotational speed, resulting in six baseline operators. Note that these operators are specific to motion control on our differential drive robot in general, and not to any particular motion task.

4.2.2. Scaffolding Advice-Operators

Advice-operator development continues with the definition of more complex operators. In particular, these operators are built, or scaffolded, from existing operators, beginning with the baseline set. Our approach provides an interface through which existing advice-operators are composed and sequenced into more complex operators.

The advice-operator building interface functions as follows. First the existing *children* operators that will contribute to the new *parent* operator are selected. Second, the range over which each operator will be applied is indicated. This allows for flexibility in the duration of the contributing operators, such that a given child operator is applied over only a portion of the execution segment if desired. For adaptability to any segment size, application range is indicated by a percentage, e.g. a range of $25 \rightarrow 50\%$ indicates the second quarter of a segment. Figure 4.1 presents an example applicability scenario where one contributing operator (Fractional Translational Speed) has an associated range of $0 \rightarrow 75\%$ and thus is applied over the first three-quarters of the execution

segment, and a second contributing operator (Static Rotational Speed) has a range of 50 → 100% and thus is applied over the final half of the segment.

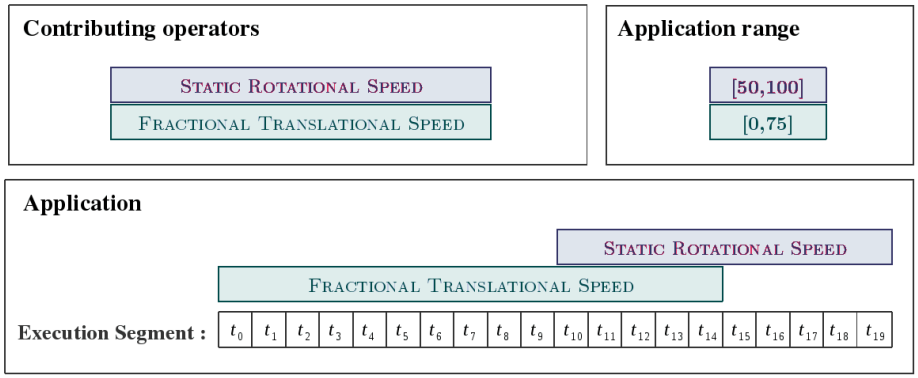

FIGURE 4.1. Example applicability range of contributing advice-operators.

In the third step, the parameters for the new operator are specified. To define a parameter for the parent operator, a single parameter is selected from the parameter list of each contributing child operator. The new parameter combination across all children is given a name, and this set now constitutes a single new parameter for the parent operator. Another parameter combination may then be specified, as per the intents of the designer; the only limit on the number of parameters that may be associated with a given operator is the number of unique combinations of child parameters. The new operator is then added to the operator list and available for immediate use.

Upon application, selecting this new operator and one of its parameters triggers an application hierarchy. The hierarchy begins with the application of each indicated child operator, over the execution segment subset indicated by the application range of the child operator, and with the parameter specified within the selected parent parameter combination. If a given child operator is not itself a baseline operator, it too is defined by a set of children operators, its associated parameter specifies a parameter combination for these children, and these children are accordingly applied. This process continues until all operators under consideration are baseline operators, whose application is mathematically defined as described in Section 4.2.1.

To illustrate this building interface, consider the development of an operator named *Adjust Turn*, outlined in Figure 4.2. In step one the contributing child operators are selected. Two operators are indicated: the Fractional operator for the *rotational* speed action, and the Fractional operator for the *translational* speed action. In step two the applicability range of each child operator is indicated; here both operators are applicable over the entire execution segment (0 → 100% of the segment). In step three, parameters for the new operator are specified. The first parameter specified, named *tighten*, sets the parameter for the rotational speed Fractional operator as *increase* and the translational speed Fractional operator as *decrease*. The second parameter specified, named *loosen*, sets the parameter for the rotational speed Fractional operator as *decrease* and the translational

speed Fractional operator as *increase*. The overall functioning of the new operator is to tighten a turn by increasing rotational speed and decreasing translational speed, or alternately to loosen a turn by decreasing rotational speed and increasing translational speed.

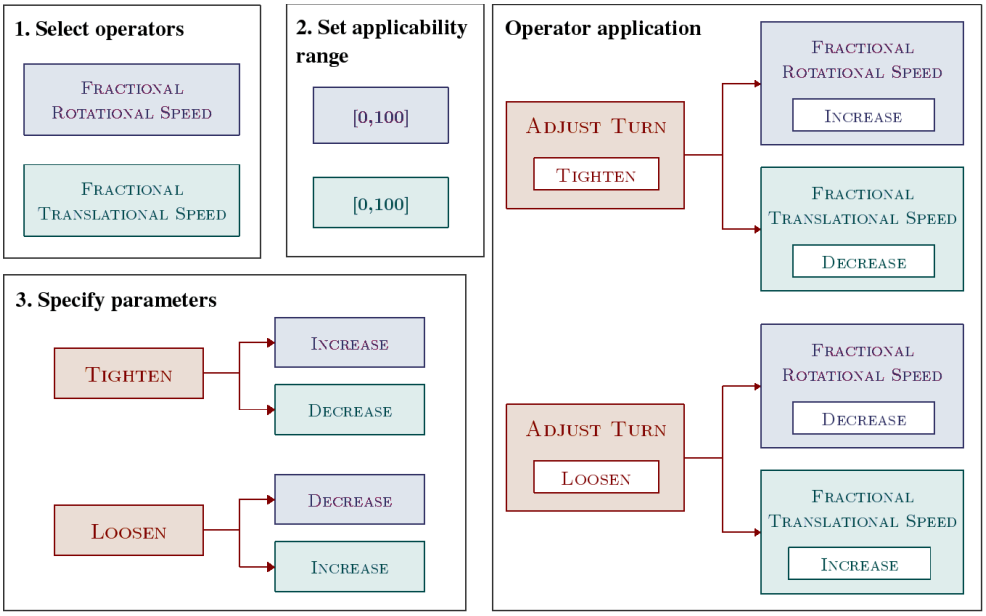

FIGURE 4.2. Advice-operator building interface, illustrated through an example (building the operator *Adjust Turn*).

In summary, advice-operators are built as a hierarchy, or tree, of existing operators. Selection of an operator triggers a sequence of calls to underlying operators. The leaf nodes of this tree are the baseline operators, whose functioning is specified by the basic hand-written mathematical functions described in the previous section. The mathematical functioning of all non-baseline operators thus results from the sequencing and composition of functions built from the baseline mathematics.

4.2.2.1. Constraining Corrections. The data modification techniques presented here amount to data *synthesis* from learner executions and teacher advice. This synthesis is subject to multiple constraints intended to produce synthesized data that is firmly grounded on both the underlying learner execution and the physical capabilities of the robot.

Beyond being a function of executed action values, the modification amounts produced by advice-operators are further tied to the physical constraints of the robot. In particular, assuming a standard acceleration value of $\dot{\nu}$, the static modification amount δ of operator *Static* is defined as $\dot{\nu}dt$, where dt is the execution framerate.[1] Our systems runs at $30Hz$, and thus $dt = 0.0\overline{3}3s$.

The final advice-modified action is additionally constrained to lie within an amount η of the executed action value. This constrains the synthesized points to lie near points that were actually executed, and thus near the capabilities of the existing demonstration set from which the policy that

[1]If a standard acceleration value is not defined for the robot system, another reasonable option for this value may be selected, for example average acceleration.

produced this execution was derived. In particular, assuming a *maximum* acceleration value of $\dot{\nu}_{mx}$, the constraining amount η is defined as $\dot{\nu}_{mx}dt$, where dt again is the execution framerate.[2] This theoretically guarantees that the advice-modified datapoint is reachable from the executed point within 1-timestep.

4.2.3. Addressing Suboptimal Synthesized Data

One consequence of constraining modifications in this manner is that the mapping represented by the synthesized data might not be consistent with the behavior desired of the *final* policy. This is because the synthesized data is constrained to lie near the learner execution, which, given that the execution was corrected, presumably was *not* consistent with the final desired behavior. Though the corrected, synthesized mapping does represent an *iterative* improvement on the mapping exhibited by the learner execution, this corrected mapping may still conflict with the desired behavior of the final policy. If the corrected mapping does not match the behavior desired of the final policy, the synthesized data equates to a suboptimal demonstration of the final policy behavior. Like any suboptimal demonstration, this data will degrade policy performance.

As an illustration, consider a learner executed translational speed, $1.2\frac{m}{s}$, that is much slower than the target behavior speed, $2.5\frac{m}{s}$, and a constrained modification amount of $0.3\frac{m}{s}$. A step in the direction of the target speed produces an advice-modified action with value $1.5\frac{m}{s}$; to add this data to the demonstration set equates to providing a demonstration at speed $1.5\frac{m}{s}$. While this action is an improvement on the learner executed speed, it is suboptimal with respect to the target behavior speed of $2.5\frac{m}{s}$. The addition of this datapoint to the demonstration set thus amounts to providing the final policy with a suboptimal demonstration.

To circumvent this complication, our approach augments the state observation formulation with internal observations of the current action values. This anchors the action *predictions* of the state-action mapping to the *current* observed action values. Mappings now represent good behavior *given* the current action values, and thus will not conflict with the final policy even if they are not good demonstrations of the target behavior. Returning to our illustration, under this formulation the speed $1.5\frac{m}{s}$ will be considered an appropriate prediction only when the current speed is $1.2\frac{m}{s}$. Should the learner later revisit that area of the world with a speed of $2.0\frac{m}{s}$, for example, the policy will not attempt to slow the learner down to $1.5\frac{m}{s}$.

The benefit of this observation formulation is more robust and flexible advice-giving, since advice that improves the behavior of an iterative policy, but is suboptimal for the final policy, is no longer a hazard. A drawback is that this observation formulation increases the observation-space, which typically correlates to slower learning times and the need for more training data. An alternative solution could be to incorporate LfD techniques that explicitly address suboptimal demonstration data, though to date few such techniques exist (see Ch. 9).

[2]The maximum acceleration value $\dot{\nu}_{mx}$ may be defined either by the physical constraints of the robot or artificially by the control system.

4.3. Comparison to More Mapping Examples

In this section we present a comparison of data produced from the advice-operator technique against data produced from more teacher demonstrations. In both cases, an initial dataset is produced from demonstration, and an initial policy is derived from this dataset. Data produced under the feedback technique is then synthesized from learner executions with its current policy and teacher corrections on these executions. Data from more-demonstrations is produced in the same manner as the initial dataset, namely teacher demonstration, but the demonstrations are provided in response to learner executions with its current policy. To begin, we compare the distribution of data within the observation and action spaces, as produced under each technique. We next compare the quality of the resultant datasets, as measured by dataset size and the relative performance of policies derived from the respective datasets.

4.3.1. Population of the Dataset

The work presented in this section compares two datasets: one built using our feedback techniques, and one built from demonstration exclusively. Both datasets are seeded initially with the same demonstration data, since all of our feedback algorithms rely on an initial policy derived from demonstrations. As the learner executes, the teacher observes the learner performance and offers corrective feedback, in the case of the *feedback* dataset, or more teleoperation demonstrations, in the case of the *more-demonstration* dataset. Note that the feedback provided by the teacher includes advice-operators as well as positive credit (Sec. 2.3.1.1), which equivalently may be viewed as an identity function advice-operator that leaves a datapoint unchanged. The actual task that produced the data of this section is presented in Chapter 6 (building the primitive policy datasets); the details of the task are tangential to the focus of the discussion here.

Figure 4.3 provides a comparison of the distribution of new data within the *observation* space. This histogram presents the distance between observations contained within a dataset and the observation of a new point being added to that dataset, summarized across multiple new datapoints (1239 for feedback, 2520 for more-teleoperation). Along the horizontal axis is the Euclidean 1-NN distance between a new datapoint and the existing dataset, i.e. the distance to the single closest point within the dataset, and along the vertical axis is the relative (fractional) frequency count of a given 1-NN distance value. The most notable difference between the data derived from more demonstration and that derived from feedback lies in the smaller distance values. The more-demonstration approach more frequently provides new data that is close (≤ 0.05) to the existing dataset. Furthermore, when comparing the mean and standard deviation of the distribution of 1-NN distances, the more-teleoperation distribution has a lower mean and larger standard deviation (0.15 ± 0.14) than the feedback distribution (0.21 ± 0.02).

These observation space results suggest that our feedback techniques take larger steps away from the initial demonstration dataset, into more remote areas of the observation space. The demonstration approach possibly also visits these areas, but takes more iterations to get there. Given the

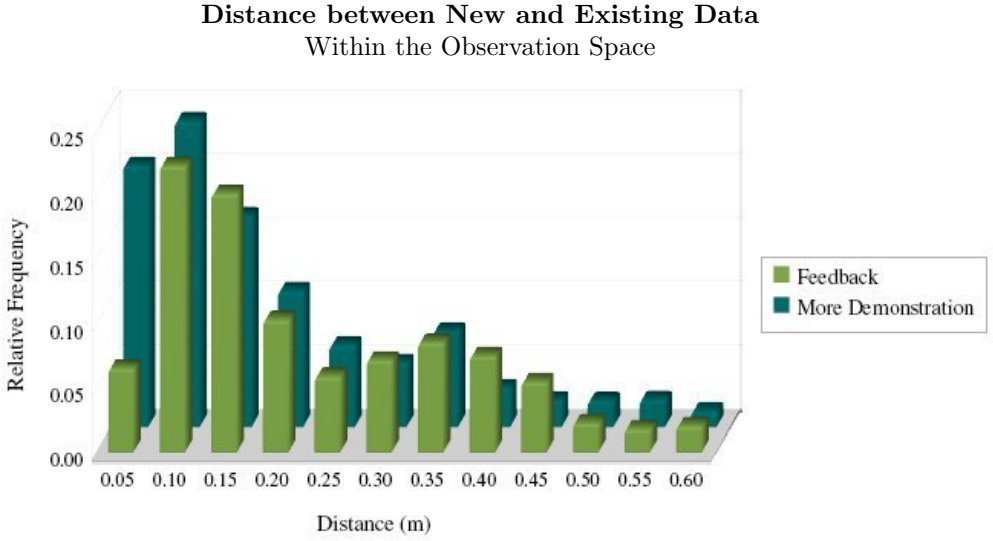

FIGURE 4.3. Distribution of observation-space distances between a newly added dataset point and the nearest point to it within the existing dataset (histogram).

performance results presented in the following section (Sec. 4.3.2) however, the more likely possibility is that there are observation areas which are visited by the learner executions and added, after feedback-modification, the feedback dataset, but that are unvisited by the teacher during demonstration. One possible explanation for this absence of teacher demonstration is a lack of knowledge on the part of the teacher that demonstration is needed in these observation-space areas. Another explanation is that the particular execution technique of the teacher does not take the demonstration into these observation-space areas, or perhaps that limitations in the mechanism employed for demonstration complicate access to these areas; both of these explanations indicate a difference in *correspondence* between the teacher and learner.

Figure 4.4 provides a visual representation of the distribution of new data within the *action* space. This figure plots the 2-D action output (translational and rotational speed) against itself, with each axis representing a single action (a similar plot was not produced for the dataset observations, since the observation space is 6-dimensional and thus difficult to represent visually). The red diamonds plot the original demonstration data, and thus data that is shared between the datasets of each technique. The green triangles plot new data added through more demonstration, and the blue squares new data added through feedback techniques. Visual inspection reveals a similar trend to that observed in the observation-space results: namely, that the actions produced through the more-demonstration technique (green triangles) are closer to the actions already present within the dataset (red squares) and are those actions produced through feedback techniques (blue squares). The mean and standard deviation of the distribution of actions within the initial, more-demonstration and feedback datasets are respectively: $(1.37 \pm 0.65\frac{m}{s}, 0.02 \pm 0.64\frac{rad}{s})$, $(0.96 \pm 0.22\frac{m}{s}, -0.51 \pm 0.39\frac{rad}{s})$ and $(1.83 \pm 0.53\frac{m}{s}, -0.56 \pm 0.35\frac{rad}{s})$. The distance (Euclidean) between these distribution means are: 0.75 initial→more-teleoperation, 0.67 initial→feedback and 0.87 more-teleopration→feedback. These results confirm that more-demonstration dataset is closer to the initial data than the feedback

dataset, and furthermore that the more-demonstration and feedback datasets are (relatively) distant from each other.

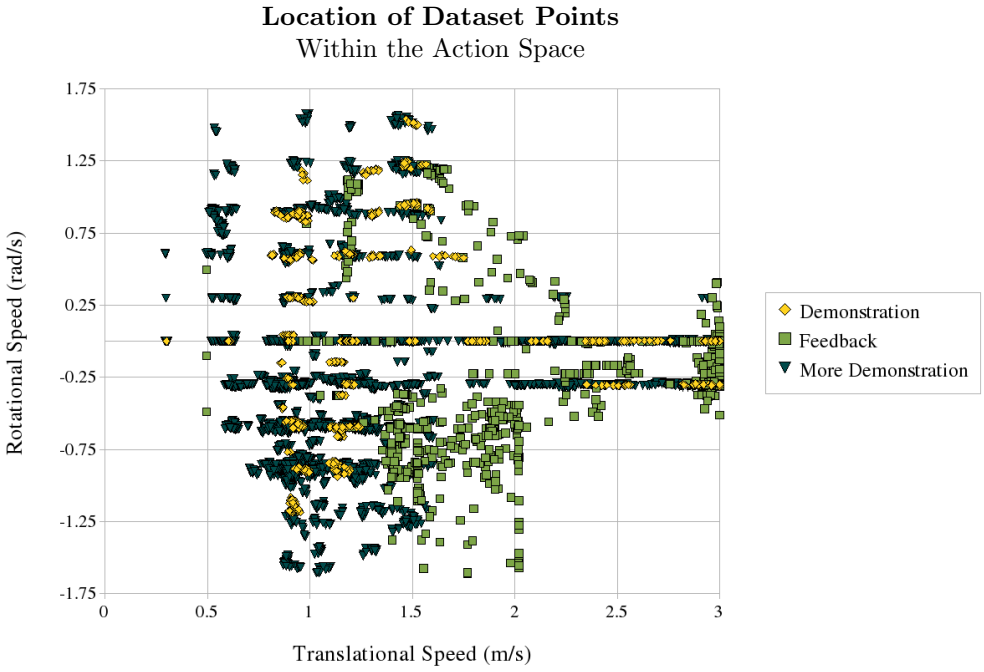

FIGURE 4.4. Plot of the location of dataset points within the action space.

Of particular interest to note is that our feedback techniques produce data within areas of the action space that are entirely *absent* from the contents either demonstration dataset (e.g. in the area around $(1.75\frac{m}{s}, -0.75\frac{rad}{s})$). One possible explanation is that, during demonstration, the teacher never encounters situations, or areas of the observation space, in which these action combinations are appropriate to execute. Another explanation is that the teacher is unable, or unwilling, to demonstrate these action combinations; a hypothesis that is further supported by the observation that many of these action combinations contain higher speeds than the demonstrated action combinations. Also worthwhile to note is that the trend of action similarity between the initial and more-demonstration datasets is somewhat expected, since the same technique that produced the initial dataset actions (teleoperation) also produced the more-demonstration dataset actions. The differences that do exist between these two action sets is likely attributable to executions visiting different observation-space areas with distinct action requirements, as well as the inherent variability of execution within noisy environments.

This section has shown feedback techniques to produce data that is further from the existing dataset within the observation space, and that is absent from either demonstration dataset within the action space. Dataset population within different areas of the observation and action spaces, however, does not necessarily imply that the behavior produced by this data is more desirable. The next section takes a deeper look at the quality of the datasets produced under each approach.

4.3.2. Dataset Quality

The work presented in this section again compares two datasets, one built using our feedback techniques and the other from demonstration. The details of the actual task that produced the data of this section is again tangential to the focus of the discussion here, and is presented in Chapter 6 (building the scaffolded policy datasets, up to the 74^{th} practice run). We define a *practice run* as a single execution by the learner, which then receives either feedback, or an additional demonstration, from the teacher.

Figure 4.5 presents the growth of each dataset over practice runs. The actual dataset sizes, as measured by number of datapoints, are shown as solid lines; since each dataset was initialized with a different number of datapoints, the dashed lines show the dataset sizes with these initial amounts subtracted off. We see that the growth of the more-demonstration dataset far exceeds that of the feedback dataset, such that in the same number of practice runs the more-demonstration approach produces approximately three times the number of datapoints.

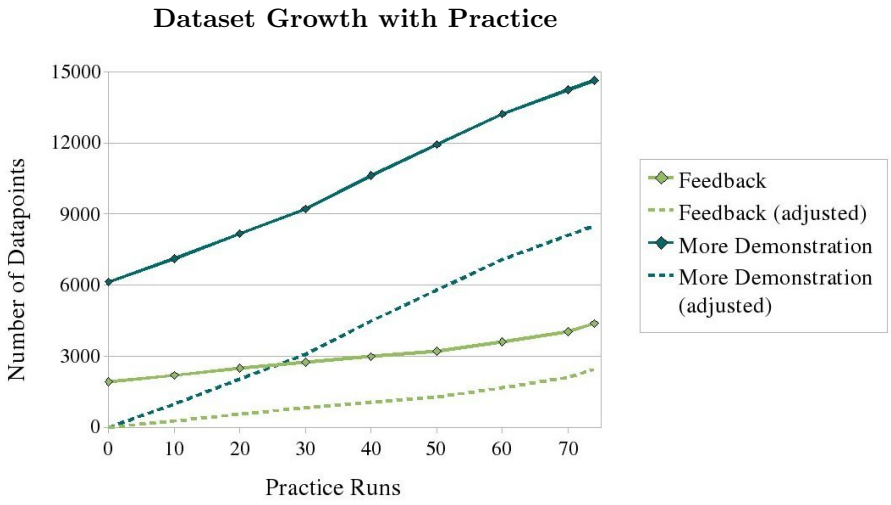

FIGURE 4.5. Number of points within a dataset across practice runs (see text for details).

For a given poor learner execution, usually only a portion of the execution is in need of improvement. The more-demonstration technique provides to the learner with a complete new execution, while the feedback technique corrects and provides only portion of an execution. For this reason, the more-demonstration technique nearly always[3] adds more points to the dataset than the technique of providing feedback. If it were the case that the dataset produced through feedback techniques was underperforming the more-demonstration dataset, this would suggest that the smaller dataset in fact had omitted some relevant data. This, however, is not the case.

Figure 4.6 presents the performance improvement of these datasets over practice runs. Here performance improvement is measured by the percentage of the task successfully completed. The

[3]The exceptions being when the entire learner execution actually is in need of correction, or if the portion requiring improvement is at the start of the execution.

performance of both datasets is initially quite poor ($0.58 \pm 0.18\%$ for more-demonstration, $5.99 \pm 0.24\%$ for feedback, average of 10 executions). Across practice runs, the performance of the more-demonstration dataset does improve, marginally, to a $13.69 \pm 0.02\%$ success rate (average of 50 executions). The performance of the feedback dataset is a marked improvement over this, improving to a $63.32 \pm 0.28\%$ success rate[4] (average of 50 executions).

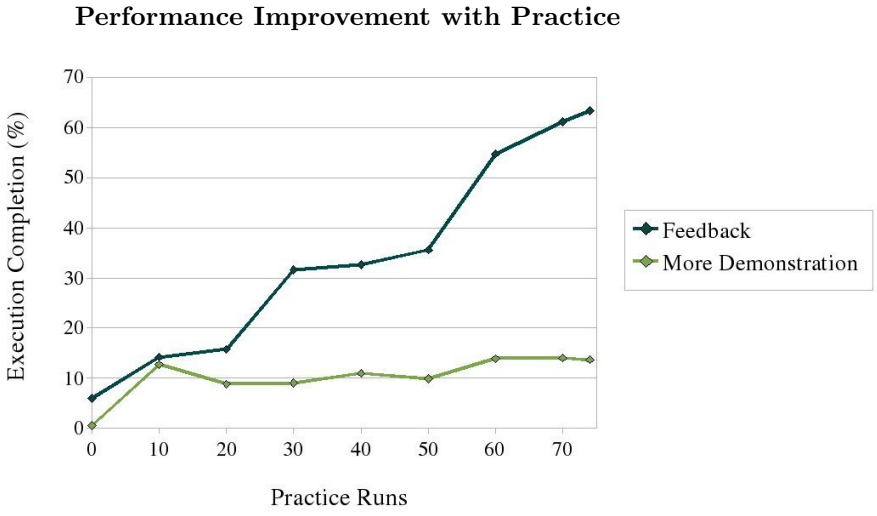

FIGURE 4.6. Performance improvement of the policies derived from the respective datasets across practice runs.

With these performance results, we may conclude that our feedback technique is omitting primarily redundant data, that does not improve policy performance. Since the feedback dataset in fact actually *exceeds* the performance of the more-demonstration dataset, we may further conclude that relevant data is being *missed* by demonstration, in spite of its larger dataset. Furthermore, this data is being *captured* by teacher feedback.

4.4. Future Directions

The work presented in the following chapters of this thesis, further validate advice-operators as effective tools for correcting policies within continuous action-spaces. Many future directions for the development and application of advice-operators exist, a few of which are identified here. The first is the application of action advice-operators to more complex spaces than 2-dimensional motion control. Next is the development of observation advice-operators. To conclude we consider are feedback techniques that correct points within the demonstration dataset.

4.4.1. Application to More Complex Spaces

Advice-operators that influence translational and rotational speeds prescribe the limit of advice-operators explored in this thesis. This limit is motivated by two considerations. The first is our focus

[4]The feedback technique is able to achieve a 100% (approximately) success rate with more practice runs; see the results in Section 6.2.2.3 for further details.

on motion control tasks for a Segway RMP robot, which is a wheeled differential drive robot without manipulators or other motion control considerations requiring high-dimensional action-spaces. The second is that this is a manageable action-space for the initial development and implementation of the advice-operator technique; namely, that translational and rotational speeds are straightforward to isolate for the purposes of behavior correction and thus for advice-operator development.

Low-dimensional action-spaces do not, however, define the limit of advice-operators in general. The application of advice-operators to higher-dimensional action-spaces is not assumed to be straightforward or obvious, and may require the development of additional translation techniques. For example, consider a manipulation task performed by a humanoid arm. An advice-operator that instructs the arm end effector to a different relative 3-D position in space, e.g. "more to the right", might be simple for a feedback teacher to use, i.e. to identify as the fix for a particular executed behavior. The *workings* of this advice-operator however, that translate "more to the right" into modified joint angles and actuator speeds, are *not* simple, and advice-operator development therefore is not straightforward. The introduction of an additional translation mechanism in this case could be an inverse controller that translates 3-D end effector position into joint angles. The advice-operator then only needs to translate "more to the right" into a 3-D spatial position; a simpler task than the translation into joint angles and actuation speeds.

We identify the development of advice-operators for more complex action-spaces as a rich area for future research. Until such research is performed, no concrete claims may be made about the feasibility, or infeasibility, of advice-operators as tools for policy correction in high-dimensional action-spaces.

4.4.2. Observation-modifying Operators

The majority of the advice-operators developed for the work of this thesis are *action*-modifying advice-operators. Equally valid, however, are *observation*-modifying advice-operators.

The robot experiments of the following chapter (Ch. 5) will present one such observation-modifying example. The general idea of this technique will be to encode the target value for a goal into the observation computation, and develop an operator that modifies the state-action mapping by resetting the target value of this goal to the executed value. In addition to behavior goals, target values for execution metrics could similarly be encoded into the observation computation and modified by an associated operator. This general idea - to encode a target value for some measure and then modify it to match the executed value - is one option for the development of observation operators.

To develop operators that modify observation elements that are not a product of performance metrics or goals, however, becomes more complicated. For example, to modify an observation that computes the distance to a visually observed obstacle essentially amounts to hallucinating obstacle positions, which could be dangerous. The development, implementation and analysis of more observation-modifying advice-operators is identified as an open ended and interesting direction for future research.

4.4.2.1. Feedback that Corrects Dataset Points. Our corrective feedback technique modifies the value of learner-executed data, and thus *corrects* points *executed* by the *learner*. Our negative binary credit feedback technique (or critique, Ch. 3) modifies the use of the demonstration data by the policy, and thus *credits* points within the *dataset*. An interesting extension of these two techniques would be to define a feedback type that *corrects* points within the *dataset*.

Our advice-operator feedback technique corrects a policy by providing new examples of the correct behavior to the policy dataset. Poor policy performance can result from a variety of causes, including dataset sparsity, suboptimal demonstrations, and poor teacher-learner correspondence. These last two causes produce poor quality demonstration data. The more examples of good behavior provided through corrective feedback, the less influence this poor quality data will have on a policy prediction. The poor quality data will, however, always be present, and therefore still be considered to some extent by the regression techniques of the policy.

An alternate approach could correct the actual points within the dataset, instead of adding new examples of good behavior. The advantage of such an approach would be to increase the influence of a correection and thus also the rate of policy improvement, since a gradual overpowering of the suboptimal demonstration data with good examples would not be necessary; the bad data would be corrected itself directly. How to determine which dataset points should be corrected would depend on the regression technique. For example, with 1-NN regression, the single point that provided the regression prediction would be corrected. With more complex regression approaches, such as kernelized regression, points could be corrected in proportion to their contribution to the regression prediction, for example.

This approach is somewhat riskier than just adding new data to the set, since it assumes to know the *cause* of the poor policy performance; namely, suboptimal demonstration data. As was discussed in the previous chapter, there are many subtleties associated with crediting dataset points for poor policy performance (Sec. 3.3.2.1). Taking care to correct only *nearby* datapoints would be key to the soundness of this approach. Indeed, in our application of binary critique feedback, the extent to which demonstration datapoints are penalized is scaled inversely with the distance between that point and the query point for which the predication was made. The F3MRP framework already computes a measure of dataset support for a given prediction; this measure could be used to determine whether the dataset points contributing to the prediction should be corrected or not, for example. Alternatively, this measure could be used to decide between two incorporation techniques; namely, to correct the dataset point if the prediction was well supported, and to add a new behavior example if the prediction was unsupported.

4.5. Summary

This chapter has introduced advice-operators as a technique for providing continuous-valued policy corrections. Paired with the segment selection technique of our F3MRP framework, the approach becomes appropriate for correcting continuous-valued policies sampled at high frequency, such as those in motion control domains. In particular, the selection of a single execution segment and single advice-operator enables the production of continuous-valued corrections for multiple execution datapoints.

We have presented our principled approach to the development of action advice-operators. Under this paradigm, a baseline set of operators are automatically defined from the available robot actions. A developer may then build new operators, by composing and sequencing existing operators, starting with the baseline set. The new datapoints produced from advice are additionally constrained by parameters automatically set based on the robot dynamics, to firmly ground the synthesized data on a real execution as well as the physical limits of the robot.

A comparison between datasets produced using exclusively demonstration versus our teacher feedback techniques was additionally provided. The placement of data within the observation and action spaces was shown to differ between the two approaches. Furthermore, the feedback technique produced markedly *smaller datasets* paired with considerably *improved performance*. Note that in the next chapter (Ch. 5) we will show policies derived from datasets built from teacher feedback to exhibit *similar or superior* performance to those built from more teacher demonstration. In Chapter 6 we will show not only superior performance, but the *enablement* of task completion.

Directions for future work with advice-operators were identified to include their application to more complex action spaces. Such an application would provide a concrete indication of the feasibility, or infeasibility, of advice-operators as a tool for policy correction within high-dimensional action-spaces. The implementation of a full set of observation-modifying operators was another area identified for future research, along with operators that enact a broader impact by modifying the contents of the dataset.

CHAPTER 5

Policy Improvement with Corrective Feedback

THE ability to improve a policy from learner experience is useful for policy development, and one very direct approach to policy improvement is to provide corrections on a policy execution. Corrections encode more information than the binary credit feedback of the BC algorithm (Ch. 3), by further indicating a preferred state-action mapping. Our contributed *advice-operators* technique (Ch. 4) is an approach that simplifies providing policy corrections within continuous state-action spaces. In this chapter we introduce *Advice-Operator Policy Improvement (A-OPI)* as an algorithm that employs advice-operators for policy improvement within an LfD framework (Argall et al., 2008). Two distinguishing characteristics of the A-OPI algorithm are *data source* and the *continuity* of the state-action space.

Data source is a key novel feature of the A-OPI approach. Though not an explicit part of classical LfD, many LfD systems do take policy improvement steps. One common approach is to add more teacher demonstration data in problem areas of the state space and then re-derive the policy. Under A-OPI, more data is instead synthesized from *learner executions* and *teacher feedback*, according ot the advice-operator technique. In addition to not being restricted to the initial demonstration set contents, under this paradigm the learner furthermore is not limited to the abilities of the demonstrator.

Also key to A-OPI is its applicability to continuous state-action spaces. Providing corrections within continuous state-action spaces represents a significant challenge, since to indicate the correct state or action involves a selection from an infinite set. This challenge is further complicated for policies sampled at high frequency, since to correct an overall executed behavior typically requires correcting multiple execution points and thus multiple selections from the infinite set. The A-OPI algorithm provides corrections through advice-operators and the segment selection technique of our F3MRP feedback framework. Under our algorithm, a single piece of advice there provides *continuous*-valued corrections on *multiple* executions points. The A-OPI algorithm is therefore suitable for continuous state-action policies sampled at high frequency, such as those present within motion control domains.

In this chapter we introduce A-OPI as an novel approach to policy improvement within LfD. A-OPI is validated on a Segway RMP robot performing a spatial positioning task. In this domain, A-OPI enables similar or superior performance when compared to a policy derived from more teacher

demonstrations. Furthermore, by concentrating new data exclusively to the areas visited by the robot and needing improvement, A-OPI produces noticeably smaller datasets.

The following section introduces the A-OPI algorithm, providing the details of overall algorithm execution. Section 5.2 presents a case study implementation of A-OPI, using simple advice-operators to correct a motion task. Section 5.3 presents an empirical implementation and evaluation of A-OPI, using a richer set of advice-operators to correct a more complex motion task. The conclusions of this work are provided in Section 5.4, along with the identification of directions for future research.

5.1. Algorithm: Advice-Operator Policy Improvement

The Advice-Operator Policy Improvement (A-OPI) algorithm is presented in this section. To begin, we briefly discuss providing corrections within the algorithm, followed by a presentation of the algorithm execution details.

5.1.1. Correcting Behavior with Advice-Operators

The purpose of advice within A-OPI is to correct the robot's policy. Though this policy is unknown to the human advisor, it is represented by the observation-action mapping pairs of a student execution. To correct the policy, this approach therefore offers corrective information about these executed observation-action pairings. The key insight to the A-OPI approach is that pairing a modified observation (or action) with the *executed* action (or observation) now represents a corrected mapping. Assuming accurate policy derivation techniques, adding this datapoint to the demonstration set and re-deriving the policy will thus correct the policy. Figure 5.1 presents a diagram of data synthesis from student executions and teacher advice (bottom, shaded box); for comparison, the standard source for LfD data from teacher executions is also shown (top).

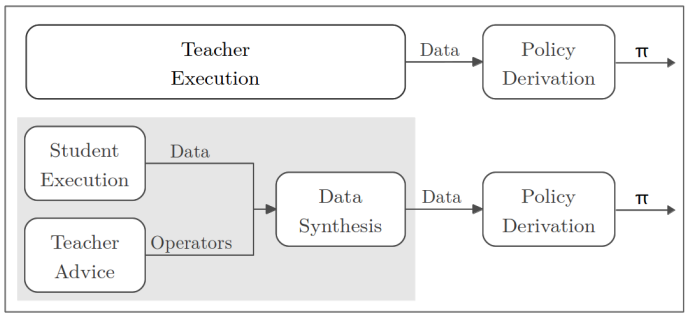

FIGURE 5.1. Generating demonstration data under classical LfD (top) and A-OPI (bottom).

This work empirically explores advice-operators at two levels. The first looks at simple operators that add static corrective amounts (Sec. 5.2). The second looks at more complex and flexible operators, that compute non-static corrective amounts according to functions that take as input the values of the executed observations or actions (Sec. 5.3).

Unlike BC, the incorporation of teacher feedback does *not* depend on the particulars of the regression technique, and any may be used. This implementation employs the Locally Weighted Learning regression technique described in Section 2.1.1.3. Here the observation-scaling parameter

Σ^{-1}, within which the width of the averaging kernel is embedded, is tuned through Leave-One-Out-Cross-Validation (LOOCV). The implemented LOOCV minimizes the Least Squared Error of the regression prediction, on the demonstration dataset D.

Teacher feedback under the A-OPI algorithm involves indicating an appropriate advice-operator, along with a segment of the execution over which to apply the operator. This feedback is used by the algorithm to *correct* the learner execution. The corrected execution points are then treated as new data, added to the demonstration set and considered during the next policy derivation. Figure 5.2 presents a schematic of this approach, where dashed lines indicate repetitive flow and therefore execution cycles that are performed multiple times. In comparison to the schematic of the baseline feedback algorithm (Fig. 2.4), note in particular the details of the dataset update.

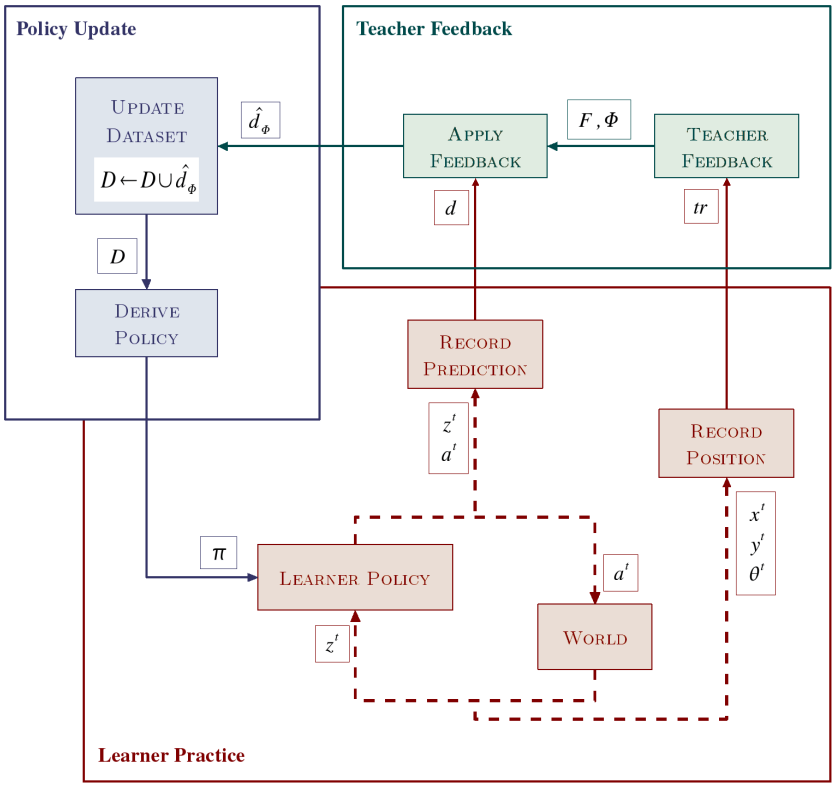

FIGURE 5.2. Policy derivation and execution under the Advice-Operator Policy Improvement algorithm.

5.1.2. Algorithm Execution

Algorithm 3 presents pseudo-code for the A-OPI algorithm. This algorithm is founded on the baseline feedback algorithm of Chapter 8. The distinguishing details of A-OPI lie in the type and application of teacher feedback (Alg. 1, lines 17-18).

The teacher demonstration phase produces an initial dataset D, and since this is identical to the demonstration phase of the baseline feedback algorithm (Alg. 1, lines 1-8), for brevity the details

of teacher execution are omitted here. The learner execution portion of the practice phase (Alg. 3, lines 5-9) also proceeds exactly as in the baseline algorithm, recording the policy predictions and robot positions respectively into the execution traces d and tr.

Algorithm 3 *Advice-Operator Policy Improvement*

1: Given **D**
2: *initialize* $\pi \leftarrow$ `policyDerivation`(D)
3: **while** *practicing* **do**
4: *initialize* $d \leftarrow \{\},\ tr \leftarrow \{\}$
5: **repeat**
6: *predict* $\mathbf{a}^t \leftarrow \pi(\mathbf{z}^t)$
7: *execute* $\mathbf{a}^t$
8: *record* $d \leftarrow d \cup (\mathbf{z}^t, \mathbf{a}^t),\ tr \leftarrow tr \cup (x^t, y^t, \theta^t)$
9: **until** done
10: *advise* $\{\ op, \Phi\ \} \leftarrow$ `teacherFeedback`$(\ tr\ )$
11: **for all** $\varphi \in \Phi,\ (\mathbf{z}^\varphi, \mathbf{a}^\varphi) \in d$ **do**
12: **if** *op* is observation-modifying **then**
13: *modify* $(\mathbf{z}^\varphi, \mathbf{a}^\varphi) \leftarrow (op(\mathbf{z}^\varphi), \mathbf{a}^\varphi)$
14: **else** {*op* is action-modifying}
15: *modify* $(\mathbf{z}^\varphi, \mathbf{a}^\varphi) \leftarrow (\mathbf{z}^\varphi, op(\mathbf{a}^\varphi))$
16: **end if**
17: **end for**
18: *update* $D \leftarrow D \cup \hat{d}_\Phi$
19: *rederive* $\pi \leftarrow$ `policyDerivation`(D)
20: **end while**
21: **return** π

During the teacher feedback portion of the practice phase, the teacher first indicates a segment Φ, of the learner execution trajectory requiring improvement. The teacher feedback is an advice-operator $op \equiv F$ (Sec. 2.4.2), selected from a finite list, to correct the execution within this segment (line 10).

The teacher feedback is then applied across all points recorded in d and within the indicated subset Φ (lines 11-17); this code segment corresponds to the single `applyFeedback` line of Algorithm 1. For each point $\varphi \in \Phi,\ (\mathbf{z}^\varphi, \mathbf{a}^\varphi) \in d$, the algorithm modifies either its observation (line 13) or action (line 15), depending on the type of the indicated advice-operator. The modified datapoints are added to the demonstration set D (line 18).

These details of advice-operator application are the real key to correction incorporation. To then incorporate the teacher feedback into the learner policy simply consists of rederiving the policy from the dataset D, now containing new advice-modified datapoints (line 19).

5.2. Case Study Implementation

This section presents a case-study implementation of the A-OPI algorithm on a real robot system. The task is executed by a Segway RMP robot, which is a dynamically balancing differential drive platform produced by Segway LLC (Nguyen et al., 2004). Initial results were promising, showing improved performance with corrective advice and superior performance to providing further teacher demonstrations alone.

5.2.1. Experimental Setup

As a first study of the effectiveness of using advice-operators, algorithm A-OPI is applied to the task of following a spatial trajectory with a Segway RMP robot (Fig. 5.3). First presented are the task and advice-operators for this domain, followed by empirical results.

FIGURE 5.3. Segway RMP robot performing the spatial trajectory following task (approximate ground path drawn in yellow).

5.2.1.1. Spatial Trajectory Following Task and Domain. Within this case-study, the learner is tasked with following a sinusoidal spatial trajectory. The robot senses its global position and heading within the world through wheel encoders sampled at 30Hz. The observations computed by the robot are 3-dimensional: x-position, y-position, and robot heading (x, y, θ). The actions predicted are 2-dimensional: translational speed and rotational speed (ν, ω).

Teacher demonsrations of the task are performed via joystick teleoperation of the robot. To challenge the robustness of A-OPI, the demonstration set was minimal and contained only *one* teacher execution (330 datapoints). A total of three policies are developed for this task:

Baseline Policy: Derived from the initial demonstration set.

Feedback Policy: Provided with policy corrections, via advice-operators.

More Demonstration Policy: Provided with more teacher demonstrations.

The learner begins practice with the Baseline Policy. The incorporation of teacher feedback on the learner executions updates this policy. This *execution-feedback-update* cycle continues to the satisfaction of the teacher, resulting in the final Feedback Policy. For comparison, a policy that instead receives further teacher demonstrations is also developed, referred to as the More Demonstration Policy. This policy is provided with additional demonstration traces so that its data set size approximately matches the size of the Feedback Policy dataset (two additional demonstrations provided).

The motion control advice-operators for this study were developed by hand and have a basic formulation (Table 5.1). Each operator adjusts the value of a single action dimension, by a static amount.

5.2.1.2. Evaluation. For this task, executions are evaluated for efficiency and precision. *Efficiency* is defined as faster execution time. *Precision* is defined as more closely following the target sinusoidal path, computed as the mean-squared Euclidean distance to this path.

	Operator	Parameter
0	Modify Speed, static, small (rot)	[cw ccw]
1	Modify Speed, static, large (rot)	[cw ccw]
2	Modify Speed, static, small (tr)	[dec inc]
3	Modify Speed, static, large (tr)	[dec inc]

TABLE 5.1. Advice-operators for the spatial trajectory following task.
[*Key*: (rot/tr)=$(rot/transl)ational$, $(c)cw$=$(counter)clockwise$], (dec/inc)=$(de/in)crease$]

The intent of this case-study is to explore behavior changes that might be enacted through advice-operators. We focus in particular on (i) improvement beyond the demonstrator's abilities and (ii) the synthesis of new behavior characteristics.

One behavior change seen in the empirical BC results was the potential for the learner to improve beyond the performance of the demonstration teacher. To investigate this, the performance of the Feedback Policy will be compared to the teleoperation demonstration that populated the initial dataset.

Another theoretical behavior change to explore is whether the presence of synthesized data also enables the synthesis of new behavior characteristics. Improvements under the BC algorithm were restricted to the data demonstrated by the teacher; critiquing just enabled the consideration of some points more strongly than others when making predictions. With A-OPI, however, new states and actions are made available to the policy, through the advice-operator data synthesis technique. To explore the ability of A-OPI to generate new behavior characteristics, the initial demonstration set provided to the learner is intentionally deficient in some behavior characteristics desired in the final policy. The goal is to generate these characteristics through the use of advice-operators. More concretely, the demonstration set contains point turns and, for smoothness considerations, it is desired that these be absent from executions with the final policy.

5.2.2. Results

Policy performance was found to improve with advising. The robot learner was able to improve its performance beyond the capabilites in its demonstration set, with faster execution times and more precisely executed positioning. Furthermore, the advisor was able to guide the learner to a policy with features that were not attainable within the original demonstration dataset; namely, the absence of point turns.

5.2.2.1. Improvement Beyond the Demonstration Set. Precision results are presented within Figure 5.4, as the average positional error between the execution path and the target sinusoid. Each bar represents the mean of 5 executions (1-standard deviation error bars). For comparison, the performance of the single teacher demonstration execution that populated the initial dataset is also shown (*Teacher Demo*, no error bars).

Advising improved execution precision (*Feedback*). This improvement was seen over the initial policy (*Baseline*), as well as over the teleoperation execution provided as a demonstration (*Teacher Demo*). This latter result indicates that advice-operators are indeed able to produce performance improvements beyond capabilities present in the demonstration set.

More teleoperation demonstrations also improved policy performance, with the policy derived from this set (*More Demonstration*, Fig. 5.4) outperforming the initial policy (*Baseline*). This more-demonstration policy was not able to perform as well as the A-OPI policy (*Feedback*) however. Unlike the feedback policy, providing more teleoperations also was not able to improve upon, or even match, the performance of the teacher demonstration recorded in the initial demonstration dataset.

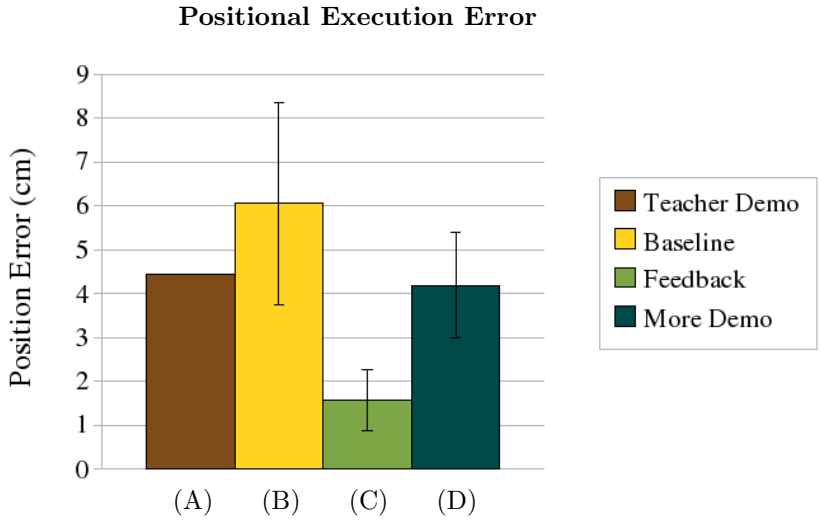

FIGURE 5.4. Execution trace mean position error compared to target sinusoidal trace.

Advising further resulted in a reduction in execution time, presented in Table 5.2. This is attributed to smoother executions without point turns (discussed in the following section), as well as to faster translational speeds, enabled through the use of the operators that change translational speed. Again, the advised policy (*Feedback*) outperforms those derived from only teleoperation (*Baseline, More Demo*), as well as the teacher execution within the demonstration set (*Demonstration*).

Teacher Demonstration	Baseline	Feedback	More Demonstration
10.89	11.35	**7.62**	10.8

TABLE 5.2. Average execution time (in seconds).

5.2.2.2. New Behavior Characteristics. Figure 5.5 shows examples of position traces during execution. For each, the target sinusoidal path (thin red line) and the execution trace (thick blue line) are shown. The letter labels attached to these executions indicate: (A) teacher demonstration execution, (B) initial policy execution (*Baseline Policy*), (C) policy execution after feedback (*Feedback Policy*), and (D) policy execution with more demonstration data (*More Demonstration Policy*). These labels correlate to the bars left to right in Figure 5.4.

Plot (A) shows the execution trace of the single teacher demonstration used to populate the demonstration set. Note that this execution does not perfectly follow the target sinusoidal path,

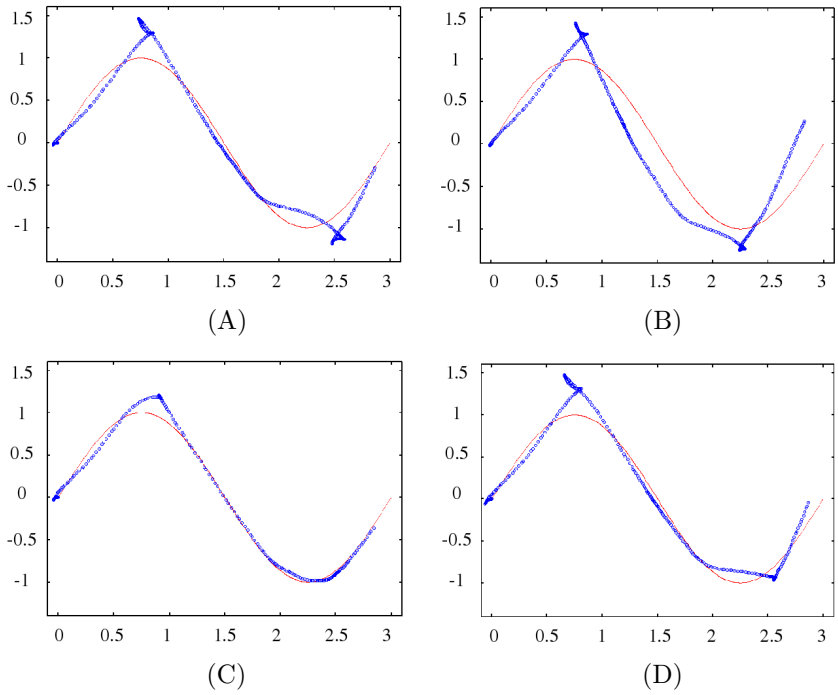

FIGURE 5.5. Example execution position traces for the spatial trajectory following task.

and that point turns were executed at the two points of greatest curvature (indicated by the trace doubling back on itself, a side-effect of the Segway platform's dynamic balancing).

A learner execution using the Baseline Policy derived from this demonstration set is shown in plot (B). This execution follows the target sinusoid more poorly than the demonstration trace, due to dataset sparsity, and similarly executes point turns. This learner execution was then advised, and the path re-executed using the modified policy. This process was repeated in total 12 times. Execution with the final Feedback Policy is shown in plot (C). This execution follows the target sinusoid better than the Baseline Policy execution *and* better than the teacher demonstration execution. Furthermore, this execution does not contain point turns.

Using the policy derived from more teleoperation data, shown in plot D, the learner is better able to follow the sinusoid than with the Baseline Policy; compare, for example, to the execution in plot (A). However, the learner is *not* able to smooth out its execution, and neither is able to eliminate point turns.

5.2.3. Discussion

In this section we first present our conclusions from this case study. We then discuss a relationship between the demonstration approach of this case study and our approach for scaffolding a simple behaviors into a policy able to accomplish a more complex behavior, which will be presented in Chapter 6.

5.2.3.1. Conclusions. Feedback offered through advice-operators was found to both improve learner performance, and enable the emergence of behavior characteristics not seen within

the demonstration executions. By contrast, the policy derived from strictly more teacher demonstrations was unable to eliminate point turns, as it was restricted to information provided through teleoperation executions. For this same reason, and unlike the advised executions, it was not able to reduce its positional error to below that of the demonstration execution (Fig. 5.4). These results support the potential of advice-operators to be an effective tool for providing corrective feedback that improves policy performance.

The correction formulation of the operators in this case-study was simple, adding only static correction amounts to single action dimensions. To more deeply explore the potential of advice-operators, next presented are the results of providing feedback on a more difficult robot task, and using a larger set of more complex and flexible operators.

5.2.3.2. Relation to Policy Scaffolding. As a small digression, here we pause to discuss the demonstration technique employed in this case study, and to compare this technique to our policy scaffolding algorithm, which was overviewed briefly in Section 2.3.3.2 and will be presented in full in Chapter 6. In this case study a simple policy, able to drive the robot in a straight line to a specified (x, y) location, was employed for the purposes of gathering demonstration data. Demonstration data for the full sinusoidal trajectory following task (Fig. 5.6, red curve) was produced as follows. First the advisor provided a sequence of five (x, y) subgoal locations (Fig. 5.6, left, end points of the blue line segments) to the simple policy. Next execution with the simple policy attaining these subgoals produced demonstration data for the sinusoidal task (Fig. 5.6, right, blue dots). This approach exploited similarities between policies that take similar actions to achieve different goal behaviors. Note that a full demonstration of the sinusoid following task was never provided by the teacher, though the teacher did guide the demonstration by providing the subgoal sequence to the simpler policy. Corrective feedback was then used to improve the performance of the new policy. More specifically, the teacher explicitly attempted to manifest new behavior through corrective feedback. The target new behavior was smooth sinusoidal trajectory following with an absence of pauses and point turns. The pauses and points turns were a consequence of sequential subgoal execution with the simpler policy, coupled with the dynamic balancing of this robot platform. The new behavior, sans pauses and point turns, was achieved (Fig. 5.5, C). In the case of this approach, feedback was used to shape a new policy behavior, by removing residual behavior characteristics seen during demonstration executions. Furthermore, like our policy scaffolding algorithm, feedback enabled the development of a more complex policy from a simpler existing policy.

5.3. Empirical Robot Implementation

This section presents results from the application of A-OPI to a more difficult motion task, again with a Segway RMP robot, and using a set of more complex operators. Policy modifications due to A-OPI are shown to improve policy performance, both in execution success and accuracy. Furthermore, performance is found to be similar or superior to the typical approach of providing more teacher demonstrations (Argall et al., 2008).

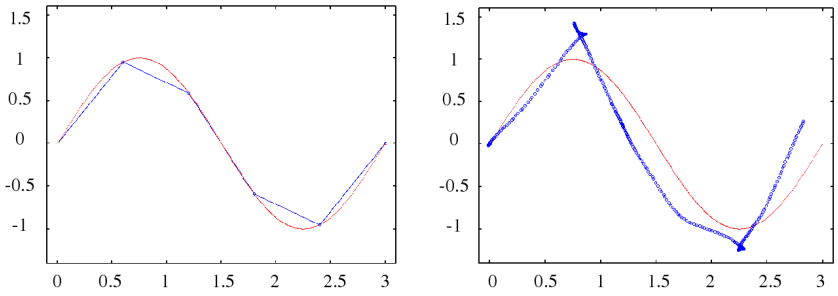

FIGURE 5.6. Using simple subgoal policies to gather demonstration data for the more complex sinusoid-following task.

5.3.1. Experimental Setup

Here the experimental details are presented, beginning with the task and domain, and followed by advice-operator and policy developments.

5.3.1.1. Robot Spatial Positioning Task and Domain. Empirical validation of the A-OPI algorithm is performed through spatial positioning with a Segway RMP robot. The spatial positioning task consists of attaining a 2D planar target position (x_g, y_g), with a target heading θ_g.

The Segway RMP platform accepts wheel speed commands, but does not allow access to its balancing control mechanisms. We therefore treat Segway control as a black box, since we do not know the specific gains or system parameter values. The inverted pendulum dynamics of the robot present an additional element of uncertainty for low level motion control. Furthermore, for this task smoothly coupled rotational and translational speeds are preferred, in contrast to turning on spot to θ_g after attaining (x_g, y_g). To mathematically define for this specific robot platform the desired motion trajectories is thus non-trivial, encouraging the use of alternate control approaches such as A-OPI. That the task is straightforward for a human to evaluate and correct further supports A-OPI as a candidate approach. While the task was chosen for its suitability to validate A-OPI, to our knowledge this work also constitutes the first implementation of such a motion task on a real Segway RMP platform.

The robot observes its global position and heading through wheel encoders sampled at 30Hz. Let the current robot position and heading within the world be represented as (x_r, y_r, θ_r), and the vector pointing from the robot position to the goal position be $(x_v, y_v) = (x_g - x_r, y_g - y_r)$. The observations computed for this task are 3-dimensional: squared Euclidean distance to the goal $(x_v^2 + y_v^2)$, the angle between the vector (x_v, y_v) and robot heading θ_r, and the difference between the current and target robot headings $(\theta_g - \theta_r)$. The actions are 2-dimensional: translational speed and rotational speed (ν, ω).

5.3.1.2. Motion Control Advice-Operators. Presented in Table 5.3 are the operators developed for this task. These operators adjust observation inputs (as in operator 0), single action dimensions by non-static amounts (as in operators 1-6) or multiple action dimensions by non-static amounts (as in operators 7,8). The amount of the non-static adjustments are determined as a function of the executed observation/action values. Note that these operators were developed by

hand, and not with our principled development approach presented in the previous chapter (Sec. 4.2). The operators are general to other motion tasks, and could be applied within other motion control domains.

	Operator	Parameter
0	Reset goal, recompute observation	
1	No turning	
2	Start turning	[cw ccw]
3	Smooth rotational speed	[dec inc]
4	No translation	
5	Smooth translational speed	[dec inc]
6	Translational [ac/de]celeration	[dec inc]
7	Turn tightness	[less more]
8	Stop all motion	

TABLE 5.3. Advice-operators for the spatial positioning task.
[*Key*: $(c)cw$=$(counter)clockwise$, (dec/inc)=$(de/in)crease$]

5.3.1.3. Policy Development. The set D is seeded with demonstrations recorded as the teacher teleoperates the robot learner (5 goal locations, with some repeat demonstrations for a total of 9 demonstrations, producing 900 data points). An initial policy is derived from this dataset.

Policy improvement proceeds as follows. A goal is selected (without replacement) from a practice set consisting of (x, y, θ) goals drawn uniformly within the bounds of the demonstration dataset $x \sim U\,(2.1, 2.4m)\,,\ y \sim U\,(-1.9, 2.1m)\,,\ \theta \sim U\,(-1.0, 2.1rad)$. The robot executes its current policy to attain this goal. The advisor observes this execution. If the execution is considered poor, the advisor offers policy improvement information. The policy is re-derived. Drawing a new goal then initiates another practice run, defined as a single *execute-feedback-update* cycle.

Three policies are developed using distinct techniques, differing in *what* is offered as policy improvement information. A total of four policies are therefore developed within this domain:

Baseline Policy: Derived from the initial demonstration set.

Feedback Policy: Provided with policy corrections, via advice-operators.

Feedback Hybrid Policy: Initially provided with more teacher demonstrations; later provided with policy corrections via advice-operators.

More Demonstration Policy: Provided with more teacher demonstrations.

These are referred to collectively as the *improvement policies*.

Policy performance is evaluated on a test set, consisting of 25 (x, y, θ) goals, randomly drawn from a uniform distribution within the bounds of the demonstration dataset. The test set is independent, and no executions associated with it receive policy improvement information. Policies are evaluated for accuracy and success. Here *accuracy* is defined as Euclidean distance between the

final robot and goal positions $e_{x,y} = \|(x_g - x_r, y_g - y_r)\|$, and the final robot and goal headings $e_\theta = |\theta_g - \theta_r|$. *Success* is defined generously as $e_{x,y} < 1.0\ m$ and $e_\theta < \frac{\pi}{2}\ rad$.

5.3.2. Results

Policy performance was found to improve with A-OPI feedback, in both execution success and accuracy. When compared to the approach of providing more teleoperation data, final improvement amounts were found to be similar or superior. Furthermore, this performance was achieved with a similar number of practice executions, but a significantly smaller final dataset.

For each policy improvement approach, the policy improvement phase was halted once performance on the test set no longer improved. The final Feedback, Feedback Hybrid and More Demonstration Policies contained data from 69, 68 and 60 advised and/or teleoperated executions, respectively (with the first 9 demonstrations for each attributed to seeding with the Baseline Policy).

5.3.2.1. Success and Accuracy Improvement. Table 5.4 presents the percent execution success of each policy on the independent test set. When compared to the Baseline Policy, all policy improvement approaches display increased success. Both of the feedback policies additionally achieve higher success than the Demonstration Policy.

Baseline	Feedback	Feedback Hybrid	More Demonstration
32	88	**92**	80

TABLE 5.4. Percent Success, Test Set

Figure 5.7 plots the average position and heading error on the test set goals, for each policy. For positional error, all improvement policies display similar performance, which is a dramatic improvement over the Baseline Policy. For heading, the Feedback Policy reduces more error than the Feedback Hybrid Policy, with both showing marked improvements over the Baseline Policy. By contrast, the More Demonstration Policy displays *no* overall improvement in heading error. That heading error proved in general more difficult to improve than positional error is consistent with our prior experience with this robot platform, which is highly sensitivity to rotational dead reckoning error accumulation.

The iterative nature of the A-OPI policy development approach (Section 5.3.1.3) results in intermediate policies being produced. A sampling of these policies were also evaluated on the test set, to mark improvement progress for each policy improvement technique. Figure 5.8 shows the average position and heading error, on the test set goals, for these intermediate policies. Superior heading error is consistently produced by corrective feedback, throughout policy improvement. Greater positional error improvement is initially seen through demonstration. Corrective feedback reduces positional error more slowly, but does eventually converge to the demonstration level.

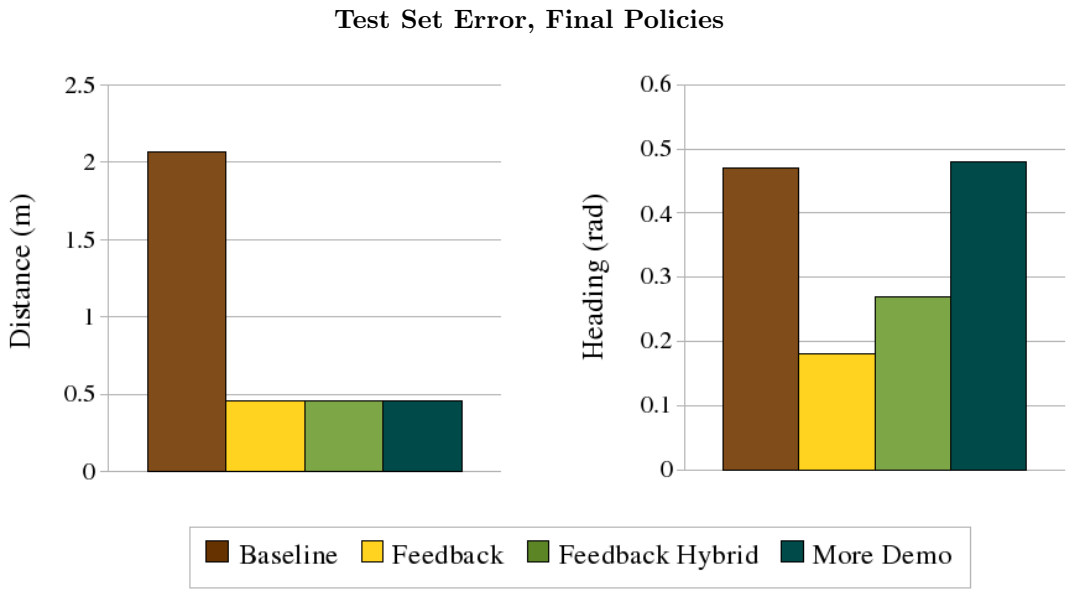

FIGURE 5.7. Average test set error on target position (left) and heading (right), with the *final* policies.

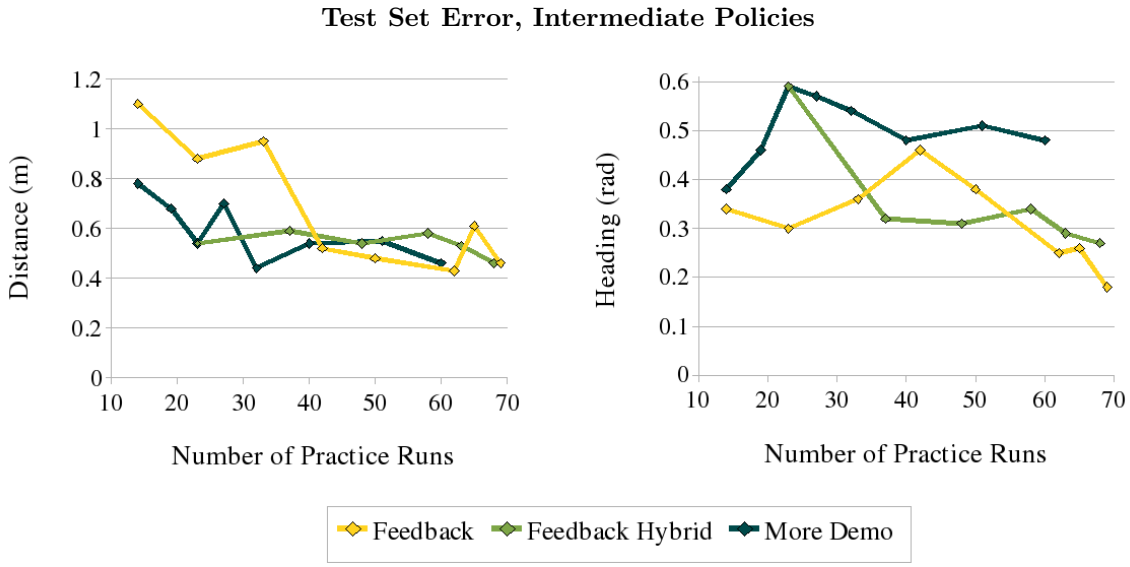

FIGURE 5.8. Average test set error on target position (left) and heading (right), with *intermediate* policies.

The additional demonstration phase of the Feedback Hybrid Policy resulted from seeding with an intermediate version of the More Demonstration Policy (23 demonstrations). Following this seeding, only corrective feedback is offered. The Feedback Hybrid Policy thus initially also displays the superior reduction in positional error, and inferior reduction in heading error, of the More Demonstration Policy. This is followed by substantial reductions in heading error through corrective feedback.

5.3.2.2. More Focused Datasets. How many *datapoints* are added with each execution varies greatly depending upon whether the execution receives corrective feedback or more demonstration (Fig. 5.9). This is because, in contrast to teleoperation, only subsets of a corrected execution are added to D; in particular, only those execution points which actually receive corrections. States visited during good performance portions of the student execution are *not* redundantly added to the dataset. In this manner, the final policy performances shown in Figure 5.7 are achieved with much *smaller* datasets for both feedback policies, in comparison to the More Demonstration Policy. Note that the results of Figure 5.8 are plotted against the number of *practice runs* contributing to the set D, and not the number of *datapoints* in D.

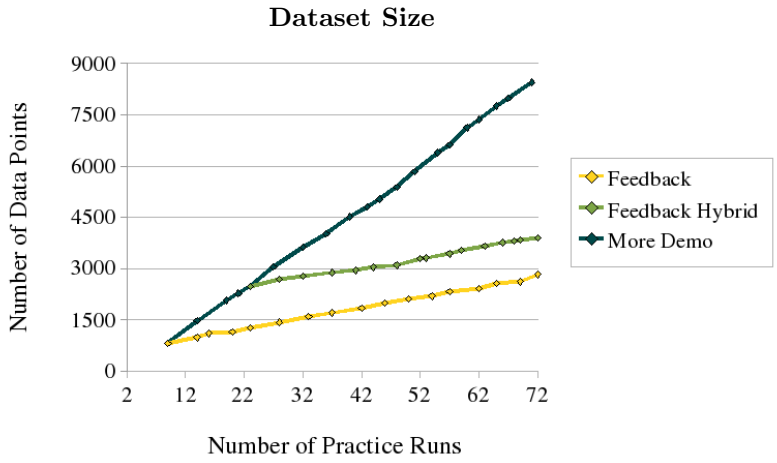

FIGURE 5.9. Dataset size growth with demonstration number.

5.4. Discussion

This section provides a discussion of the A-OPI algorithm and the presented empirical implementation. Noteworthy conclusions that may be drawn from this work are detailed first, followed by a discussion of future directions for the A-OPI algorithm specifically, and directions for teacher feedback in general.

5.4.1. Conclusions

Discussion in this section recaps some of the advantages afforded from the use of advice-operators as tools for the correction of continuous-valued, frequently sampled, policies.

5.4.1.1. Advice-Operator Selection and Application. These empirical results confirm that a human teacher was able to effectively advise within *continuous* action spaces using the A-OPI algorithm. This occurred without requiring that the teacher provide the actual continuous-values for the corrections. Additionally, the teacher was able to select operators without requiring value-based details of the execution (e.g. speed). Both of these aspects result from the fact that the robot, and not the human, applies the advice-operator.

One goal when developing the motion control operators of this domain was to exploit the respective strengths of a robot learner and human teacher. Humans in general are good at providing qualitative evaluations of performance quickly and accurately (for example, "drive faster"), but are less adept at determining exact quantitative measures (for example, "drive exactly 1.6 m/s"). By contrast, the robot has access to its internal state including current driving speeds, and is able to reason quickly and accurately about quantity.

5.4.1.2. An Observation Advice-Operator. The operators in this work presented one example of an observation-modifying advice-operator: the operator "Reset goal, recompute observation". The observations computed for the spatial positioning task encoded the target final position and heading (x_g, y_g, θ_g). This operator functioned by resetting the target as the *actual* final position and heading (x_f, y_f, θ_f) attained during an execution. The operator then recomputed all of the observations along the execution path as though this final position/heading (x_f, y_f, θ_f) had been the true target position/heading. This operator was applied to executions that exhibited a path that performed well according to the measures of speed and path smoothness, but poorly according to the measure of accurately attaining the target position/heading. The intent of this operator was to remember such executions as good demonstrations of attaining the position/heading (x_f, y_f, θ_f).

As mentioned in the previous chapter (Sec. 4.4.2), the general idea of this technique is to encode the target value for a goal - here the target (x, y, θ) position - into the observation computation, and develop an operator that modifies the state-action mapping by resetting the target value to the executed value. The single operator presented here constitutes only a small exploration of observation-modifying advice-operators, and a more complete investigation is identifies as a rich area for future research.

5.4.1.3. Advising is a Reasonable Task. Teacher feedback under A-OPI consists of selecting segments of an execution requiring improvement and indicating an advice-operator to apply over this segment. In this manner, the selection of a *single* execution segment and *single* advice-operator from a finite list enables the *continuous-valued* correction of *multiple* execution points. This quality makes A-OPI suitable for domains with continuous actions sampled at high frequency, such as low-level motion control domains. Without such a mechanism, to provide continuous-valued corrections for a large number of executions points would be highly taxing on a human teacher. With A-OPI, this becomes a reasonable task for the feedback teacher.

Our feedback teacher was able to provide advice quickly, for a variety of reasons. One reason was the ability to advise multiple execution points at once, through the F3MRP segment selection technique. Another reason was that the A-OPI algorithm is robust to fuzzy selections of execution points, and the cause of this is twofold. First, since our employed regression techniques treat each datapoint independently, a point added to the dataset does not depend on whether nearby points from the execution were also added. Second, since the policy is sampled rapidly and exists within a continuous state-action space, physical constraints on the robot and world mandate that adjacent execution points are similar in both observation and action values; our robot has the ability neither to teleport nor to infinitely accelerate. This similarity makes it unlikely that a correction would be appropriate for one point but inappropriate for its neighbor.

Chapter 3 discussed the benefit of having a *human* teacher provide the critiquing feedback (Sec. 3.3.1.2). While the automation of simple overall evaluations of an execution was noted to be feasible. The ability of a human to address the issue of *credit assignment* was highlighted as a strength. When providing corrective advice, the human teacher again addresses the issue of which portions of an execution receive feedback. Here, however, the *evaluation* tied to the feedback additionally is not simple or straightforward. That is, to provide a correction is more complex than providing an overall binary evaluation of success. Not only are more complex evaluation measures are employed, for example if a turn was "tight" enough, but based on these evaluations a correction must be selected. Advice-operators simplify this process, by offering a finite list for this selection. Even with this simplification, however, it is unlikely that some form of automated feedback would be able to provide a correction, again highlighting the strength of a human feedback provider.

5.4.2. Future Directions

This section identifies potential areas for future research with the A-OPI algorithm. The first option considered is to interleave demonstration and corrective feedback. We then consider other formulations for corrective feedback.

5.4.2.1. Policy Improvement from Advice and Demonstration. Two improvement approaches, advising and more teleoperation, were considered by the hybrid policy. More specifically, the hybrid policy was seeded with an intermediate version of the More Demonstration Policy and then provided with corrective feedback, in an attempt to exploit the strengths of each technique. This approach was motivated from noting that (i) more demonstration quickly reduced initial positional error but struggled to reduce heading error, and (ii) corrections gradually reduced both types of error overall. The supposed reason for this difference is that demonstration is more effective for reaching previously undemonstrated areas of the state space; by contrast, corrections can only be offered on states already visited, and therefore already reachable with the current policy. Correspondingly, advice is more effective at correcting visited states; by contrast, it is difficult for the teacher to revisit real world states through teleoperation, especially when they lie in the middle of a motion trajectory.

These observations underline the benefit of providing both corrections *and* more demonstration. Advice-operators are most effective when correcting an already adequate policy; using advice to iteratively modify action selection in order to creep towards a distant desirable part of the state-space is inefficient and not recommended. By contrast, demonstration is most effective for general state-space population; using teleoperation in an attempt to exactly revisit states and demonstrate alternate behaviors is similarly inefficient and not recommended. The approaches, in short, compliment each other.

In this implementation of the hybrid policy, more demonstration and corrective feedback happened in distinct phases. While some demonstration does need to occur first, to provide a baseline policy for advising to build upon, demonstration does not need to occur *only* first. One direction for future work could interleave providing corrections and more demonstration. Alternately, a sufficiently populated demonstration set could be provided at the start. This is fundamentally different,

however, to providing more demonstration in response to *student* executions. A sufficiently populated dataset requires prior knowledge of all states the student will visit; generally impossible in continuous-state real worlds.

5.4.2.2. Other Options for Corrective Advice. Advice-operators offer a formulation for corrective advice within continuous state-action domains. The key contribution of this technique is the functional formulation of advice-operators, where mathematical computations are performed on execution data points. The functional formulation makes providing continuous-valued corrections reasonable, and is less appropriate for providing discrete-valued corrections. To provide a fractional correction to a high-level, discrete action, behavior policy is less immediate; for example, to increase a *pick-up box* action by 20% does not make sense. This does not mean, however, that advice-operators are not applicable within discrete action domains; it is just not appropriate for their formulation to fragment actions. One appropriate formulation, for example, could adjust the number times a repetitive discrete action is executed, such as the number of discrete steps taken by a humanoid robot. Advice-operators therefore may be appropriate for some discrete-action domains, but not necessarily for all.

Many other formulations for corrective feedback are also possible, and we present some example formulations here. One already noted is the selection of a correct action from a discrete set, within discrete-action domains. A higher-level example is a behavior that operates by selecting between multiple policies, as will be the case with our policy scaffolding algorithm presented in the next chapter. A correction within this scenario could indicate the appropriate policy to select (note that this conjecture is hypothetical, and such feedback is not included within our policy scaffolding algorithm). One hypothetical policy formulation could involve selection between multiple weightings on the observation space, to vary the importance given to different observation features within distinct areas of the state space. Corrective feedback in this domain could correct the selection of observation weight.

5.5. Summary

This chapter has introduced Advice-Operator Policy Improvement (A-OPI) as an algorithm that improves the performance of a policy learned through demonstration via the use of corrective teacher feedback. Here teacher feedback results in new, synthesized, demonstration data, that does not depend on teacher demonstrations nor revisiting past execution states.

Empirical validation of A-OPI was performed on a Segway RMP robot performing planar motion tasks. The first implementation was a case study that used a simple set of advice-operators to correct a spatial trajectory following task. Empirical results showed improvements in precision and efficiency, and both beyond the abilities of demonstrator. Corrective advice was also found to enable the emergence of novel behavior characteristics, absent from the demonstration set. The second implementation used a complex set of advice-operators to correct a more difficult spatial positioning task. Empirical results showed performance improvement, measured by success and accuracy. Furthermore, the feedback policies displayed similar or superior performance when compared to a policy developed from exclusively more teleoperation demonstrations. The A-OPI approach also produced

noticeably smaller datasets, without a sacrifice in policy performance, suggesting that the datasets were more focused and contained less redundant data.

Key features of the A-OPI approach facilitate the ability to provide corrections within low-level motion control domains. In particular, the segment selection technique allows multiple actions to be corrected by a single piece of advice. The advice-operator mechanism translates high-level advice into continuous-valued corrections. Selection of an operator is reasonable for a human to perform, since the set of advice-operators is discrete; by contrast, selection from an infinite set of continuous-valued corrections would not be reasonable.

Future directions for the A-OPI algorithm were identified to include interleaving the two policy improvement techniques of corrective advice and more teleoperation demonstrations. The formulation of corrective advice within other domains was also discussed.

CHAPTER 6

Policy Scaffolding with Feedback

Given the challenge of developing robust control policies, the ability to reuse and incorporate existing policies designed to address related tasks is a practical feature for any policy learning system. While LfD has seen success on a variety of robotics applications, as tasks become more complex, demonstration can become more difficult. One practical extension of the LfD approach thus is to incorporate simpler behaviors learned from demonstration into larger tasks, especially if such tasks are too complex to demonstrate in full. We introduce *Feedback for Policy Scaffolding (FPS)* as an algorithm that builds a complex policy from component behaviors learned from demonstration.

One crucial issue when building a policy from primitives is how to effectively incorporate the existing behaviors into the new policy. Whether the behaviors are to be composed or sequenced, behaviors must be linked and how this linking occurs is a key design decision. The FPS algorithm takes the approach of aiding this process with human feedback. The feedback types considered by this algorithm are binary flags of good performance and corrective advice, provided through the advice-operator framework (Ch. 4). Advice under this formulation does not require revisiting states to provide feedback, which is particularly attractive for complex tasks that may be inconvenient or infeasible to demonstrate completely.

This chapter contributes an algorithm that extends the A-OPI algorithm by selecting between multiple policies and incorporating an additional feedback phase. In phase one of the FPS algorithm, multiple policies are developed from demonstrated motion primitives, using a slightly modified version of the A-OPI algorithm. During phase two, a single complex policy is then derived from these primitives, and executions with the complex policy on a novel task are evaluated. By providing corrections on this execution, the FPS algorithm develops a policy able to execute this more complex, undemonstrated task. This development is achieved without ever requiring a full demonstration of the novel task.

We validate FPS within a simulated motion control domain, where a robot learns to drive on a racetrack. A policy built from demonstrated motion primitives and human feedback is developed and able to successfully execute a more complex, undemonstrated task. Advice is shown to improve policy performance within both feedback phases: when offered in response to individual motion primitive executions, as well as executions with the complex task policy. Finally, comparisons to an

exclusively demonstration-based approach show the FPS algorithm to be more concise and effective in developing a policy able to execute the complex task.

The following section introduces the FPS algorithm, providing details of behavior building through teacher feedback and algorithm execution. Section 6.2 presents an empirical implementation of FPS, including descriptions of the task and domain. Experimental results, from both feedback phases, are also provided. The conclusions of this work are presented in Section 6.3, along with directions for future research.

6.1. Algorithm: Feedback for Policy Scaffolding

This section presents the Feedback for Policy Scaffolding algorithm. This section begins by motiving why a more complex policy is built from demonstration and teacher feedback. This is followed with a discussion of the algorithm execution details.

6.1.1. Building Behavior with Teacher Feedback

We begin with a discussion of some reasons why one might use teacher feedback to build a policy that produces a more complex behavior, rather than demonstrating that behavior outright. One consideration is that as behaviors become more complex, demonstrating the behavior in full can become more difficult. In this case, the teacher may be able to demonstrate behavior primitives for the task but not the task in full, or be able to provide higher quality demonstrations for primitives than for the full task. Under our algorithm, feedback may be used to improve, possibly even to enable, the policy at transition points that link demonstrated behavior primitives. In doing so, it enables expression of the full task behavior, without requiring its full demonstration.

A second consideration is that reaching all states encountered during task execution can become increasingly difficult as tasks become more complex. States may be infeasible, inconvenient or undesirable to reach for a variety of reasons that only compound as task complexity increases. A drawback to demonstration is the need to visit such states in order to provide task execution information in that area. One of the notable benefits of providing feedback under the F3MRP framework is that states do *not* need to be revisited to provide execution feedback.

A final consideration is that the demonstrated motion primitives may provide a base for multiple complex behaviors. Through the reuse of these primitives, the effort required to develop policies for the complex behaviors reduces.

6.1.2. Feedback for Policy Scaffolding Algorithm Execution

Algorithms 4 and 5 present pseudo-code for the FPS algorithm. This algorithm is founded on the baseline feedback algorithm of Chapter 2. The distinguishing details of FPS lie in the types and applications of teacher feedback (Alg. 1, lines 17-18) and the development of multiple primitive policies. The FPS algorithm furthermore includes of a second phase of learner practice during which the primitive policies are scaffolded together.

Execution occurs in two phases. Given a set Ξ of n primitive behaviors, the first phase develops a policy π_{ξ_j} for each primitive behavior $\xi_j \in \Xi, j = 1 \ldots n$, producing a set of policies Π (Alg. 4).

The second phase develops a policy for a more complex behavior, building on the primitive policies $\pi_{\xi_j} \in \Pi$ (Alg. 5).

6.1.2.1. Phase 1: Primitive Policy Development. The development of each primitive policy begins with teacher demonstration. For each developed primitive ξ_j, the teacher demonstration phase produces an initial dataset D_{ξ_j} exactly as during the demonstration phase of the baseline feedback algorithm (Alg. 1, lines 1-8); for brevity the details of teacher execution are omitted here. The learner execution portion of the practice phase proceeds similarly to the baseline algorithm, recording the policy predictions and robot positions respectively into the execution traces d and tr (Alg. 4, lines 6-17). The only difference here is that the data support τ^t of the regression prediction (Sec. 2.3.2.1) is recorded into tr as well, for use by the F3MRP framework for the visual display of the robot execution path.

Algorithm 4 *Feedback for Policy Scaffolding: Phase 1*

1: Given $\mathbf{D}$
2: *initialize* $\Pi \leftarrow \{\ \}$
3: **for all** behavior primitives $\xi_j \in \Xi$ **do**
4: *initialize* $d \leftarrow \{\},\ tr \leftarrow \{\}$
5: *initialize* $\pi_{\xi_j} \leftarrow \texttt{policyDerivation}\big(D_{\xi_j}\big)\ ,\ D_{\xi_j} \in \mathbf{D}$
6: **while** *practicing* **do**
7: *initialize* $d \leftarrow \{\},\ tr \leftarrow \{\}$
8: **repeat**
9: *predict* $\{\ \mathbf{a}^t, \tau^t\ \} \leftarrow \pi_{\xi_j}\left(\mathbf{z}^t\right)$
10: *execute* $\mathbf{a}^t$
11: *record* $d \leftarrow d \cup \left(\mathbf{z}^t, \mathbf{a}^t\right),\ tr \leftarrow tr \cup \left(x^t, y^t, \theta^t, \tau^t\right)$
12: **until** done
13: *advise* $\{\ \digamma, \Phi\ \} \leftarrow \texttt{teacherFeedback}(\ tr\ )$
14: *apply* $\hat{d}_\Phi \leftarrow \texttt{applyFeedback}(\ \digamma, \Phi, d\ )$
15: *update* $D_{\xi_j} \leftarrow D_{\xi_j} \cup \hat{d}_\Phi$
16: *rederive* $\pi_{\xi_j} \leftarrow \texttt{policyDerivation}(D_{\xi_j})$
17: **end while**
18: *add* $\Pi \leftarrow \Pi \cup \pi_{\xi_j}$
19: **end for**
20: **return** Π

During the teacher feedback portion of the practice phase, the teacher first indicates a segment Φ, of the learner execution trajectory requiring improvement. The teacher further indicates feedback $\digamma$, which may take two forms (line 13). The first is a binary credit, $\digamma \equiv c$, to indicate areas of good performance. The second is an advice-operator, $\digamma \equiv op$, to correct the execution within this segment.

The teacher feedback is then applied across all points recorded in d and within the indicated subset Φ (line 14). Both feedback forms produce new data. In the case of feedback $\digamma \equiv c$, data d_Φ from the selected segment is added to D_{ξ_j} as executed, without modifications ($\hat{d}_\Phi = d_\Phi$). In the case of feedback $\digamma \equiv op$, each selected point is corrected by the advice-operator, as in the A-OPI algorithm (Alg. 3, lines 11-17). The result is feedback-modified data $\hat{d}_\Phi$, which is added to dataset D_{ξ_j} (line 15). Policy π_{ξ_j} for primitive ξ_j is then rederived from this set.

Note the similarity between this Phase 1 portion of the FPS algorithm and the A-OPI algorithm (Alg. 3, Ch. 5). The differences lie in the development of *multiple* policies and the use of the positive credit feedback form in addition to advice-operators. This implementation also takes advantage of the full F3MRP framework, with the visual display of data support τ.

6.1.2.2. Phase 2: Policy Scaffolding. The development of the complex policy builds on the primitive policies in the set Π. This development therefore does *not* begin with teacher demonstration of the complex task. Two distinguishing features of second phase of the FPS algorithm are (i) the selection between multiple policies and (ii) need to select a dataset to receive any new data produced as a result of learner executions.

Figure 6.1 presents a schematic of our scaffolding approach, where dashed lines indicate repetitive flow and therefore execution cycles that are performed multiple times. In comparison to the schematic of the baseline feedback algorithm (Fig. 2.4), note in particular that the learner policy selects between multiple policies and the need to select a dataset prior to a dataset update.

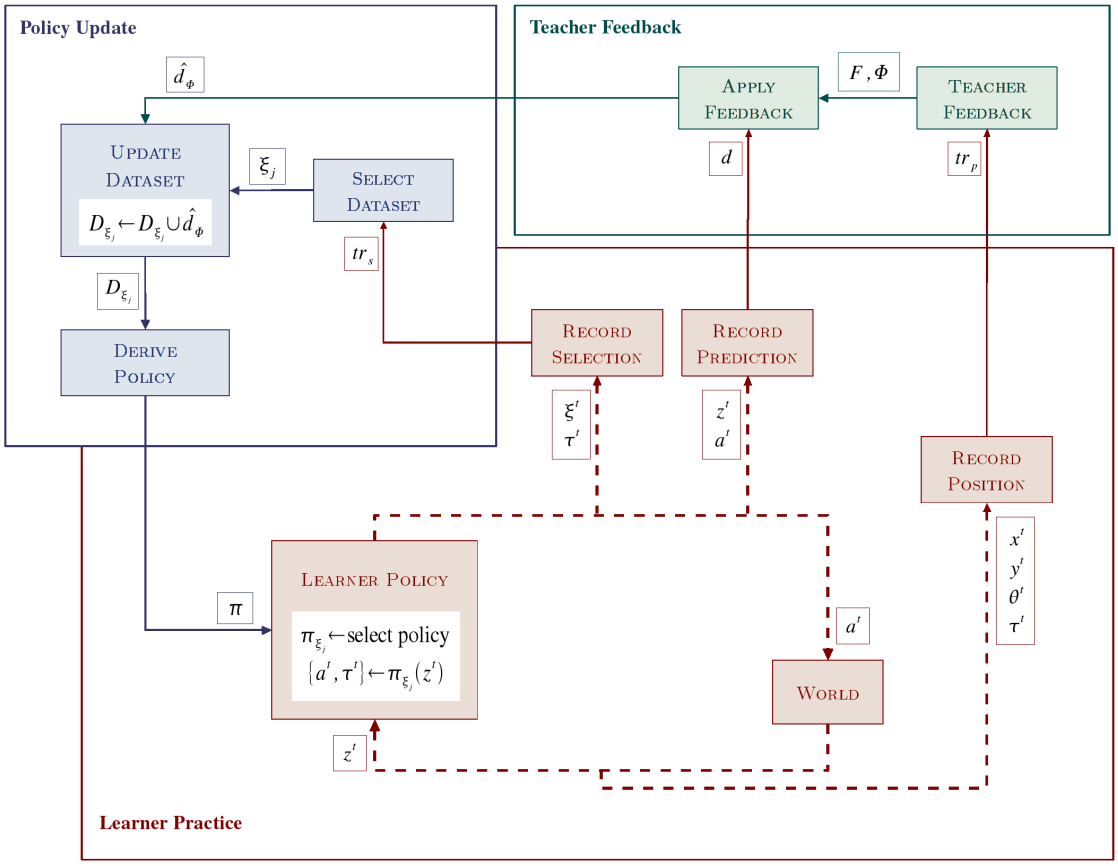

FIGURE 6.1. Policy derivation and execution under the Feedback for Policy Scaffolding algorithm.

Phase 2 begins with the initialization of more demonstration datasets (Alg. 5, line 2). Specifically, n empty datasets are generated, each associated with one primitive policy. Notationally, let

Algorithm 5 *Feedback for Policy Scaffolding: Phase 2*

1: Given $\Pi, \mathbf{D}$
2: *initialize* $D_{\xi_{i=(n+1)\ldots 2n}} \leftarrow \{\ \}$
3: **while** *practicing* **do**
4: *initialize* $d \leftarrow \{\}$, $tr_s \leftarrow \{\}$, $tr_p \leftarrow \{\}$
5: **repeat**
6: *select* $\pi_{\xi_j} \leftarrow$ `policySelection`$(\ \mathbf{z}^t\ )$, $\pi_{\xi_j} \in \Pi$
7: *predict* $\{\ \mathbf{a}^t, \tau^t\ \} \leftarrow \pi_{\xi_j}(\mathbf{z}^t)$
8: *execute* $\mathbf{a}^t$
9: *record* $d \leftarrow d \cup (\mathbf{z}^t, \mathbf{a}^t)$, $tr_s \leftarrow tr_s \cup (\tau^t, \xi^t = \xi_j)$, $tr_p \leftarrow tr_p \cup (x^t, y^t, \theta^t, \tau^t)$
10: **until** done
11: *advise* $\{\ \digamma, \Phi\ \} \leftarrow$ `teacherFeedback`$(\ tr_p\ )$
12: **for all** $\varphi \in \Phi$, $(\mathbf{z}^\varphi, \mathbf{a}^\varphi) \in d, (\tau^\varphi, \xi^\varphi) \in tr_s$ **do**
13: *apply* $\hat{d}_\varphi \leftarrow$ `applyFeedback`$(\ \digamma, \mathbf{z}^\varphi, \mathbf{a}^\varphi\ )$
14: *select* $D_{\xi_\iota} \leftarrow$ `datasetSelection`$(\ \tau^\varphi, \xi^\varphi\ )$, $D_{\xi_\iota} \in \mathbf{D}$
15: *update* $D_{\xi_\iota} \leftarrow D_{\xi_\iota} \cup \hat{d}_\varphi$
16: *rederive* $\pi_{\xi_\iota} \leftarrow$ `policyDerivation`(D_{ξ_ι}), $\pi_{\xi_\iota} \in \Pi$
17: **end for**
18: **end while**
19: **return** Π

new data set $D_{\xi_{i+n}}$ be associated with existing primitive dataset D_{ξ_i}, resulting in a total of $2n$ datasets $D_j \in \mathbf{D}, j = 1 \cdots 2n$. Colloquially, call dataset D_{ξ_i+n} the *feedback* dataset associated with primitive dataset D_{ξ_i}. Some of the new feedback-generated data will be added to these feedback datasets; this process is described further in the details of feedback incorporation. Each feedback dataset is treated identically to the primitive datasets during policy execution. That is, a policy is derived from each feedback dataset, and these policies are considered for selection during execution of the more complex policy.

During the learner execution portion of the practice run (lines 5-10), the learner executes with the complex policy. At each decision cycle, the operation of this policy proceeds in two steps. The first step is to select between all contributing policies π_{ξ_j} based on observation $\mathbf{z}^t$ (line 6); the details of this selection are provided in Section 6.1.3.1. The second step is to predict action $\mathbf{a}^t$ according to $\pi_{\xi_j}(\mathbf{z}^t)$, with prediction support τ^t (line 7). The action is executed. A variety of information is then recorded (line 9). The predicted action is recorded in the prediction trace d, along with observation $\mathbf{z}^t$. The measure of data support τ^t for the prediction is recorded, along with the tag ξ_j indicating the selected policy, within the selection trace tr_s, which will be utilized during the selection of which dataset receives feedback-modified data. The learner position and heading within the world (x^t, y^t, θ^t) is recorded along with the prediction support τ^t into the position trace tr_p, for use during the F3MRP visual display for the teacher.

During the teacher feedback and policy update portion of a practice run (lines 11-17), the advisor first indicates a segment Φ, of the learner execution, and provides feedback $\digamma$ for that segment. As in Phase 1, two feedback forms are available: binary credit, $\digamma \equiv c$, and corrective advice, $\digamma \equiv op$.

The teacher feedback is then applied across all points within the indicated subset Φ. For each selected execution point, indexed as $\varphi \in \Phi$, the first step is to apply feedback $\digamma$ to the point $(\mathbf{z}^{\varphi}, \mathbf{a}^{\varphi})$ recorded in d (line 13). The next step is to select one of the datasets to receive this data (line 14). Specifically, dataset D_{ξ_ι} is selected, based on the recorded data support τ^{φ} and tag ξ^{φ} associated with the execution point; the details of this selection are provided in Section 6.1.3.2. The selected dataset is updated with the feedback-modified data point (line 15) and the policy π_{ξ_ι} associated with this dataset is rederived (line 16).

6.1.3. Scaffolding Multiple Policies

One key factor when building a policy from simpler behaviors is how to select between the behaviors. A second key factor is how to incorporate teacher feedback into the built-up policy. The design of each of these factors wtihin the FPS algorithm are discussed here. Later sections (6.3.2.1 and 6.3.2.2 respectively) discuss a few of the iterative stages from the development of these factors.

6.1.3.1. Selecting Primitive Policies. Primitive selection assumes that primitives occupy nominally distinct areas of the observation-space. This assumption relies on the state observation formulation capturing aspects of the world that are unique to the demonstrations of each primitive policy. For example, in the following section two of the primitives developed for a racetrack driving task are *turn left* and *turn right*. Demonstrations of the *turn left* primitive occur in areas of the track that curve to the left, and of *turn right* in areas that curve to the right. Observations are formulated to incorporate a notion of track curvature, and so the demonstrations for each primitive occupy distinct areas of the observation-space.

Primitive selection then is treated as a classification problem. For each primitive ξ_j, a kernelized distance $\phi(\mathbf{z}^t; \mathbf{z}_i) = e^{|\mathbf{z}_i - \mathbf{z}^t|\Sigma^{-1}|\mathbf{z}_i - \mathbf{z}^t|}$ (Eq. 2.2) is computed between query point $\mathbf{z}^t$ and each point $\mathbf{z}_i \in D_{\xi_j}$. The distance computation is Euclidean and the kernel Gaussian. The parameter Σ^{-1} is a constant diagonal matrix that scales each observation dimension and embeds the bandwidth of the Gaussian kernel. This parameter is tuned through 10-folds Cross Validation to optimize the least squared error on primitive label classification. Weight w_{ξ_j} for policy ξ_j is computed by summing the k largest kernel values $\phi(\mathbf{z}^t, :)$; equivalent to selecting the $k-$nearest points in D_{ξ_j} to query $\mathbf{z}^t$ ($k = 5$). The primitive ξ_j with the highest weight w_{ξ_j} is then selected.

6.1.3.2. Incorporating Teacher Feedback. The FPS algorithm produces data from teacher feedback on policy executions with *multiple* primitive policies. A variety of options exist for how to then incorporate this data into the multiple underlying datasets.

To begin, let us establish two ideas. First, this work assumes that in state-space areas covered by the dataset of a particular primitive, the behavior of this primitive matches the intended behavior of the more complex policy. If this is not the case, and the two behaviors conflict, then that primitive should not be incorporated into the complex policy in the first place. Second, the feedback forms considered in this chapter both produce new data. The new data derives from learner executions, such that every new datapoint $\hat{d}_{\varphi} = (\hat{\mathbf{z}}^t, \hat{\mathbf{a}}^t)$ derives from an execution point $(\mathbf{z}^t, \mathbf{a}^t)$. Each learner execution point is predicted by a single primitive policy, as discussed in the previous section.

Two factors determine into which dataset a new datapoint $\hat{d}_\varphi$ is added: the policy ξ^t that predicted the execution point $(\mathbf{z}^t, \mathbf{a}^t)$, and the measure of data support τ^t for that prediction. In particular, if the policy that made the prediction is a primitive policy, the point is added to its dataset *if* the prediction had strong data support. Otherwise, the point is added to the *feedback* dataset associated with primitive ξ^t (Sec. 6.1.2.2). By contrast, data generated from execution points predicted by a feedback policy are automatically added to its dataset, regardless of dataset support.

This idea of prediction data support is the same as that used by the F3MRP feedback interface when providing the teacher with a visual representation of the measure of data support for predictions, as described in Chapter 2 (Sec. 2.3.2.1). Each primitive ξ_j therefore has an associated Poisson distribution modeling the distribution of $1-$NN distances between points in D_{ξ_j}, with mean $\mu_{\xi_j} = \lambda_{\xi_j}$ and standard deviation $\sigma_j = \sqrt{\lambda_{\xi_j}}$. We set by hand the threshold on strong prediction support to $\tau_j = \mu_{\xi_j} + 5\sigma_{\xi_j}$, based on empirical evidence. Thus a prediction made by policy π_{ξ_j} for query point $\mathbf{z}^t$ with distance $\ell_{\mathbf{z}^t}$ to the nearest point in D_{ξ_j} is classified as strongly supported if $\ell_{\mathbf{z}^t} < \tau_{\xi_j}$ and weakly supported otherwise.

The motivation behind this approach is to avoid adding data to the primitive datasets that conflicts with the primite behavior. We assume that observations nearby to points included in a primitive dataset, and which therefore were strongly supported during their associated action prediction, are associated with behavior that does not conflict with the intended behavior of the primitive dataset.

6.2. Empirical Simulation Implementation

This section presents results from the application of the FPS algorithm to a simulated motion control domain. Motion control primitives are learned for a racetrack driving task, and are scaffolded into a policy able to perform the full task. Furthermore, all FPS policies are found to outperform those developed under the typical approach of providing more teacher demonstrations.

6.2.1. Experimental Setup

Here we present the experimental details, beginning with the task and domain, and followed by a description of the various policies developed for empirical evaluation.

6.2.1.1. Racetrack Driving Task and Domain. The domain chosen to validate the FPS policy building algorithm is a simulated differential drive robot within a racetrack driving domain. The task consists of the robot driving along a racetrack, with two failure criteria: the robot either stops moving or crosses a track boundary. The dimensions and parameters set in this simulated domain derive from real world execution with a Segway RMP robot.

In particular, the robot is assumed to observe the world through a single monocular camera and wheel encoders, and state observations are sampled at 30Hz. The camera is forward facing and observes track borders within its field of view $(130°, 5m)$, and represents the track as a set of points, with each point corresponding to a single image pixel projected into the ground plane. Track observations and robot position updates are subject to 1% Gaussian noise, and the robot execution speeds $([-5, 5]\frac{rad}{s}, [0, 3]\frac{m}{s})$ and accelerations $([-10, 10]\frac{rad}{s^2}, [-10, 10]\frac{m}{s^2})$ are limited. To simplify the

interactions between the robot and the track boundaries, the robot is represented as a point and the track width ($1.5m$) is shrunk by moving each border towards the centerline by an amount equal to the radius of the robot ($0.3m$).

The robot is controlled by setting target translational and rotational speeds. Demonstrations are performed via human teleoperation, which consists of incrementally decreasing or increasing the translational and rotational speeds as the robot moves along the racetrack. The motion control advice-operators developed for this task were developed using our contributed approach for principled action-operator development (Sec. 4.2), and are presented in Table 6.1. Note that operators $0-5$ are the baseline operators for each robot action (rotational and translational speed), and operators $6-8$ were built through the operator-scaffolding technique.

	Operator	Parameter
0	Modify Speed, static (rot)	[cw ccw]
1	Modify Speed, static (tr)	[dec inc]
2	Modify Speed, fractional (rot)	[dec inc zero]
3	Modify Speed, fractional (tr)	[dec inc zero]
4	Modify Speed, incremental fractional (rot)	[dec inc]
5	Modify Speed, incremental fractional (tr)	[dec inc]
6	Adjust Both, fractional	[dec inc]
7	Adjust Turn, fractional	[loosen tighten]
8	Adjust Turn, incremental fractional	[loosen tighten]

TABLE 6.1. Advice-operators for the racetrack driving task.

[*Key*: (rot/tr)=$(rot/transl)ational$, $(c)cw$=$(counter)clockwise$, (dec/inc)=$(de/in)crease$]

At each execution time step, the robot computes a local track representation by fitting a 3-degree polynomial to track border points visually observed, in the current and recent-past timesteps. The coefficients of this polynomial are included as state observation features. Observations further include the current value of each action, in accordance with the F3MRP observation formulation of our principled approach (Sec. 4.2.3). The observations computed for this task thus are 6-dimensional: current rotational and translational speeds, and the 4 polynomial coefficients of the observed track. The actions are 2-dimensional: target rotational and translational speeds (ν, ω).

Lastly, we note that the aim of the developed policy is to reactively drive on a racetrack. The robot thus has no a priori map of the track, nor does it attempt to build up a map during execution.

6.2.1.2. Policy Development. Policy development begins with teleoperation demonstrations of the motion primitives by the teacher. In this domain the demonstrated motion primitives Ξ (Fig. 6.2) are: *turn right* (ξ_R), *go straight* (ξ_S), *turn left* (ξ_L). The steps taken by the FPS algorithm during policy development are to:

1. Demonstrate the motion primitives, and derive initial policies.
2. Provide feedback on the motion primitive policies' performance.
3. Derive an initial scaffolded policy from the resultant primitive policies.

4. Provide feedback on the scaffolded policy performance.

Demonstration of each primitive behavior is performed 3 times from a single location on the track appropriate for the associated primitive motion, for a total of 9 demonstrations (dataset sizes: $|D_{\xi_R}| = 297$, $|D_{\xi_S}| = 154$, and $|D_{\xi_L}| = 245$ points, summed datasets' size: 696 points). An initial policy is derived from each of these datasets; we refer to these collectively as the set of initial demonstrated primitive policies, PD_I.

Next the learner executes with each policy in PD_I, on a subset of the track corresponding to the respective location of the primitive behavior's demonstrations. The teacher observes these executions, offers feedback, and the executing policy is updated. Data generated from teacher feedback on executions with a given policy ξ_i is added to the respective primitive dataset D_{ξ_i} ($i = R, S, L$). This observation-*feedback*-update cycle constitutes a single *practice run*, and continues to the satisfaction of the teacher. Final feedback versions of the primitive policies are the result of this first phase of feedback. We refer to the collection of these policies as the final feedback primitive policies, PF_F (summed datasets' size: $1,835$ point).

The learner then builds a policy able to execute a more complex behavior; that is, able to execute on the full racetrack. A baseline scaffolded policy, that operates by selecting between the primitive policies in PF_F, is built; we refer to this as the initial feedback scaffolded policy, SF_I. The learner executes with this policy on the full track. Again these executions are observed by the teacher, and the observation-feedback-update cycle continues to her satisfaction. Here, data generated from feedback is added to either the primitive datasets *or* their respective *feedback* datasets, according to the paradigm described in Section 6.1.3.2. The final feedback scaffolded policy SF_F is the result from this second phase of feedback (summed datasets' size: $6,438$ points).

For comparative purposes, the approach of providing exclusively teacher demonstrations is also evaluated. This approach closely follows the advising approach just outlined, but the teacher provides more teleoperation demonstrations instead of feedback. In response to learner executions with the initial primitive policies in PD_I, the learner is provided with more *motion primitive* demonstrations. This observation-*demonstration*-update cycle constitutes a single *practice run*, and continues to the satisfaction of the teacher. These demonstrations are added to to the respective datasets for each primitive. Final demonstration versions of the primitive policies are the result. We refer to the collection of these policies as the final demonstration primitive policies, PD_F (summed datasets' size: $4,608$ points). A baseline scaffolded policy is then built from the these primitive policies; we refer to this policy as the initial scaffolded demonstration policy SD_I. In response to learner executions with policy SD_I on the full track, the teacher provides *full track* teleoperation demonstrations. These demonstrations are added to a new dataset, from which a policy is derived that also is considered during execution of the scaffolded policy. The result is the final scaffolded demonstration policy SD_F (summed datasets' size: $14,657$ points).

To summarize, a total of 13 policies are developed for empirical evaluation:

Primitive, Demonstration Initial (PD_I)	: Initial demonstration of each primitive (3 in total)
Primitive, Feedback Final (PF_F)	: Feedback for PD_I (3 in total)
Primitive, Demonstration Final (PD_F)	: More demonstration for PD_I (3 in total)
Scaffolded, Feedback Initial (SF_I)	: Initial scaffolding of the PF_F policies
Scaffolded, Feedback Final (SF_F)	: Feedback for SF_I
Scaffolded, Demonstration Initial (SD_I)	: Initial scaffolding of the PD_F policies
Scaffolded, Demonstration Final (SD_F)	: More demonstration for SD_I

6.2.1.3. Evaluation. Each of the developed policies are evaluated on racetrack executions. In particular, each of the primitive policies (contained in sets PD_I, PF_F, PD_F) is evaluated on the track subset appropriate to their respective primitive behavior (Fig. 6.2, left). Each of the scaffolded policies (SF_I, SF_F, SD_I, SD_F) is evaluated on the full track (Fig. 6.2, right). Executions proceed along the track (right to left) and end when the robot runs off the track or reaches the finish line.

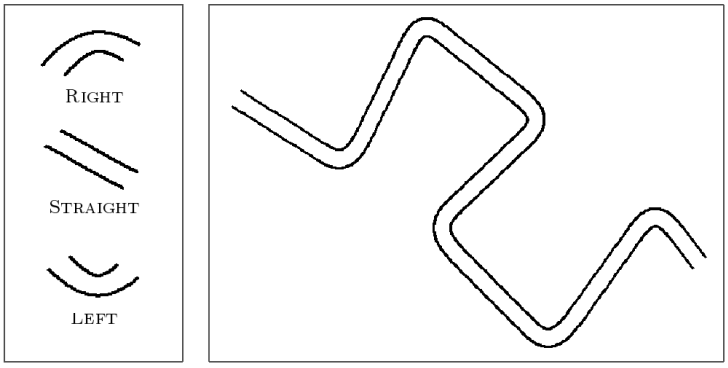

FIGURE 6.2. Primitive subset regions (left) of the full racetrack (right).

Policy performance is measured according to the following metrics. *Success* indicates the ability of the robot to stay on the track, and is measured by the percentage of the track subset (for primitive policy executions) or full track (for scaffolded policy executions) completed. *Speed* is measured by the average translational execution speed. *Efficiency* is measured as the execution time, and is governed jointly by speeds and the execution ground path. This measure is computed for successful executions exclusively; for unsuccessful executions, that by definition aborted early, time is not a useful measure.

6.2.2. Results

The FPS algorithm successfully learned motion control primitives through a combination of demonstration and teacher feedback. Furthermore, a policy built upon these primitives with feedback was able to successfully execute a more complex, *undemonstrated*, behavior. In each phase

of FPS algorithm development, teacher feedback was found to be critical to the development and performance improvement of policies. Additionally, the policies that resulted from FPS feedback far outperformed those that received only further teacher demonstrations.

6.2.2.1. Motion Primitives Learned from Demonstration. Three motion primitives were successfully learned from demonstration and human feedback. Full details of these results are presented in Table 6.2. For clarity of presentation, for the figures and tables of this section the label *Baseline* refers to the primitive policies in PD_I, *Feedback* to those in PF_F and *More Demonstration (More Demo)* those in PD_F. This constitutes Phase 1 of the FPS algorithm.

Policy	Success (%)	Speed, Transl [mean] ($\frac{m}{s}$)	Efficiency (s)
Baseline, Right	47.97 ± 1.45	0.61 ± 0.01	-
Feedback, Right	$\mathbf{97.61 \pm 12.0}$	$\mathbf{1.67 \pm 0.02}$	1.93 ± 0.07
MoreDemo, Right	51.79 ± 8.54	0.65 ± 0.01	-
Baseline, Straight	100.0 ± 0.0	0.60 ± 0.00	5.67 ± 0.13
Feedback, Straight	100.0 ± 0.0	$\mathbf{2.74 \pm 0.05}$	$\mathbf{1.26 \pm 0.03}$
MoreBaseline, Straight	100.0 ± 0.0	1.73 ± 0.34	3.11 ± 0.62
Demo, Left	$\mathbf{99.21 \pm 1.31}$	0.97 ± 0.01	2.76 ± 0.05
Feedback, Left	91.28 ± 19.30	$\mathbf{1.47 \pm 0.39}$	$\mathbf{1.80 \pm 0.41}$
MoreDemo, Left	43.76 ± 8.21	0.60 ± 0.02	-

TABLE 6.2. Performance of the Primitive Policies

That the motion primitives were successfully learned by FPS was evidenced by the ability of each primitive policy in PF_F to complete executions on the corresponding track subsets. Figure 6.3 shows the percent completed execution, for each primitive policy on its respective track subset (average of 50 executions, 1-standard deviation error bars).

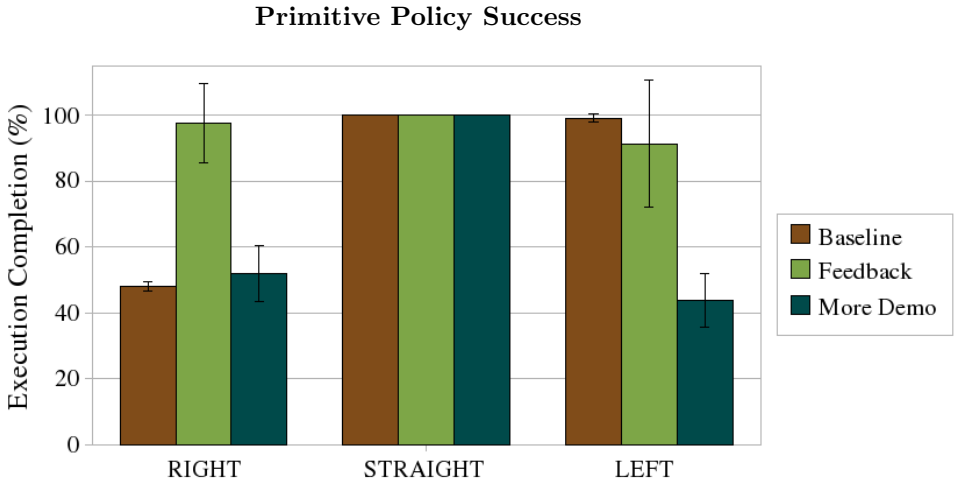

FIGURE 6.3. Driving task percent completion with each of the primitive behavior policies.

For the *turn right* primitive behavior, the initial policy in PD_I is unable to complete the task (Fig. 6.3, category *right*, brown bar). The *turn right* policy (in PF_F) resulting after Phase 1 development under the FPS algorithm, however, was able to complete the task (green bar). Furthermore, executions with this policy are much faster on average (Fig. 6.4). This policy developed over 23 practice runs, resulting in 561 new datapoints. By contrast, the *turn right* policy (in PD_F) resulting from more teleoperation demonstrations was not able to complete the task, or even to improve upon the performance or speed of the baseline policy (blue bar). Furthermore, this policy was developed with significantly more practice runs and training data: 36 practice runs, resulting in $2,846$ new datapoints.

For the *go straight* primitive behavior, the initial policy in PD_I *is* able to complete the task (Fig. 6.3, category *straight*, brown bar). The policy executes the task extremely slowly however (Fig. 6.4). By contrast, the *go straight* policy (in PF_F) resulting after Phase 1 development under the FPS algorithm is able to increase execution speeds (green bar), such that the *average* speed over the execution approaches the maximum speed of the robot ($3.0\frac{m}{s}$), all without compromising the success of the executions. This policy developed over 27 practice runs, resulting in 426 new datapoints. The *go straight* policy (in PD_F) resulting from more teleoperation demonstrations also improves execution speeds over the baseline policy, but not as dramatically (blue bar). This policy developed over 36 practice runs, resulting in $1,630$ new datapoints.

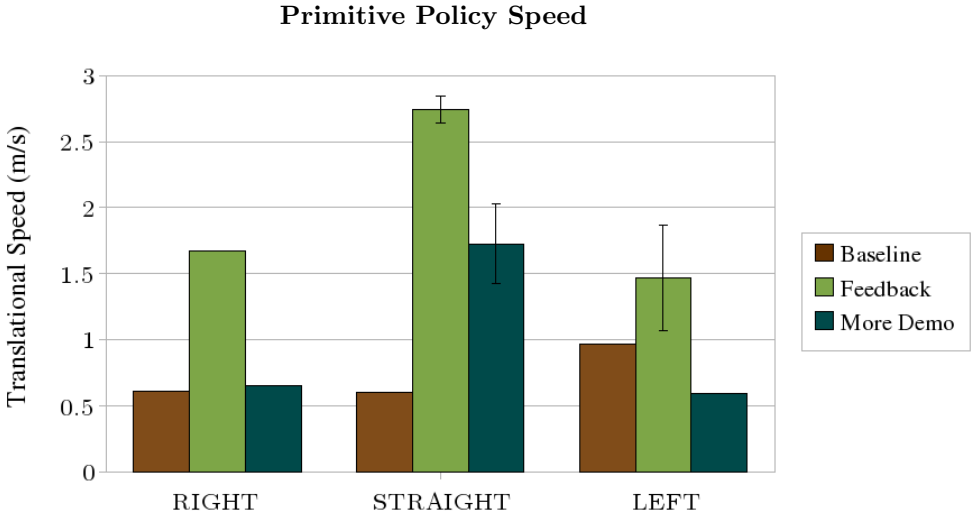

FIGURE 6.4. Driving task average translational execution speed with each of the primitive behavior policies.

For the *turn left* primitive behavior, the initial policy in PD_I again is able to complete the task (Fig. 6.3, category *left*, brown bar). The *turn left* policy (in PF_F) resulting after Phase 1 development under the FPS algorithm also is able complete the task (green bar), though occasionally does go off the track (Fig. 6.3). This is a consequence of the policy being more aggressive than the baseline policy, as evidenced by its increased execution speeds (Fig. 6.4, category *left*, green bar).

The advantage of this aggression is much more efficient executions (Tbl. 6.2, $Feedback$, $Left$), but at the cost of occasional incomplete executions. This policy developed over 8 practice runs, resulting in 252 new data points. The *turn left* policy (in PD_F) resulting from more teleoperation demonstrations actually decreased the performance of the initial policy, both in success and speed (blue bar). Presumably more demonstrations in this case created ambiguous areas for the policy, a complication that would perhaps clear up with the presentation of more disambiguating demonstrations. This policy developed over 12 practice runs, resulting in 965 new datapoints.

6.2.2.2. Undemonstrated Task Learned from Primitives and Feedback. A policy able to execute a more complex, *undemonstrated*, behavior was successfully developed through the scaffolding of the learned primitive policies and the incorporation of teacher feedback. The complex task here is the robot driving on the full racetrack. Before any practice runs with teacher feedback, the complex policy, derived solely from selection between the developed feedback primitive policies PF_F, is unable to execute this task in full. Performance improvement over 160 practice runs is presented in Figure 6.5. Each practice run produces a new iterative policy. Each plot point represents an average of 10 track executions with a given iterative policy, and a regularly sampled subset of the iterative policies were evaluated in this manner (sampled every 10 policies, 17 iterative policies evaluated in total). This constitutes Phase 2 of the FPS algorithm, after which the learner is able to consistently execute the complex task in full.

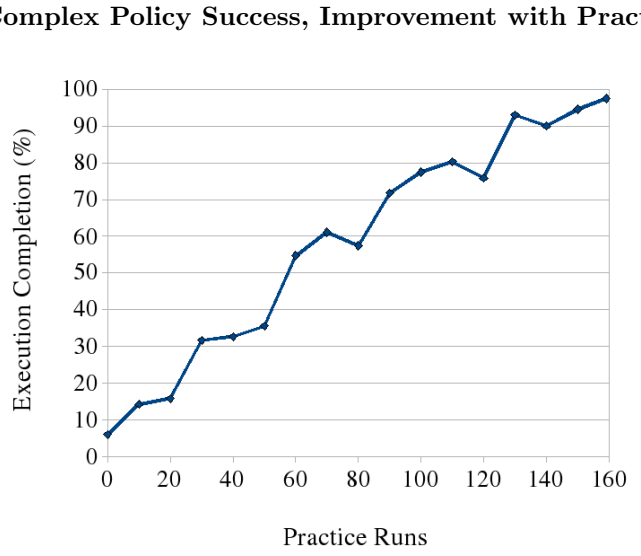

FIGURE 6.5. Driving task percent completion during complex policy practice.

6.2.2.3. Improvement in Complex Task Performance. Beyond the development of a policy *able* to perform the more complex task, Phase 2 of the FPS algorithm further enabled performance *improvement* such that task executions became faster and more efficient. Figure 6.6

shows the percent completed execution of multiple policies on the full track task (average of 50 executions, 1-standard deviation error bars); further details of these results are presented in Table 6.3.

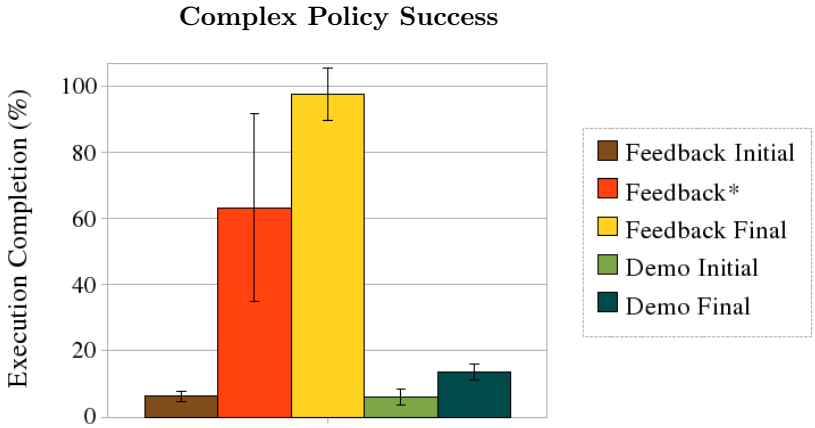

FIGURE 6.6. Driving task percent completion with full track behavior policies.

As noted above, the initial FPS policy SF_I, derived exclusively from the primitive feedback policies in PF_F, was not able to complete this task (Fig. 6.6, *Feedback Initial*). Neither was the initial policy SD_I derived exclusively from the primitives in PD_F developed from more teacher demonstrations (*Demo Initial*). Both of these policies perform similarly poorly in the measure of execution success.

The final policy SF_F that results after Phase 2 of the FPS algorithm, however, is able to consistently execute the task successfully (Fig. 6.6, *Feedback Final*). This policy developed over 160 practice runs, resulting in $4,503$ new datapoints. By contrast, the policy SD_F that resulted from more teleoperation demonstrations (*Demo Final*) was not able to complete the task, though it did improve upon the performance the initial demonstration policy (*Demo Initial*). The policy SD_F developed over 75 practice runs, resulting in $8,520$ new datapoints.

Policy development terminated at the discretion of the feedback teacher. In general the teacher decided to terminate development once a policy displayed either satisfactory performance or no further performance improvement. In the development of final demonstration policy SD_F, however, the teacher aborted policy development due to her frustration with the extremely slow rate of policy improvement. Since the final FPS policy SF_F received feedback on more practice runs than the more demonstration policy SD_F (159 *vs.* 74), the above comparison between these policies is not quite fair. To provide a fair comparison, results from an *iterative* FPS policy are also presented (Fig. 6.6, *Feedback**). This policy is not the final FPS policy, but rather the result of development after only 74 practice runs, the same number of practice runs as the final teleoperation policy. These runs produced $2,448$ new datapoints; far fewer than the 74 runs of the demonstration policy, which produced $8,520$ new points. Against this iterative policy, however, the final demonstration policy also does not measure well. The iterative policy (*Feedback**) significantly outperforms the

final demonstration policy (*Demo Final*) on the success measure, though it does not yet perform as successfully or as consistently as the final FPS policy (*Feedback Final*), .

Policy	Success (%)	Speed, Transl [mean, max] ($\frac{m}{s}$)	Speed, Rot [max] ($\frac{rad}{s}$)
Feedback Initial	6.32 ± 1.72	0.42 ± 0.21 , 1.51 ± 0.36	0.88 ± 0.3
*Feedback**	63.32 ± 28.49	2.24 ± 0.18 , 3.04 ± 0.17	2.14 ± 0.2
Feedback Final	$\mathbf{97.51 \pm 7.94}$	$\mathbf{2.34 \pm 0.03}$, $\mathbf{3.07 \pm 0.01}$	$\mathbf{2.57 \pm 0.06}$
Demo Initial	5.95 ± 0.17	0.58 ± 0.00 , 0.66 ± 0.13	0.15 ± 0.10
Demo Final	13.69 ± 2.36	1.01 ± 0.13 , 1.51 ± 0.56	0.98 ± 0.14

TABLE 6.3. Performance of the Scaffolded Policies

Policies are additionally evaluated for speed, the results of which are presented in Figure 6.7. Since the full track is much longer than any of the primitive subsets, and thus offers more opportunity to reach higher speeds, the maximum speeds attained during executions is shown in addition to the average speed.

The speed performance results closely resemble those of success performance. Namely, the final FPS policy (*Feedback Final*) far outperformed both the initial FPS policy (*Feedback Initial*) as well as the final demonstration policy (*Demo Final*). The final demonstration policy did offer some improvement over the initial demonstration policy (*Demo Initial*), but not nearly as much as the iterative FPS policy provided for comparison (*Feedback**). Interesting to note is that this iterative FPS policy produced speeds similar to the final FPS in nearly all measures, though with slightly larger standard deviations (Tbl. 6.3). This suggests that the motivation for continuing practice and development beyond the iterative policy was related to performance *consistency*, in addition to execution success.

6.2.2.4. Appropriate Behavior Selection. The demonstration teacher provides no explicit indication of circumstances in which the behavior of a given primitive is suitable for the complex task. Instead, the FPS algorithm selects between behavior primitives based solely on the distribution of their data within the observation space. In this section we show that this selection paradigm did in fact choose behavior that was appropriate for the various portions of the complex task execution.

Figure 6.8 shows examples of primitive policy selection, during full track executions with the final FPS policy SF_F. Five example executions are shown as panels A-E. Across a panel, a single execution trace is displayed in triplicate, with each plot corresponding to the selection of a different policy primitive. Within a single plot, execution points for which the given behavior policy was selected are displayed as blue circles, and those for which the primitive behavior's associated feedback policy was selected are displayed as red circles. For reference, the full execution trace is displayed as well (cyan line). Each execution begins at the track start line (green line, lower right corner) and ends at either the finish line (magenta line, upper left corner) or the point at which the robot drove off the track (e.g. panel C).

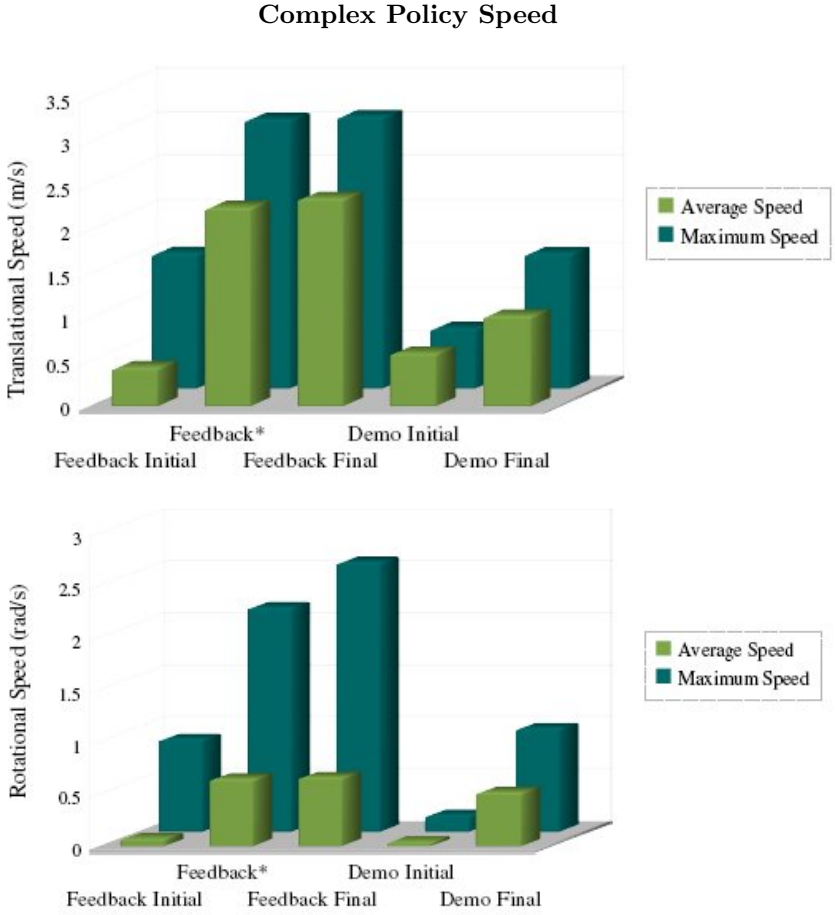

FIGURE 6.7. Driving task average and maximum translational (top) and rotational (bottom) speeds with full track behavior policies.

A macro-level analysis across each panel confirms the correct selection of primitive policies overall; namely that in right-turn areas of the track the *right* behavior was selected, in straight areas the *straight* behavior was selected and in left-turn areas the *left* behavior was selected.

Upon closer inspection, interesting differences between policy selections are discernible. For example, in the right-turn track areas, across all panels the *right-feedback* policy is seldom selected (red circles, first column); by contrast the *straight-feedback* and *left-feedback* policies are frequently selected in their respective track areas (red circles, second and third columns). There are three possible algorithmic explanations for this low selection frequency. One: Predictions made by the *right* policy never received teacher feedback. Two: The *right* policy predictions did receive teacher feedback, but were rarely predicting in areas of the state-space outside the support threshold τ for the *right* policy. Both of these reasons would result in the *right-feedback* dataset receiving few datapoints and consequently having very limited data support. Three: The behavior of the complex task policy developed in such a way that it no longer visits the state-space areas that populate the *right-feedback* dataset. In this particular experimental case, the second reason accounts for the low selection frequency.

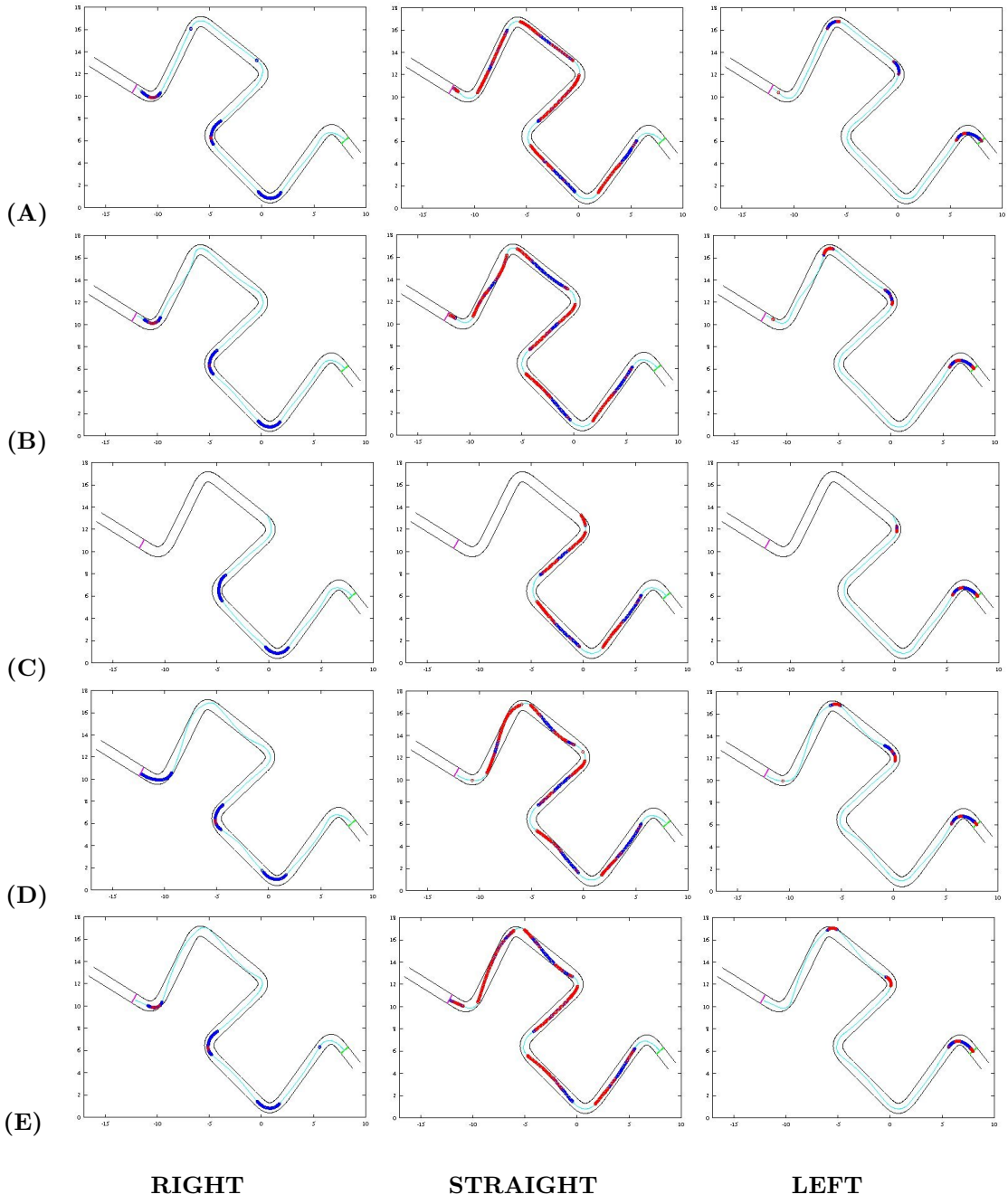

FIGURE 6.8. Example complex driving task executions (rows), showing primitive behavior selection (columns); see text for details.

Other selection differences may be observed between panel executions. For example, between panels A and B, note the differences between selection of the *straight* primitive policy at the topmost track segment. The execution in panel A selects the *straight-feedback* policy almost exclusively (red circles), while the execution in panel B by comparison prefers the *straight* policy (blue circles). Panel A's execution runs close to the right-hand border, while B's runs close to the left-hand border,

indicating that these are state-space areas supported respectively by the *straight* and *straight-feedback* datasets. Another example is panel C, which shows an execution that runs off the track before completion. Specifically, it runs off the track after a left turn. Comparisons between panel C's selection of behavior *left* at that turn and the selection in all other panels shows that the duration of the selection of the *left* policy (third column) is shorter in panel C, suggesting that the robot transitioned into its selection of the *straight* behavior too early, before completing the left turn. Not all selection transitions are this delicate however; for example compare the exit strategies of panels D and E on the final track turn. Panel D continues to select the *right* policy until reaching the finish line, while panel E transitions to select the *straight-feedback* policy, yet *both* exit the turn and complete the track successfully.

We draw two general conclusions from this qualitative analysis. The first is that our primitive selection paradigm is performing sensibly. The second is that the effects of primitive policy selection are complex, and a policy may be alternately sensitive or insensitive to primitive selection at various points during task execution. We therefore highlight the investigation of alternate approaches to primitive behavior selection as a rich area for future research (Sec. 6.3.2.1).

6.3. Discussion

This section presents a discussion of the FPS algorithm and its above empirical implementation. Directions for future research are then identified, as extensions to discussions that overview the development of two key design considerations of this algorithm: primitive behavior selection, and feedback incorporation into multiple policies.

6.3.1. Conclusions

Here conclusions relating to FPS algorithm development and experimental results are detailed. First highlighted are some noteworthy gains provided by this algorithm, followed by a discussion of the role dataset support plays in the algorithm and the datasets that result from teacher feedback.

6.3.1.1. Reuse of Primitives Learned from Demonstration. The empirical results confirm that FPS was able to successfully build a policy for an undemonstrated task, from existing primitive policies learned from demonstration. There are two crucial gains to such an approach.

The first gain is that the multiple motion primitive policies were developed from *demonstration*. Demonstration has many attracive features as a medium for knowledge transfer from human teacher to robot learner, as already noted throughout this document. Moreover, this demonstration technique was aided with teacher *feedback*, provided under the F3MRP framework. Without this feedback, the learned primitive policies are less successful, less efficient, slower, and in some cases even unable to complete the target behavior. This is true not just of the initial primitive policies derived from demonstration, but also of the policies provided with more demonstrations in response to learner execution performance. The FPS algorithm therefore provides a more efficient and effective LfD technique for the development of these motion primitive policies.

Even with the advantages secured through demonstration and teacher feedback however, policy development typically is still a non-trivial task. The second gain of the FPS approach therefore

is the ability to *reuse* the primitives within another task. The full track task was shown to be sufficiently complex that the improvements afforded by demonstrations of the full task behavior were significantly smaller than those gained through teacher feedback. Moreover, this performance difference between the feedback and more-demonstration techniques was much larger for the complex task than for the simpler primitive policies (compare Figs. 6.3 and 6.6). These figures suggest that the complex task would not be learned through demonstration exclusively, i.e. from full task demonstrations alone, unless provided with a very large quantity of demonstration data, again underlining the advantage of simple policy reuse within this complex domain.

6.3.1.2. Feedback Incorporation based on Data Support. One core idea in the FPS feedback framework is the measure of data support associated with policy predictions. This measure is employed in two distinct steps of the feedback framework operation.

Data support is first considered by the *teacher* when selecting feedback for a given learner execution. The graphical display of the F3MRP interface provides the teacher with an idea of data support throughout the execution. Segments of the execution that correspond jointly to good performance and weak data support are selected to receive positive credit feedback from the teacher. The data support measure in this case drives both *where* and *what type* of feedback is provided. Since this feedback produces new data, the support measure consequently also drives the generation of new demonstration examples.

Prediction data support is next used by the *learner* when selecting which dataset receives new feedback-generated data. One of the research questions considered in this thesis is how to incorporate teacher feedback into complex tasks. The idea of data-supported predictions is fundamental to the feedback incorporation approach taken in the FPS algorithm. The intent here is to maintain the performance integrity of the learned primitive behaviors. The data support measure in this case guides policy development, both of the primitive behavior policies, as well as their associated primitive-feedback policies, whose datasets receive those points deemed unsuitable for, i.e. distant from, the primitive behavior datasets.

6.3.1.3. Even More Focused Datasets. One result of the A-OPI experimental validations was the development of much smaller datasets with corrective advice in comparison to more demonstrations (Ch. 5, Sec. 5.3.2.2). These smaller datasets furthermore produced similar or superior performance, prompting the conclusion that the data contained in the A-OPI datasets was less redundant and more focused.

This trend is also seen with the FPS datasets of this work, and appears to only magnify with the more complex task of this domain. In particular, the combined size of the three primitive policy datasets developed with more demonstration ($6,137$) is more than three times the size of the comparable FPS primitive datasets ($1,935$) and in fact is on par with the size of the *final* FPS scaffolded policy ($6,438$). The size of the final scaffolded more-demonstration policy dataset ($14,657$) is more than double the final FPS dataset size, and this is with far fewer practice runs (74 *vs.* 168) The size of the comparable iterative FPS dataset ($4,383$) is less than a third of the final demonstration dataset.

In contrast to the Chapter 5 results however, these teleoperation policies never perform similarly to their FPS counterparts, and instead usually display significantly *inferior* performance. This observation suggests that not only is the added data less redundant, but furthermore includes *relevant* data that is *not* being produced by the demonstration. In particular, revisiting states to demonstrate a correct behavior is difficult with motion control tasks, and this difficulty scales with task complexity. The difficulty in revisiting states was noted when the teacher attempted to provide corrective demonstrations for primitive executions on the track subsets, and only increased with the full track executions when revisiting execution states even approximately became nearly impossible. Further detrimental to the developed policies was the fact that the teacher often was required to reproduce suboptimal behavior in order to *reach* the state intended to receive a corrective demonstration. The result was that poor demonstration data was provided to the learner in addition to the behavior correction in the target area. While there do exist a handful of algorithms that explicitly deal with bad demonstrations (see Sec. 9.1.3), this is not typical of LfD in general and furthermore was not the focus of this work.

In contrast to all of the difficulties associated with revisiting state, policy improvement under the F3MRP framework does *not* require revisiting any states in order to provide feedback. We posit that the value of this key advantage of the F3MRP framework only grows as tasks and domains become more complex.

6.3.2. Future Directions

This section provides ideas for the extension and improvement of the FPS algorithm. The presentation is framed within a discussion of two key design decisions for this algorithm: how to select between primitive behaviors, and how to incorporate teacher feedback into the complex policy.

6.3.2.1. Primitive Behavior Selection. The initial version of the algorithm performed no explicit primitive selection. Instead, this implementation relied exclusively on the local nature of the employed regression techniques, and the assumption that primitives occupy distinct areas of the state-space. Demonstrations of all primitive behaviors were pooled into a single dataset. Locally Weighted Learning regression techniques consider only nearby data in the regression prediction. Since all nearby data was assumed to result from demonstrations of the same primitive, when operating in the state-space area of a particular primitive the regression technique in theory incorporates data from a single primitive only, and thus implicitly performs primitive selection.

The performance of this implicit primitive selection, however, was found to be substandard. Though primitives do occupy nominally distinct areas of the state-space, the implicit primitive selection paradigm relied on the stronger assumption of no overlap between these areas. Violations of this assumption resulted in the mixing of data from different primitives into a single regression prediction, and consequently poor performance in regions of primitive data overlap. Poor performance will result if the actions of different behavior primitives conflict, for example if the rotational speed of $1.0\frac{rad}{s}$ for primitive *turn left* versus $1.0\frac{rad}{s}$ for *turn right* become a mixed action prediction that does not turn at all, $0.0\frac{rad}{s} = \frac{1}{2}(1.0\frac{rad}{s}) + \frac{1}{2}(-1.0\frac{rad}{s})$.

The next algorithm version therefore relaxed the assumption of non-overlapping state-spaces between primitives. Datapoints were restricted to make predictions only with other points of the same primitive type; effectively isolating demonstrations of a particular primitive type into separate datasets, and treating each primitive as its own policy. This implementation then performed explicit primitive selection at execution time based on query point nearness, by selecting the primitive with the dataset containing the nearest point to the query point.

A more robust version of this implementation is included in the final version of the algorithm. Primitives similarly are treated as individual policies, and primitive selection occurs as a function of query point distance to the primitive datasets. Selection does not consider only the single nearest dataset point however, but rather a kernelized weighting over the k-nearest points within each primitive dataset.

A variety of alternatives are possible for the choice of how to select between primitive behaviors. Of particular interest would be to explore approaches that do not assume primitives to reside in distinct areas of the state-space, and thus do not require an a observation-space formulation that supports this assumption. As noted in Section 6.2.2.4, the effects of primitive selection on policy performance can be complex, and the investigation of alternate paradigms is highlighted as a promising area for future research.

6.3.2.2. Teacher Feedback Incorporation into Multiple Policies. Our discussions of teacher feedback in the previous chapters considered feedback that updates a single policy. For feedback types that produce new data, the policy update happens simply by incorporating this data into the demonstration set. The work of this chapter, however, considers a scaffolded policy built from *multiple* primitive policies and their respective datasets. Behavior for the scaffolded policy was never demonstrated and, therefore, no corresponding *scaffolded-behavior* dataset exists. The question then is how to incorporate new data produced from executions with this scaffolded policy.

The first feedback incorporation approach considered during development of the FPS algorithm added a new datapoint to the dataset of the primitive policy that predicted the execution point receiving feedback. For query points close to the dataset providing the prediction, and whose target behavior thus was close to the behavior of the dataset primitive, this approach was reasonable and on par with advising a single policy. The more complex policy, however, may enter areas of the state-space that were unvisited during the development of the primitive policies. A query produced in these areas is therefore distant from all of the primitive datasets; even the nearest dataset, that provided the prediction for the query. The behavior of the predicting primitive may even conflict with the intended behavior of the more complex policy in that state-space area, especially if aspects of the target behavior were not captured by any of the policy primitives. Corrections that take a step in the direction of this final policy behavior step *away* from the target primitive behavior. For query points distant from the dataset of the policy providing the prediction, whose target behavior is *not* close to or perhaps even conflicts with the behavior of that policy, adding such points to the primitive dataset therefore compromises the intended behavior of the primitive. The result is degraded performance on the learned primitive policy behavior, due to over-generalization.

The second feedback incorporation approach that was considered placed all new data into a single separate feedback dataset. The policy derived from this dataset was then used for selection during execution with the more complex policy, just like the primitive policies. While this approach improved policy performance, it too began to over-generalize. Here however the *feedback* policy over-generalized, rather than the primitive policies. The dataset of this policy contained the weakly data-supported predictions of *all* primitive policies, which vary markedly in their respective intended behaviors. That a policy derived from this set proved to be overly general is therefore not too surprising. To mitigate over-generalization, the final algorithm associated a feedback dataset with *each* policy primitive. The policies produced from these datasets proved to be sufficiently focused, and satisfactory policy performances resulted.

However, the decision to associate one feedback dataset with each primitive policy was somewhat arbitrary, and made based on empirical support of good policy performance (Fig. 6.8). One could envision many other equally valid approaches to determining the number of feedback datasets, such as techniques that automatically cluster new data into datasets. Such approaches could even be extended to the development of the primitive policies. In the current approach, the demonstration teacher effectively provides a label for each execution, indicating into which dataset the demonstration should be placed. An approach like clustering could allow these labels to instead be automatically determined by the algorithm.

A final consideration is the value of the data support threshold τ. This threshold determines whether to add new data to an existing primitive dataset or to its associated feedback dataset. If this threshold is overly strict, then data that actually could be incorporated into the primitive policy, because the behavior it represents is sufficiently similar to the primitive behavior, is unnecessarily excluded from that primitive dataset. If this threshold is overly lax, then data may be included that compromises the primitive policy, if the behavior it represents is sufficiently dissimilar from or conflicts with the primitive behavior. In this work, the value of the data support threshold was set by hand. How the threshold might be tuned automatically from the data is not obvious, but if introduced such a mechanism would improve the robustness and statistical foundations of this algorithm.

6.3.2.3. Real Robot Application. In the existing behavior architecture of our Segway RMP robot, we accomplish autonomous control through the use of hierarchical state machines (Simmons and Apfelbaum, 1998). A state machine for control consists of a set of control states, each of which encodes a control policy. Transitions between different control states occur as a function of observed state, and a state machine may terminate (i.e. enter an absorbing state) with success or failure. A state machine may therefore be viewed as providing deliberative goal-driven behavior, that will either fail or succeed and achieve the goal.

We arrange these state machines into *hierarchies*, where a state machine may transition to, or call, other state machines. Two control policy types are available: complex or primitive. A complex policy calls another control policy, thus passing control to another state machine. This builds a hierarchy of state machines. By contrast, primitive policies call primitive control actions

that command the robot (e.g. velocity control). Note, that state machines may also execute in parallel, with or without synchronization coupling their execution.

We employ this hierarchical state machine architecture for autonomous control of Segway RMP robots. Our prior work with this robot platform is extensive, and consists for the most part of heterogeneous, multi-robot and human-robot, domains. In particular, we have used for this hierarchy for control within Segway Soccer, a league within Robocup robot soccer, in which teams of humans and robots organize and cooperate together in soccer competition against other human-robot teams (Argall et al., 2006). We have also used this architecture for control within the Boeing Treasure Hunt project, in which heterogeneous teams of humans and a variety of robots coordinate together in pursuit of a common task goal (Jones et al., 2006; Dias et al., 2008).

The power in a skill hierarchy lies in its task decomposition; that is, by enabling a larger task to be easily decomposed into sub-tasks that can be solved as independent problems. Moreover, each of the resulting skills may be reused for other similar problems. The drawback to this approach is that for a real robot, the control architecture is often hand coded. Typically, it is not task decomposition which is difficult. Rather, most of designer effort focuses on developing the control policies which call primitive actions, and tuning the transitions between policies. Moreover, the performance of skills using such policies is highly dependent on robot hardware and the environment. In our experience, building such policies has often been tedious and time-consuming.

The FPS policy building algorithm offers an alternative to the drawback of hand-coding the control policies that call primitive actions. Here primitive control behaviors are *learned* from demonstration, and refined through teacher feedback. Furthermore, the *transitions* between primitive behaviors are also learned. These transitions are a function of the world observation formulation - relaxing this requirement has already been discussed as an area for future research - and teacher feedback.

The empirical implementation of this chapter took care to keep the simulation realistic to our Segway RMP robot platform, from setting system parameters to camera-based sensing. The natural next step for FPS is to validate the algorithm on our robot platform. We expect good results for the building of behavior primitives, given our past success in using corrective feedback to refine behavior on the Segway RMP (Ch. 5). The simulation results presented here support the expectation that policy scaffolding on a real robot will also produce good results. The results additionally suggest that building complex behaviors with our scaffolding algorithm will be significantly more efficient and successful than providing teacher demonstrations alone, an advantage that becomes all the more relevant when demonstrations involve executing physical actions on a mobile robot within the real world.

6.4. Summary

The Feedback for Policy Scaffolding (FPS) algorithm has been introduced here as an approach that uses teacher feedback to build complex policy behaviors. More specifically, the algorithm operates in two phases. The first phase develops primitive motion behaviors from demonstration and feedback provided through the F3MRP framework. The second phase scaffolds these primitive

behaviors into a policy able to perform a more complex task. F3MRP feedback is again employed, in this case to assist scaffolding and thus enable the development of a sophisticated motion behavior.

The FPS algorithm was implemented within a simulated motion domain. In this domain, a differential drive robot was tasked with driving along a racetrack. Primitive behavior policies, which represent simple motion components of this task, were successfully learned through demonstration and teacher feedback. A policy able accomplish a more complex behavior, i.e. to drive the full track, was successfully developed from the primitive behaviors and teacher feedback. In both development phases, empirical results showed policy performance to improve with teacher feedback. Furthermore, all FPS policies outperformed the comparative policies developed from teacher demonstrations alone. While these exclusively demonstrated policies were able to nominally perform the primitive behavior tasks, albeit with varying degress of success and always less adeptly than the FPS policies, they were never able to perform the more complex task behavior.

This work again underlines the benefit, afforded through the F3MRP framework, of not needing to revisit state to provide feedback. State formulations in this work depended on the robot's observation of the track borders in addition to execution speeds, resulting in a 6-dimensional continuous-valued state space; to accurately revisit a state under such a formulation is effectively impossible. The execution segment selection technique of F3MRP is more accurate at focusing feedback to the required states than is teacher demonstration, which must revisit the states to provide corrective feedback. The result here was smaller resultant datasets paired with superior policy performance. This chapter also highlighted the many potential gains to using primitives learned from demonstration to build more complex policies; in particular, the potential for primitive policy reuse.

One indicated area for future research was alternate approaches to the incorporation of feedback into a complex behavior policy. The implementation of other techniques for the selection between, and scaffolding of, primitive behavior policies was also identified as an area for future research, in particular within domains where primitive behaviors do not occupy distinct areas of the state-space.

CHAPTER 7

Weighting Data and Feedback Sources

One attractive feature of LfD is that demonstration does not require robotics expertise, which opens policy development to non-robotics-experts. We predict that the need for programming techniques that are accessible to non-experts will increase as robots become more prevalent within general society. This chapter considers demonstration data from multiple sources, for example multiple demonstration teachers. We argue that this consideration will be particularly relevant for general society LfD robotics applications, where multiple users of a robot system is likely. Multiple users easily translates into multiple policy developers, and therefore into multiple LfD teachers.

Algorithm *Demonstration Weight Learning (DWL)* explicitly considers LfD policy development that incorporates demonstration data from multiple sources (Argall et al., 2009a). The algorithm further considers the possibility that all LfD data sources are not equally reliable; for example, multiple teachers may differ in their respective abilities to perform a given task. Algorithm DWL considers reliability, by assigning a weight to each data source. The weight is automatically determined and updated by the algorithm, based on learner performance using the LfD policy.

Another way to view DWL is as an alternate approach to incorporate teacher feedback into a complex behavior. In Chapter 6, feedback was used to refine primitive policies, and to facilitate their contribution to a complex behavior. By contrast, here feedback is treated as a distinct *data source*. More specifically, different types of feedback, like corrective advice and binary performance flags, are treated as *separate* sources. Since DWL addresses the issue of data source reliability and stochasticity, by assigning a weight to each data source and dynamically updating that weight as the policy develops, the algorithm thus also evaluates different *feedback types* for reliability.

This chapter contributes DWL as an algorithm that incorporates data from multiple demonstration sources, and reasons explicitly about their respective reliabilities. A first implementation of DWL in a simulated robot domain is presented. Results find data sources to indeed be unequally reliable in the domain, and data source weighting to impact task execution success. Furthermore, DWL is shown to produce appropriate source weights that improve policy performance.

The following section introduces the DWL algorithm, with the details of algorithm execution and weighting between multiple demonstration data sources. Section 7.2 presents the implementation and results of our empirical validation. In Section 7.3, the conclusions of this work are provided, along with direction for future research.

7.1. Algorithm: Demonstration Weight Learning

This section presents the Demonstration Weight Learning algorithm. The algorithm assigns a weight to each data source, and dynamically updates the weight during policy development. Figure 7.1 presents a schematic of our data source weighting algorithm, where dashed lines indicate repetitive flow and therefore execution cycles that are performed multiple times. In comparison to the schematic of the baseline feedback algorithm (Fig. 2.4), note in particular that at execution time the learner selects between multiple policies, and that a single reward received for an execution is incorporated into an update of the data source weights.

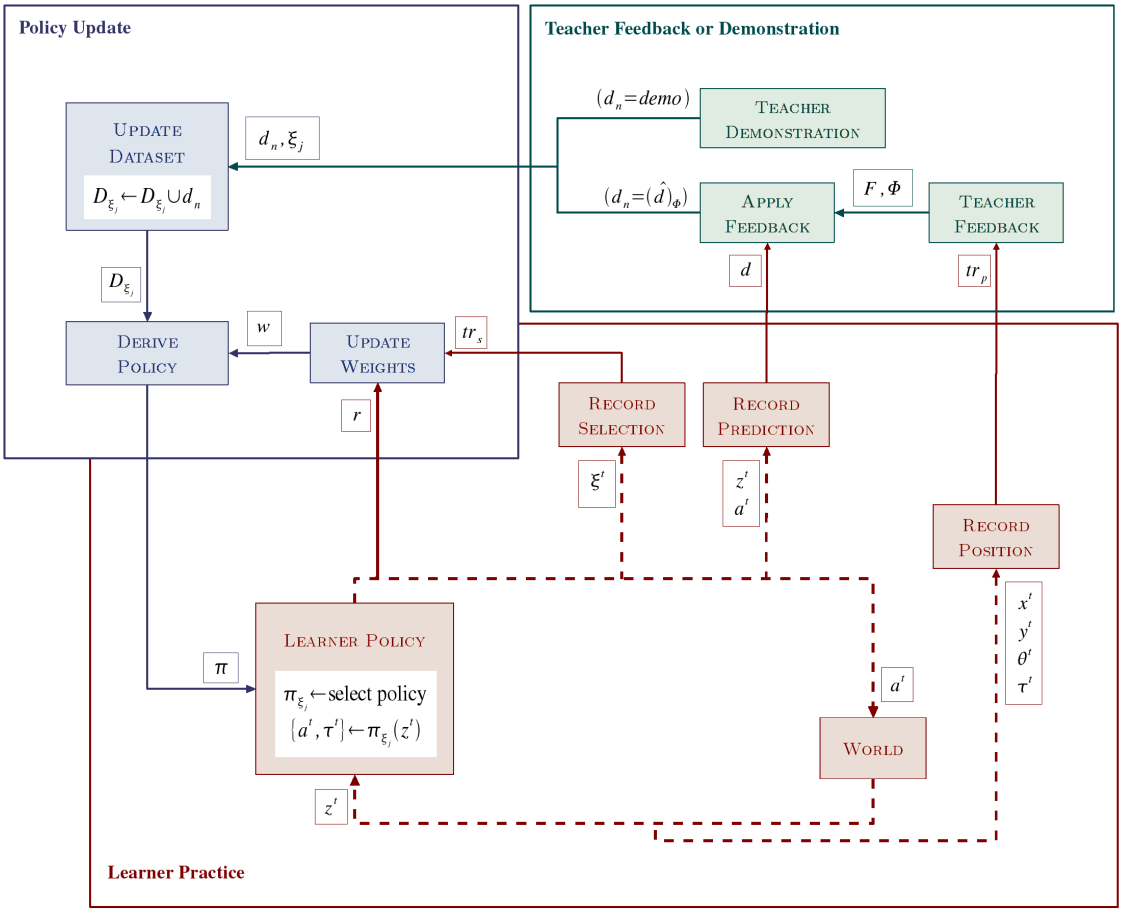

FIGURE 7.1. Policy derivation and execution under the Demonstration Weight Learning algorithm.

7.1.1. Algorithm Execution

Given a set of demonstration sources Ξ, DWL automatically determines a set of weights $\mathbf{w}$ over the sources. These weights are considered when deriving a policy π from the demonstration data $\mathbf{D}$, and are dynamically updated in response to learner execution performance. Psuedo-code for DWL is presented in Algorithm 6.

Algorithm 6 *Demonstration Weight Learning*

1: Given $\mathbf{D}$
2: *initialize* $\mathbf{w} \leftarrow 1$
3: *initialize* $\Pi \leftarrow$ `policyDerivation(` $\mathbf{D}, \mathbf{w}$)
4: **while** *practicing* **do**
5: *initialize* $d \leftarrow \{\},\ tr_s \leftarrow \{\},\ tr_p \leftarrow \{\}$
6: **repeat**
7: *select* $\pi_{\xi_j} \leftarrow$ `policySelection(` $\mathbf{z}^t, \mathbf{w}$) , $\pi_{\xi_j} \in \Pi$
8: *predict* $\{\ \mathbf{a}^t, \tau^t\ \} \leftarrow \pi_{\xi_j}$ ($\mathbf{z}^t$)
9: *execute* $\mathbf{a}^t$
10: *record* $d \leftarrow d \cup (\mathbf{z}^t, \mathbf{a}^t),\ tr_s \leftarrow tr_s \cup (\xi^t = \xi_j),\ tr_p \leftarrow tr_p \cup (x^t, y^t, \theta^t, \tau^t)$
11: **until** done
12: *receive* r
13: *adapt* $\mathbf{w} \leftarrow$ `weightUpdate(` r, tr_s)
14: *advise* $\{\ F, \Phi\ \} \leftarrow$ `teacherFeedback(` tr_p)
15: *apply* $\hat{d}_\Phi \leftarrow$ `applyFeedback(` F, Φ, d)
16: *select* $D_{\xi_\varphi} \leftarrow$ `datasetSelection(` F) , $D_{\xi_\varphi} \subseteq \mathbf{D}$
17: *update* $D_{\xi_\varphi} \leftarrow D_{\xi_\varphi} \cup \hat{d}_\Phi$
18: *rederive* $\Pi \leftarrow$ `policyDerivation(` $\mathbf{D}, \mathbf{w}$)
19: **end while**
20: **return** Π

Like the baseline feedback algorithm, the first phase of DWL consists of teacher demonstration. Here, however, demonstrations from multiple sources produce dataset $\mathbf{D}$, where source ξ_i produces data subset $D_{\xi_i} \subseteq \mathbf{D}$. The details of data recording during teacher demonstration are identical to those of the baseline feedback algorithm (Alg. 1, lines 1-8), and for brevity those details are omitted here. From each dataset $D_{\xi_i} \in \mathbf{D}$ an initial policy π_{ξ_i} is derived, producing a set of policies Π (line 3). All data source weights are initialized to 1.

The second phase of DWL consists of learner practice. At each decision cycle, the first step is to select between all contributing policies π_{ξ_j} based on the current observation $\mathbf{z}^t$ and data source weights $\mathbf{w}$ (line 7); the details of this selection are provided in Section 7.1.2. The second step is to predict action $\mathbf{a}^t$ according to $\pi_{\xi_j}(\mathbf{z}^t)$, with prediction support τ^t (line 8). The action is executed. A variety of information is then recorded (line 10). The predicted action is recorded in the prediction trace d, along with observation $\mathbf{z}^t$. The selected policy is recorded within the selection trace tr_s; this information will be used during the weight update. The learner position and heading within the world (x^t, y^t, θ^t) is recorded along with the prediction support τ^t (Sec. 2.3.2.1) into the position trace tr_p, for use by the F3MRP framework for the visual display of the robot execution path.

Following a learner execution, the source weights, datasets and policy are all updated. First, the data source weights $\mathbf{w}$ are adapted based on execution reward r and the recorded policy selection trace tr_s (line 13); the details of this adaptation are provided in Section 7.1.2.2. Second, the teacher feedback is applied across all points recorded in d and within the indicated subset Φ, producing data $\hat{d}_\Phi$ (line 15). New data may or may not be provided in response to the learner execution performance, at the discretion of each data source ξ_i. Note that if new data is being provided by a *demonstration* teacher, instead of as a result of teacher feedback, the new data $\hat{d}_\Phi$ is provided

directly by the teacher, and the feedback application step of line 15 is not necessary. Third, a dataset D_φ is selected to receive the new data (line 16). This selection is determined by the type of feedback f encoded the contributing source ξ_φ; Section 7.2.1.2 describes the various data sources of this work in full. Fourth, new policies Π for each source are derived from the updated set $\mathbf{D}$ and adapted weights $\mathbf{w}$ (line 18).

7.1.2. Weighting Multiple Data Sources

This section presents the details of weighting multiple data sources. Policy prediction is first addressed, during which a single data source is selected. Next, an overview of *expert learning* is provided. Data source weighting is accomplished via an expert learning approach, with each data source being treated as an expert. We adopt a modified version of Exp3 (Auer et al., 1995), an expert learning algorithm with successful prior robot applications (Argall et al., 2007b; Bowling and Veloso, 2001). Unlike typical expert learning approaches, Exp3 does not depend on observing rewards for *all* experts at each decision point. Exp3 thus is able to handle domains where the consequence of only *one* prediction is observed, e.g. domains like robot motion control where predictions execute physical actions. The conclusion of this section will then present the specific details of data source selection under the DWL algorithm.

7.1.2.1. Policy Selection. At each timestep, a single policy is selected, and its action prediction is executed. To perform the policy selection, data source ξ_j is sampled at random from a uniform distribution, weighted by normalized source weight $w_{\xi_j} \in \mathbf{w}$ and kernelized distance weight. This kernelized weight $\phi(\mathbf{z}^t, :)$ is defined as in Equation 2.2, and is computed between query point $\mathbf{z}^t$ and all dataset observations $\mathbf{z}_i \in D_{\xi_j}$.

To make a prediction, data sources may employ any sort of policy derivation technique appropriate to the underlying data, for example classification or regression techniques. Our specific motion control implementation considers continuous-valued predictions and employs the Locally Weighted Learning regression technique defined in Chapter 2 (Sec. 2.1.1.3). The observation-scaling parameter Σ^{-1} is a constant diagonal matrix that scales each observation dimension to within $[0, 1]$. The matrix further embeds the width of the Gaussian kernel, which is tuned through Leave One Out Cross Validation (LOOCV) on the least squared error of the dataset predictions.

7.1.2.2. Expert Learning for Selection. Data source selection in DWL takes an *expert learning* approach. Originally proposed by Robbins (1952) to solve the k-armed bandits problem, expert learning addresses the issue of choosing between multiple action recommenders, or *experts*. Under this paradigm, executing an action receives reward, for which the recommending expert is credited. An expert's selection probability is determined by its accumulated reward, such that high reward, and therefore good performance, increases the probability of being selected. DWL treats different data sources as different experts.

Formally, at each decision cycle, t, each of k experts makes a recommendation. The algorithm selects a single expert and executes the corresponding action, resulting in payoff $r^t \in \Re$. After m decision cycles, a sequence of $r^1, r^2, \cdots, r^m$ payoffs have been awarded. The aim of expert learning is to select the best expert over all decision cycles. The learning objective is formulated in terms

of *regret*, or the difference between reward r^t of the selected action and reward r_b^t of the action recommended by the best expert. Summed over all decision cycles,

$$Regret = \sum_{t=1}^{m} r_b^t - \sum_{t=1}^{m} r^t. \tag{7.1}$$

The goal of expert learning is to minimize this total regret.

When action execution takes a physical form as in robot applications, however, only a single action may be executed and thus the reward for this action alone is observed. The rewards that would have been received by the other experts are *not* observable. Learning thus becomes a partial information game. Algorithm Exp3 (Auer et al., 1995) handles partial information games by scaling reward inversely with selection probability. The effective reward $\hat{r}^t$ earned by selected expert i from reward r^t is thus

$$\hat{r}^t = \frac{r^t}{P\left(\psi_i^t\right)} \ , \ P\left(\psi_i^t\right) = \frac{w_i^t}{\sum_{j=1}^{k} w_j^t} \tag{7.2}$$

where $P(\psi_i^t)$ is the probability of selecting expert i at decision cycle t, $\psi_i^t \equiv I(\xi^t =^? \xi_i)$ is an indicator on whether source ξ^t is equal to the source ξ_i and w_i^t is the selection weight of expert i. This compensates for the fact that experts with low probability are infrequently chosen, and therefore have fewer observed rewards. The selection weight of expert i then updates to

$$w_i^t = e^{\hat{r}^t} w_i^{t-1} \tag{7.3}$$

with weights being initially equal across experts. Note that the exponent product is equivalent to adding $\hat{r}_i^t$ to $\sum_{t=1..m} \hat{r}_i^{t-1}$, and thus represents the cumulative reward received by expert i up to trial t.

7.1.2.3. The Dynamic Weight Update in DWL. To update data source selection weights, DWL models its approach on the Exp3 expert learning algorithm. Similar to Exp3, reward is scaled inversely with data source selection probability. The DWL algorithm further makes two novel contributions to this weight update, relating to the distribution of expert rewards and determination of the expert selection probabilities.

The DWL algorithm receives a single reward r in response to a learner execution. This single execution, however, consists of a *sequence* of expert selections, which occur at each timestep. Reward therefore is not received at every decision cycle. Furthermore, multiple experts may contribute to a single execution, and accordingly also to the received reward.

To address this issue, reward is *distributed* amongst all experts that contributed to an execution. Let n_{ξ_i} be the number of execution points for which ξ_i was the selected expert. The contribution of expert ξ_i to the execution is computed as the fractional number of execution points for which ξ_i was the recommending expert, n_{ξ_i}/n_{Ξ}, $n_{\Xi} = \sum_j n_{\xi_j}$. This contribution then combines with observed reward r^t, so that the individual reward r_i^t received by expert i is computed according to

$$r_{\xi_i} = r\left(\frac{n_{\xi_i}}{n_\Xi}\right),\ n_\Xi = \sum_j n_{\xi_j} \tag{7.4}$$

This reward is then further scaled by the inverse selection probability of expert ξ_i, as in Equation 7.2:

$$\hat{r}_{\xi_i} = \frac{r_{\xi_i}}{P\left(\psi^t_{\xi_i}\right)} \tag{7.5}$$

Under DWL, expert selection probability is governed by two factors. The first is selection weight $\mathbf{w}$, as defined in Section 7.1.2.2. The second is the *distribution* of data in the state space. Sources are not assumed to contribute data uniformly across the state-space and, task proficiency aside, should not be trusted to make predictions in areas where they have no support.

To address this, the selection probability $P(\psi^t_{\xi_i})$ of expert ξ_i is formulated based on the distance between dataset D_{ξ_i} and the query point, in addition to the selection weight w_{ξ_i}. Concretely, given observation $\mathbf{z}^t$, the probability of selecting source ξ_i at timestep t is given by

$$P\left(\psi^t_{\xi_i}\right) = \frac{p\left(\psi^t_{\xi_i}|\mathbf{z}^t\right)}{\sum_{\xi_j \in \Xi} p\left(\psi^t_{\xi_j}|\mathbf{z}^t\right)} \tag{7.6}$$

$$p\left(\psi^t_i|\mathbf{z}^t\right) = w_{\xi_i} \cdot \min_{\xi_j} |\mathbf{z}^t - \mathbf{z}_j|,\ \mathbf{z}_j \in D_{\xi_i}$$

where $w_{\xi_i} \in \mathbf{w}$ is the weight of source ξ_i and $|\mathbf{z}^t - \mathbf{z}_j|$ computes the Euclidean distance between query point $\mathbf{z}^t$ and each datapoint $\mathbf{z}_j \in D_{\xi_i}$ contributed by source ξ_i.

7.2. Empirical Simulation Implementation

This section presents the details and results of applying the DWL algorithm to a simulated robot racetrack driving task. Sources are found to be unequally reliable in this domain, and source weighting is shown to improve policy performance (Argall et al., 2009a).

7.2.1. Experimental Setup

Empirical validation of the DWL algorithm is performed within a simulated robot racetrack driving domain. First described are the task and domain, followed by the employed data sources and evaluation metrics.

7.2.1.1. Racetrack Driving Task and Domain. In this domain, the robot is tasked with driving along a racetrack while staying within the track bounds. Task execution ends when the robot either completes the track ($10.34m$ length) or drives out of bounds. Note that this is a simplified, earlier version, of the racetrack driving domain presented in Chapter 6.

Robot sensing within the domain consists of three range-finders (oriented forward, left and right of the robot body). Motion dynamics are governed by robot heading and linear speed. The system runs at $30Hz$, and both changes heading and linear acceleration are bounded ($0.6m/s$ and

$0.52rad$, respectively). Gaussian noise (5%) is added to observed range information, robot heading and executed speed.

During policy execution, the observations computed by the robot are 5-dimensional: left sensor range (lr), center sensor range (cr), right sensor range (rr), a ratio of the right and left ranges (lr/rr) and a flag indicating which track border (left or right) is observed by the center sensor. This final dimension is binary; all other dimensions are continuous-valued. The actions predicted by the robot are 2-dimensional and both continuous-valued: target linear speed ν and change in heading $\Delta\theta$. Teacher demonstrations are performed via teleoperation, with the teacher commands indicating to (in/de)crease linear speed or (in/de)crease heading.

7.2.1.2. Data Sources. In this domain, three data sources are available to the learner. One source results from teacher demonstrations; the other two result from data synthesis via learner executions and teacher feedback. In summary, the three sources are:

Source Demo: Produced from teacher demonstration (D_{ξ_D}).

Source FeedbackP: Produced from positive credit feedback (D_{ξ_P}).

Source FeedbackC: Produced from corrective feedback (D_{ξ_C}).

Data from *Source Demo* is produced by human teacher teleoperation of the simulated robot during task execution, and thus is identically the teacher teleoperation demonstrations of the previous chapters. *Source FeedbackP* is produced from positive credit feedback, and thus from unmodified learner policy executions. Data from this source is a result of the teacher indicating subsets Φ of the learner execution which exhibited good performance. The recorded execution data from the subsets $\{\mathbf{z}^{\varphi}, \mathbf{a}^{\varphi}\} \in \Phi$ is then added to the dataset of *Source FeedbackP*.

Source FeedbackC produces data from corrective feedback, and thus from advice-modified learner policy executions. Here the teacher indicates poorly performing subsets Φ of the execution, as well as advice-operators to correct those subsets, and the advice-modified data is added to the dataset of *Source FeedbackC*. Three advice-operators were developed for this domain, shown in Table 7.1. The development of these operators followed our principled approach presented in Chapter 4 (Sec. 4.2). Note however that for this simplified implementation of the racetrack driving task, only a subset of the baseline action operators were employed; these are operators 0 and 1. Operator 2 was developed from the scaffolding of these baseline operators (Sec. 4.2.2).

	Operator Description	Parameter
0	Modify heading, static	[dec inc zero]
1	Modify linear speed, static	[dec inc zero]
2	Curve	[dec inc]

TABLE 7.1. Advice-operators for the simplified racetrack driving task.
[*Key*: (dec/inc)=$(de/in)crease$]

7.2.1.3. Evaluation Metrics. The evaluation metrics considered within this domain are execution success and speed. *Success* is measured by the fractional completion of the track. *Speed* is measured by the mean executed linear speed.

The weight update in DWL requires that the system provide rewards for learner executions. Reward in our system is formulated as a simple combination of success and speed. In particular, given fractional task completion ℓ^t and average linear speed ν^T for execution T, the reward received by that execution is computed as

$$r \;=\; \left(\ell^T - \bar{\ell}\right) + \left(\nu^T - \bar{\nu}\right) \tag{7.7}$$

where $\bar{\ell}$ and $\bar{\nu}$ are discounted summations of past success and speed respectively, such that $\bar{\ell} \leftarrow \bar{\ell} + \beta\left(\ell^T - \bar{\ell}\right)$ and $\bar{\nu} \leftarrow \bar{\nu} + \beta\left(\nu^T - \bar{\nu}\right)$ (here $\beta = 0.1$);

During learner practice, source selection weights are continually updated as new data is provided from various sources. A final source weighting $\mathbf{w}_f$ and dataset $\mathbf{D}_f$ are the result. For evaluation purposes, we derive a variety of policies from $\mathbf{D}_f$, defined in Table 7.2. In particular, the DWL policy *All Learned* derives from the full set $\mathbf{D}_f$ and uses learned source weights $\mathbf{w}_f$. To examine the effect of the learned weighting, policy *All Equal* also derives from the full set $\mathbf{D}_f$ but weights all sources equally. To examine data source reliability, policies derived from the data of exclusively one source, D_{ξ_D},D_{ξ_P}, or D_{ξ_C}, are also developed, respectively referred to as policies *Source Demo*, *Source FeedbackP*, and *Source FeedbackC*.

Data	Weight	Policy Name
$\mathbf{D}_f$	$\mathbf{w}_f$	*All Learned*
$\mathbf{D}_f$	equal	*All Equal*
$D_{\xi_D} \in \mathbf{D}_f$	-	*Source Demo*
$D_{\xi_P} \in \mathbf{D}_f$	-	*Source FeedbackP*
$D_{\xi_C} \in \mathbf{D}_f$	-	*Source FeedbackC*

TABLE 7.2. Policies developed for the empirical evaluation of DWL.

7.2.2. Results

This section presents empirical results from the DWL implementation. Data sources are demonstrated to be unequally reliable in this domain. The automatic data source weighting under DWL is shown to outperform an equal source weighting scheme, and furthermore to perform as well as the best contributing expert.

7.2.2.1. Unequal Data Source Reliability. To explore the reliability of each data source, track executions were performed using policies derived exclusively from one source (policies *Source C*, *Source P* and *Source C*). The results of these executions are presented in Figure 7.2 (solid bars). The performances of the three sources differ in the measures of both success and speed, confirming that in this domain the multiple data sources are indeed not equally reliable.

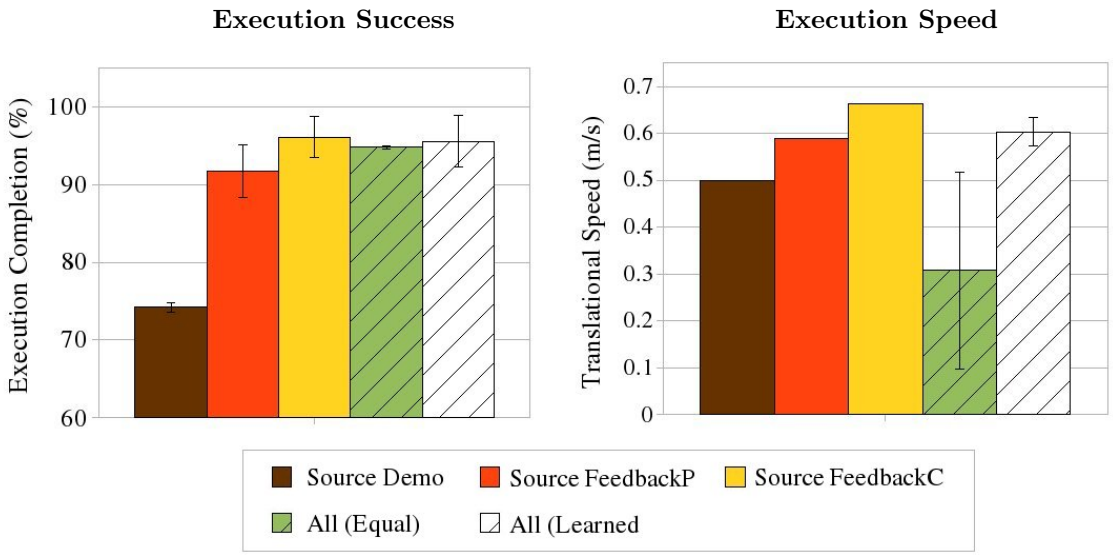

FIGURE 7.2. Mean percent task completion and translational speed with exclusively one data source (solid bars) and all sources with different weighting schemes (hashed bars).

7.2.2.2. Performance Improvement with Weighting. To examine the effects of data source weighting, executions were first performed using a policy with equal source weights (policy *All Equal*). Figure 7.2 shows that according to the success metric (left plot), the performance of equal weighting policy (green hashed bar) does match that of the best expert (yellow solid bar). On the measure of speed however (right plot), the equal weighting policy underperforms *all* experts.

By contrast, the policy that used source weights $\mathbf{w}_f$ learned under the DWL algorithm (policy *All Learned*) was able to improve upon the equal weight performance to outperform two of the three experts on average in execution speed. The average performance over 20 test executions with the learned weights $\mathbf{w}_f$ is shown in Figure 7.2 (white hashed bars). In execution speed, the learned weight policy displays superior performance over the equal weight policy ($0.61 \pm 0.03 \frac{m}{s}$ *vs.* $0.31 \pm 0.21 \frac{m}{s}$). In success, similar behavior is seen, with both giving near perfect performance. Note that the success of the learned weight policy is more variable, however, indicated by a larger standard deviation. This variability is attributed to the increased aggressiveness of the learned weight policy, which gambles occasionally running off the track in exchange for higher execution speeds.

Beyond improving on the performance of the equal weight policy, the learned weight policy furthermore begins to approach the performance of the best performing data source. The DWL algorithm thus is able to combine information from *all* of the data sources in such a manner as to outperform or match the performances of most contributing experts, and to approach the performance of the best expert.

7.2.2.3. Automatically Learned Source Weights. Selection weights are learned iteratively, throughout practice. Figure 7.3 presents the iterative selection weights as they are learned, across practice runs (solid lines). The learned weights appropriately come to favor Source FeedbackC, which is the best performing expert (Fig. 7.2, yellow bars). For reference, also shown is the fractional number of points from each source within the full set $\mathbf{D}$ (dashed lines); this fractional

composition changes as practice incorporates new data from various sources. Note that not all data sources are available at the start of practice, since the feedback data sources (FeedbackP and FeedbackC) produce demonstration data from *learner* executions, which do not exist before practice begins.

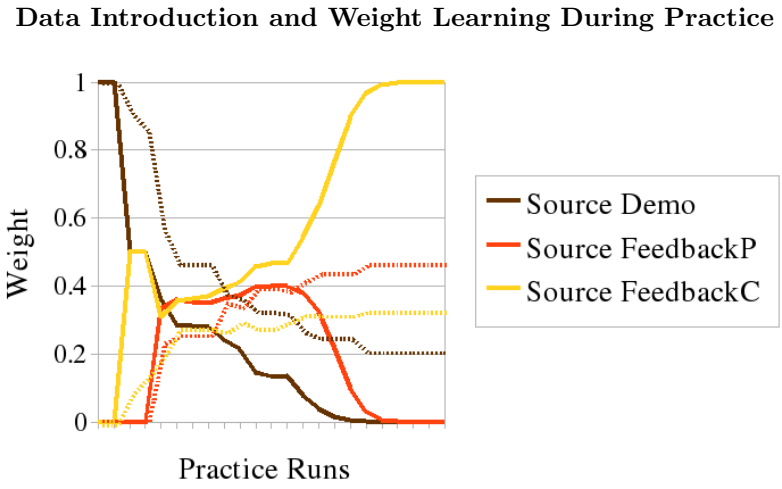

FIGURE 7.3. Data source learned weights (solid lines) and fractional population of the dataset (dashed lines) during the learning practice runs.

Figure 7.4 presents the track completion success and mean execution speeds of the iterative policies under development, during the practice runs (running average, 23 practice runs). Both measures are shown to improve as a result of learner practice, and thus with the introduction of new data and adjustment of the source weighting.

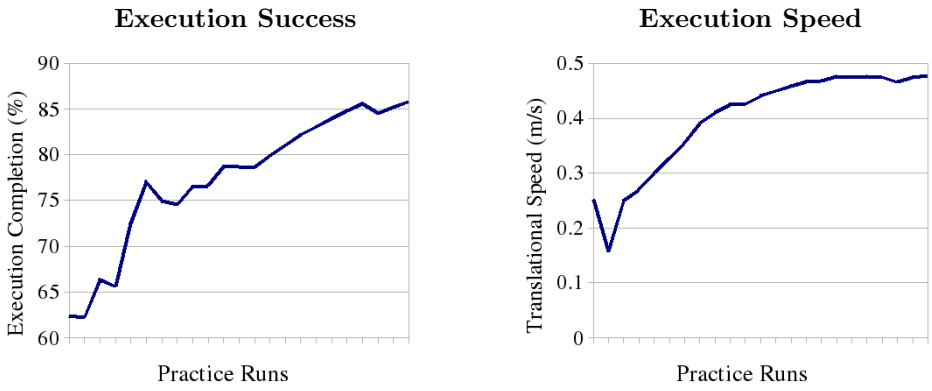

FIGURE 7.4. Percent task completion and mean translational speed during the practice runs.

7.3. Discussion

A discussion of the DWL algorithm and the above empirical implementation is presented in this section. To begin, the conclusions of this work are detailed, followed by a discussion of future directions for the DWL algorithm.

7.3.1. Conclusions

This section draws together conclusions from this empirical implementation of the DWL algorithm. First summarized are aspects of the algorithm that are key to its use as a tool for considering the reliability of multiple demonstration and feedback sources. Next a discussion of DWL's use of an execution reward is presented.

7.3.1.1. Weighting Data Sources and Feedback Types. The empirical results confirm that the various data sources are unequally reliable within this domain (Fig. 7.2). Important to remember is that the learner is *not* able to choose the source from which it receives data. Furthermore, even a poor data source can be useful, if it is the only source providing data in certain areas of the state-space. DWL addresses both of these concerns. Though the learner is not able to choose the data source, it is able to favor data from certain sources through the DWL weighting scheme. Furthermore, when selecting a data source at prediction time, both this weight *and* the state-space support of the data source are considered.

Since the expert selection probability formulation of DWL also depends on data support, even a weighting scheme that strongly favors one source, will allow for the selection of other sources. For example, in Figure 7.3 the learned weights come to strongly favor Source FeedbackC. A source other than Source FeedbackC may be selected in areas of the state-space where Source FeedbackC has not provided data, and thus where the dataset associated with Source FeedbackC is lacking in data support. This accounts for the performance differences between policies *Source FeedbackC* and *All Learned* in Figure 7.2. Though the learned weight policy strongly favors selecting the predictions of Source FeedbackC, in areas of the state space unsupported by the data contributed by Source FeedbackC the policy considers the predictions of other sources. Interesting to observe is that in these experiments, the consideration of the other sources actually degrades performance, suggesting that even the weakly data-supported predictions of Source FeedbackC are superior to the strongly-supported predictions of the inferior sources.

This empirical implementation of DWL has furthermore contributed a first evaluation of potential variability between the quality of different feedback types. Within this validation domain, feedback types were in fact shown to differ in performance ability. Corrective feedback was found to be not only the best performing feedback type, but also the best performing data source overall.

7.3.1.2. Reward Formulation and Distribution. The DWL formulation receives a single reward for an entire execution. Important to underline is that the algorithm does not require a reward at every execution *timestep*. The reward therefore only needs to evaluate overall performance, and be sufficiently rich to learn the data source *weights*; it is not necessary that the reward be sufficiently rich to learn the *task*.

The experimental domain presented here provides reward as a function of performance metrics unrelated to world state. The execution reward, therefore, *is not* a state reward. In this case, the DWL reward distribution assumes each *expert* to have contributed to the performance in direct proportion to the number of actions they recommended. By contrast, if the execution reward *is* a state reward, then the reward distribution formulation of DWL assigns equal reward to each *state* encountered during the execution. In this case, an alternate approach to reward distribution would be required; ideally one that furthermore considers reward back-propagation to earlier states.

7.3.2. Future Directions

There are many areas for the extension and application of the DWL algorithm. This section identifies a variety of such areas, including alternative approaches to source weighting, a larger number of more diverse data sources and the further evaluation of different feedback types.

Expert selection probabilities under DWL depend on data support and overall task performance. An interesting extension to the algorithm could further anchor task performance measures to state, and thus consider the *varying* performance abilities of experts in different areas of the state space. More complex domains and tasks easily could require very different skills in distinct state-space areas in order to produce good overall task performance. If demonstration teachers perform well on distinct subsets of these skills, then anchoring task performance measures to areas of the state space becomes particularly relevant.

Most of the data sources in our empirical implementation derived from the data synthesis techniques of various feedback types, while only one source derived from actual teacher demonstrations. As discussed in the motivation for this algorithm, multiple teachers contributing demonstrations is a real consideration for LfD policy development. In more complex domains, multiple teachers may even be a requirement in order to develop a policy that exhibits good performance, if the full skill set required of the task is not within the capabilities of any single teacher. The inclusion of data from multiple teachers with different demonstration styles and skill strengths is thus an interesting future application for the DWL algorithm.

This work has contributed a first evaluation of the performance abilities of different feedback types. Just as human demonstrators may vary in their respective abilities to perform certain tasks, so may feedback types be more appropriate for certain domains than for others. Some feedback types may be better suited for particular robot platforms or tasks; others may be universally inferior to the best performing feedback types. Further analysis and evaluation of the performance strengths and weaknesses of various feedback types is identified as an open area for future research.

7.4. Summary

This chapter has introduced Demonstration Weight Learning (DWL) as an algorithm that considers different types of feedback to be distinct data sources and through a weighting scheme incorporates these sources, and thus also the teacher feedback, into a policy able to perform a complex task. Data sources are selected through an expert learning inspired paradigm, where each source is treated as an expert and selection depends on both source weight and data support.

Source weights are automatically determined and dynamically updated by the algorithm, based on the execution performance of each expert.

The DWL algorithm was validated within a simulated robot racetrack driving domain. Empirical results confirmed data sources to be unequal in their respective performance abilities of this task. Data source weighting was found to improve policy performance, in the measures of both execution success and speed. Furthermore, source weights learned under the DWL paradigm were consistent with the respective performance abilities of each individual expert.

The differing performance reliabilities between demonstration data sources were discussed, as well as the subsequent use of a weighting scheme. Future research directions were identified to include the development of alternate techniques for weighting data sources. Also identified for future research was the application of DWL to complex domains requiring multiple demonstration teachers with varied skill sets, as well as further investigations that evaluate and compare different feedback types.

CHAPTER 8

Robot Learning from Demonstration

LEARNING from Demonstration is a technique that provides a learner with examples of behavior execution, from which a policy is derived.[1] There are certain aspects of LfD which are common among all applications to date. One is the fact that a teacher demonstrates execution of a desired behavior. Another is that the learner is provided with a set of these demonstrations, and from them derives a policy able to reproduce the demonstrated behavior.

However, the developer still faces many design choices when developing a new LfD system. Some of these decisions, such as the choice of a discrete or continuous action representation, may be determined by the domain. Other design choices may be up to the preference of the programmer. These design decisions strongly influence how the learning problem is structured and solved, and will be highlighted throughout this chapter. To illustrate these design choices, the discussion of this chapter is paired with a running *pick and place* example in which a robot must move a box from a table to a chair. To do so, the object must be 1) picked up, 2) relocated and 3) put down. Alternate representations and/or learning methods for this task will be presented, to illustrate how these particular choices influence task formalization and learning.

This chapter contributes a framework[2] for the categorization of LfD approaches, with a focus on robotic applications. The intent of our categorization is to highlight differences between approaches. Under our categorization, the LfD learning problem segments into two fundamental phases: *gathering* the examples, and *deriving* a policy from them. Sections 8.1 and 8.2 respectively describe these phases. We additionally point the reader to our published survey article, Argall et al. (2009b), that presents a more complete categorization of robot LfD approaches into this framework.

---

[1] Demonstration-based learning techniques are described by a variety of terms within the published literature, including Learning by Demonstration (LbD), Learning from Demonstration (LfD), Programming by Demonstration (PbD), Learning by Experienced Demonstrations, Assembly Plan from Observation, Learning by Showing, Learning by Watching, Learning from Observation, behavioral cloning, imitation and mimicry. While the definitions for some of these terms, such as imitation, have been loosely borrowed from other sciences, the overall use of these terms is often inconsistent or contradictory across articles. This thesis refers to the general category of algorithms in which a policy is derived based on demonstrated data as *Learning from Demonstration (LfD)*.

[2] This framework was jointly developed with Sonia Chernova.

8.1. Gathering Examples

This section discusses various techniques for executing and recording demonstrations. The LfD dataset is composed of state-action pairs recorded during teacher executions of a desired behavior. Exactly *how* the examples are recorded, and *what* the teacher uses as a platform for the execution, varies greatly across approaches. Examples range from sensors on the robot learner recording its own actions as it is passively teleoperated by the teacher, to a camera recording a human teacher as she executes the behavior with her own body. We formulate the LfD problem as in Chapter 2 (Sec. 2.1.1.1).

Figure 8.1 introduces our categorization for the various approaches that build a demonstration dataset, and will be referenced throughout the following subsections. This section begins with a presentation of the design decisions to consider when gathering examples, following by a discussion of potential issues in the transfer of information from the teacher to the learner. The various techniques for gathering examples are then detailed.

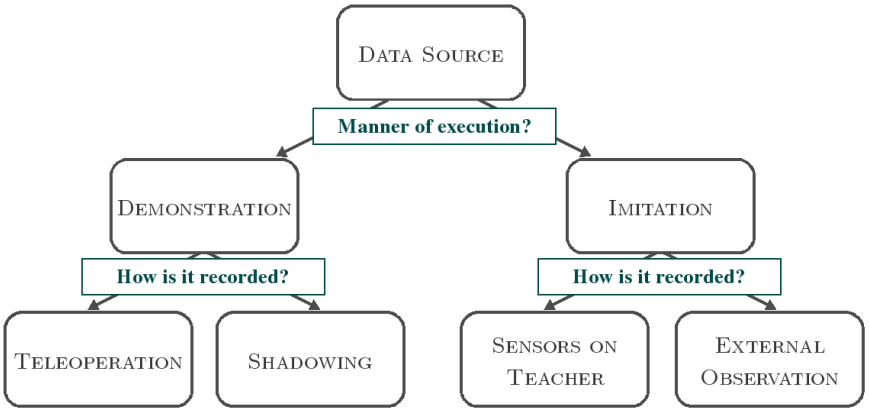

Figure 8.1. Categorization of approaches to building the demonstration dataset.

8.1.1. Design Decisions

Within the context of gathering teacher demonstrations, we identify two key decisions that must be made: the choice of *demonstrator*, and the choice of *demonstration technique*. Note that these decisions are at times affected by factors such as the complexity of the robot and task. For example, teleoperation is rarely used with high degree of freedom humanoids, since their complex motions are typically difficult to control via joystick.

Most LfD work to date has made use of human demonstrators, although some techniques also examine the use of robotic teachers, hand-written control policies and simulated planners. The choice of demonstrator further breaks down into the subcategories of (i) who *controls* the demonstration and (ii) who *executes* the demonstration.

For example, consider a robot learning to move a box, as described above. One demonstration approach could have a robotic teacher pick up and relocate the box using its own body. In this case a *robot* teacher controls the demonstration, and its *teacher* body executes the demonstration. An

alternate approach could have a human teacher teleoperate the robot learner through the task of picking up and relocating the box. In this case a *human* teacher controls the demonstration, and the *learner* body executes the demonstration.

The choice of demonstration technique refers to the *strategy* for providing data to the learner. One option is to perform batch learning, in which case the policy is learned only once all data has been gathered. Alternatively, interactive approaches allow the policy to be updated incrementally as training data becomes available, possibly provided in response to current policy performance.

8.1.2. Correspondence

For LfD to be successful, the states and actions in the learning dataset must be usable by the student. In the most straightforward setup, the states and actions of the teacher executions map directly to the learner. In reality, however, this will often not be possible, as the learner and teacher will likely differ in sensing or mechanics. For example, a robot learner's camera will not detect state changes in the same manner as a human teacher's eyes, nor will its gripper apply force in the same manner as a human hand. The challenges which arise from these differences are referred to broadly as *Correspondence Issues* (Nehaniv and Dautenhahn, 2002).

The issue of correspondence deals with the identification of a mapping from the teacher to the learner which allows the transfer of information from one to the other. We define correspondence with respect to two mappings, shown in Figure 8.2:

- The *Record Mapping* (Teacher Execution → Recorded Execution) refers to the states/actions experienced by the teacher during demonstration. This mapping is the identity $I(z, a)$ when the states/actions experienced by the teacher during execution are directly recorded in the dataset. Otherwise this teacher information is encoded according to some function $g_R(z, a) \neq I(z, a)$, and this *encoded* information is recorded within the dataset.
- The *Embodiment Mapping* (Recorded Execution → Learner) refers to the states/actions that the learner would observe/execute. When this mapping is the identity $I(z, a)$, the states/actions in the dataset map directly to the learner. Otherwise the mapping consists of some function $g_E(z, a) \neq I(z, a)$.

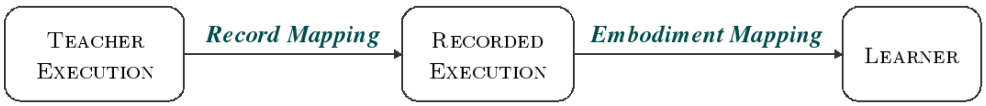

Figure 8.2. Mapping a teacher execution to the learner.

For any given learning system, it is possible to have neither, either or both of the record and embodiment mappings be the identity. Note that the mappings do not change the content of the demonstration data, but only the reference frame within which it is represented. The intersection of these configurations is shown in Figure 8.3 and discussed further within subsequent sections. The left and right columns represent identity (Demonstration) and non-identity (Imitation) embodiment

mappings, respectively. Each column is subdivided by identity (top) or non-identity (bottom) record mappings. The quadrant contents identify data gathering approaches.

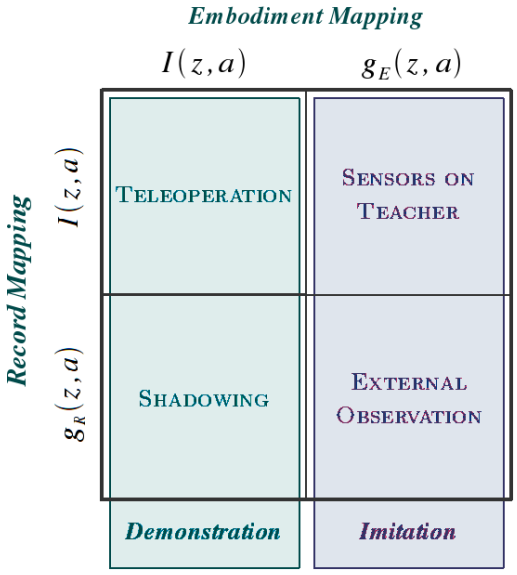

FIGURE 8.3. Intersection of the record and embodiment mappings.

The embodiment mapping is particularly important when considering real robots, compared with simulated agents. Since actual robots execute real actions within a physical environment, providing them with a demonstration involves a physical execution by the teacher. Learning within this setting depends heavily upon an accurate mapping between the recorded dataset and the learner's abilities.

Recalling again our box relocation example, consider a human teacher using her own body to demonstrate moving the box, and that the demonstration is recorded by a camera. Let the teacher actions, A_T, be represented as human joint angles, and the learner actions, A_L, be represented as robot joint angles. In this context, the teacher's demonstration of the task is observed by the robot through the camera images. The teacher's exact actions are unknown to the robot; instead, this information must be extracted from the image data. This is an example of a $g_R(z,a) \neq I(z,a)$ *record* mapping, $A_T \rightarrow D$. Furthermore, the physical embodiment of the teacher is different from that of the robot and his actions (A_T) are therefore *not* the same as those of the robot (A_L). Therefore in order to make the demonstration data meaningful for the robot, a mapping $D \rightarrow A_L$ must be applied to convert the demonstration into the robot's frame of reference. This is one example of a $g_E(z,a) \neq I(z,a)$ *embodiment* mapping.

Our categorization of LfD data source groups approaches according to the absence or presence of the record and embodiment mappings. We first split LfD data acquisition approaches into two categories based on the embodiment mapping, and thus by execution platform:

- *Demonstration*: There is no embodiment mapping, because demonstration is performed on the actual robot learner (or a physically identical platform). Thus $g_E(z,a) \equiv I(z,a)$.

- *Imitation*: There exists an embodiment mapping, because demonstration is performed on a platform which is *not* the robot learner (or a not physically identical platform). Thus $g_E(z, a) \neq I(z, a)$.

Approaches are then further distinguished within each of these categories according to record mapping (Fig. 8.1). We categorize according to these mappings to highlight the levels at which correspondence plays a role in demonstration learning. Within a given learning approach, the inclusion of each additional mapping introduces a potential injection point for correspondence difficulties; in short, the more mappings, the more difficult it is to recognize and reproduce the teacher's behavior. However, mappings also reduce constraints on the teacher and increase the generality of the demonstration technique.

8.1.3. Demonstration

When teacher executions are *demonstrated*, by the above definition there exists no embodiment mapping issue between the teacher and learner (left-hand column of Fig. 8.3). There may exist a non-direct record mapping, however, for state and/or actions. This occurs if the states experienced (actions taken) by the demonstrator are not recorded directly, and must instead be inferred from the data.

8.1.3.1. Descriptions. Based on this record mapping distinction, we identify two methods for providing demonstration data to the robot learner:

- *Teleoperation*: A demonstration technique in which the robot learner platform is *operated* by the teacher, and the robot's sensors record the execution. The record mapping is direct; thus $g_R(z, a) \equiv I(z, a)$.

- *Shadowing*: A demonstration technique in which the robot learner records the execution using its own sensors while attempting to match or *mimic* the teacher motion as the teacher executes the task. The record mapping is *not* direct; thus $g_R(z, a) \neq I(z, a)$.

During teleoperation, a robot is operated by the teacher while recording from its own sensors. Teleoperation provides the most direct method for information transfer within demonstration learning. However, teleoperation requires that operating the robot be manageable, and as a result not all systems are suitable for this technique. For example low-level motion demonstrations are difficult on systems with complex motor control, such as high degree of freedom humanoids. The strength of the teleoperation approach is the direct transfer of information from teacher to learner, while its weakness is the requirement that the robot be operated in order to provide a demonstration.

During shadowing, the robot platform mimics the teacher's demonstrated motions while recording from its own sensors. The states/actions of the true demonstration execution are not recorded; rather, the learner records its own mimicking execution, and so the teacher's states/actions are *indirectly* encoded within the dataset. In comparison to teleoperation, shadowing requires an extra algorithmic component that enables the robot to track and actively shadow (rather than passively

be teleoperated by) the teacher. The technique does not, however, require that the teacher be able to operate the robot in order to provide a demonstration.

Also important to note is that demonstration data recorded by real robots frequently does not represent the *full* observation state of the teacher. This occurs if, while executing, the teacher employs extra sensors that are not recorded. For example, if the teacher observes parts of the world that are inaccessible from the robot's cameras (e.g. behind the robot, if its cameras are forward-facing), then state, as observed by the teacher, differs from what is actually recorded as data.

8.1.3.2. Implementations. Teleoperation is very effective at reducing the effects of teacher-learner correspondence on the demonstrations, but does require that actively controlling the robot during the task be manageable, which might not be the case. Shadowing has the advantage of not requiring the teacher to actively control the robot, but does require that the learner be able to identify and track the teacher; furthermore, the observations made by the teacher during the execution are not directly recorded.

Demonstrations recorded through human teleoperation via a joystick are used in a variety of applications, including flying a robotic helicopter (Ng et al., 2004), robot kicking motions (Browning et al., 2004), object grasping (Pook and Ballard, 1993; Sweeney and Grupen, 2007), robotic arm assembly tasks (Chen and Zelinsky, 2003) and obstacle avoidance and navigation (Inamura et al., 1999; Smart, 2002). Teleoperation is also applied to a wide variety of simulated domains, ranging from static mazes (Clouse, 1996; Rao et al., 2004) to dynamic driving (Abbeel and Ng, 2004; Chernova and Veloso, 2008a) and soccer domains (Aler et al., 2005), and many other applications.

Human teachers also employ techniques other than direct joysticking for demonstration. In kinesthetic teaching, a humanoid robot is not actively controlled but rather its passive joints are moved through desired motions (Billard et al., 2006). Demonstration may also be performed through speech dialog, where the robot is told specifically what actions to execute in various states (Breazeal et al., 2006; Lauria et al., 2002; Rybski et al., 2007). Both of these techniques might be viewed as variants on traditional teleoperation, or alternatively as a sort of high level teleoperation. In place of a human teacher, hand-written controllers are also used to teleoperate robots (Grollman and Jenkins, 2008; Rosenstein and Barto, 2004; Smart, 2002).

Demonstrations recorded through shadowing teach navigational tasks by having a robot follow an identical-platform robot teacher through a maze (Demiris and Hayes, 2002), follow a human teacher past sequences of colored markers (Nicolescu and Matarić, 2001a) and mimic routes determined from observations of human teacher executions (Nehmzow et al., 2007). Shadowing also has a humanoid learn arm gestures, by mimicking the motions of a human demonstrator (Ogino et al., 2006).

8.1.4. Imitation

For all *imitation* approaches, as defined above, embodiment mapping issues do exist between the teacher and learner, and so $g_E(z, a) \neq I(z, a)$ (right-hand column of Fig. 8.3). There may exist a direct or non-direct record mapping, however, for state and/or actions.

8.1.4.1. Description. Similar to the treatment of demonstration above, approaches for providing imitation data are further divided by whether the record mapping is the identity or not.

- *Sensors on Teacher* : An imitation technique in which sensors located on the executing body are used to record the teacher execution. The record mapping *is* direct; thus $g_R(z, a) \equiv I(z, a)$.
- *External Observation* : An imitation technique in which sensors external to the executing body are used to record the execution. These sensors may or may not be located on the robot learner. There record mapping *is not* direct; thus $g_R(z, a) \neq I(z, a)$.

The sensors-on-teacher approach utilizes recording sensors located *directly on* the executing platform. Having a direct record mapping alleviates one potential source for correspondence difficulties. The strength of this technique is that precise measurements of the example execution are provided. However, the overhead attached to the specialized sensors, such as human-wearable sensor-suits, or customized surroundings, such as rooms outfitted with cameras, is non-trivial and limits the settings within which this technique is applicable.

Imitation performed through external observation relies on data recorded by sensors located *externally to* the executing platform. Since the actual states/actions experienced by the teacher during the execution are not directly recorded, they must be inferred. This introduces a new source of uncertainty for the learner. Some LfD implementations first extract the teacher states/actions from this recorded data, and then map the extracted states/actions to the learner. Others map the recorded data directly to the learner, without ever explicitly extracting the states/actions of the teacher. Compared to the sensors on teacher approach, the data recorded under this technique is less precise and less reliable. The method, however, is more general and is not limited by the overhead of specialized sensors and settings.

8.1.4.2. Implementations. The sensors-on-teacher approach requires specialized sensors and introduces another level of teacher-learner correspondence, but does not require that the learner platform be actively operated or able to track the teacher during task execution. The external-sensors approach has the lowest requirements in terms of specialized sensors or actively operating the robot, but does introduce the highest levels of correspondence between the recorded teacher demonstrations and later executions with the learner platform.

In the sensors-on-teacher approach, human teachers commonly use their own bodies to perform example executions by wearing sensors able to record the human's state and actions. This is especially true when working with humanoid or anthropomorphic robots, since the body of the robot resembles that of a human. In this case the joint angles and accelerations recorded by a sensor-suit worn by the human during demonstration commonly are used to populate the dataset (Aleotti and Caselli, 2006; Calinon and Billard, 2007; Ijspeert et al., 2002a).

In the external-sensors approach, typically the sensors used to record human teacher executions are vision-based. Motion capture systems utilizing visual markers are applied to teaching human motion (Amit and Matarić, 2002; Billard and Matarić, 2001; Ude et al., 2004) and manipulation tasks (Pollard and Hodgins, 2002). Visual features are tracked in biologically-inspired frameworks that link

perception to abstract knowledge representation (Chella et al., 2006) and teach an anthropomorphic hand to play the game Rock-Paper-Scissors (Infantino et al., 2004).

A number of systems combine external sensors with other information sources; in particular, with sensors located directly on the teacher (Lopes and Santos-Victor, 2005; Lieberman and Breazeal, 2004; Matarić, 2002). Other approaches combine speech with visual observation of human gestures (Steil et al., 2004) and object tracking (Demiris and Khadhouri, 2006), within the larger goal of speech-supported LfD learning.

Several works also explore learning through the external observation of non-human teachers. A robot learns a manipulation task through visual observation of an identical robotic teacher (Dillmann et al., 1995), and a simulated agent learns about its own capabilities and unvisited parts of the state space through observation of other simulated agents (Price and Boutilier, 2003). A generic framework for solving the correspondence problem between differently embodied robots has a robotic agent learn new behaviors through imitation of another, possibly physically different, agent (Alissandrakis et al., 2004).

8.1.5. Other Approaches

Within LfD there do exist exceptions to the data source categorization we have presented here. These exceptions record only states during demonstration, without recording actions. For example, by having a human draw a path through a 2-D representation of the physical world, high level path-planning demonstrations are provided to a rugged outdoor robot (Ratliff et al., 2006) and a small quadruped robot (Ratliff et al., 2007; Kolter et al., 2008). Since actions are not provided in the dataset, no state-action mapping is learned for action selection. Instead, actions are selected at runtime by employing low level motion planners and controllers (Ratliff et al., 2006, 2007), or by providing state-action transition models (Kolter et al., 2008).

8.2. Deriving a Policy

This section discusses the various techniques for deriving a policy from a dataset of state-action examples, acquired using one of the above data gathering methods. We identify three core approaches within LfD to deriving policies from demonstration data, introduced in the following section. Across all approaches, minimal parameter tuning and fast learning times requiring few training examples are desirable.

We identify three core approaches within LfD to deriving a policy. Shown in Figure 8.4 the approaches are termed *mapping function*, *system model*, and *plans*:

- *Mapping Function*: Demonstration data is used to directly approximate the underlying function mapping from the robot's state observations to actions ($f() : Z \rightarrow A$).
- *System Model*: Demonstration data is used to determine a model of the world dynamics ($T(s'|s, a)$), and possibly a reward function ($R(s)$). A policy is then derived using this information.

- *Plans*: Demonstration data is used to associate a set of pre- and post-conditions with each action ($L(\{preC, postC\}|a)$), and possibly a sparsified state dynamics model ($T(s'|s,a)$). A planner, using these conditions, then produces a sequence of actions.

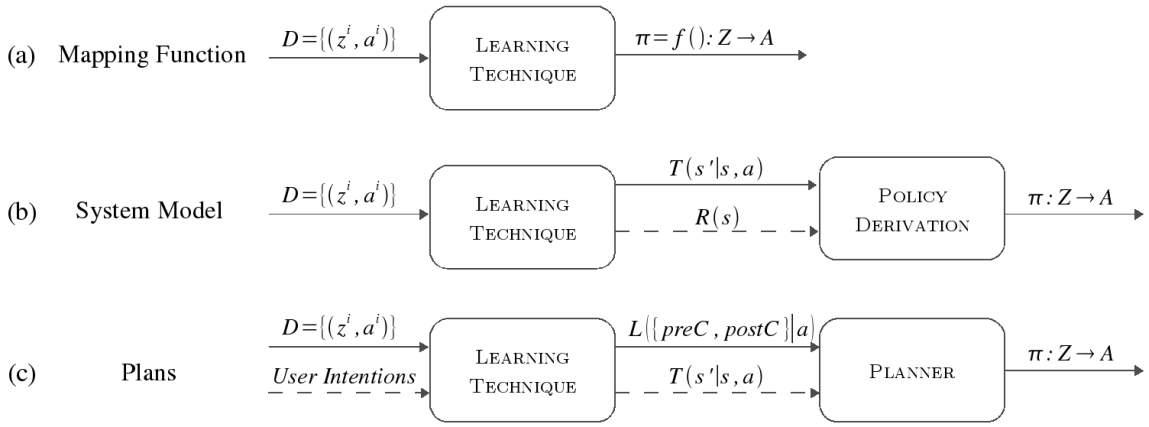

FIGURE 8.4. Typical LfD approaches to policy derivation.

8.2.1. Design Decisions

A fundamental choice for any designer of an LfD system is the selection of an algorithm to generate a policy from the demonstration data. We categorize the various techniques for policy derivation most commonly seen within LfD applications into the three core policy derivation approaches. Our full categorization is provided in Figure 8.5. Approaches are initially split between the three core policy derivation techniques. Further splits, if present, are approach-specific. This section begins with a discussion of the design decisions involved in the selection of a policy derivation algorithm, followed by the details of each core LfD policy derivation technique.

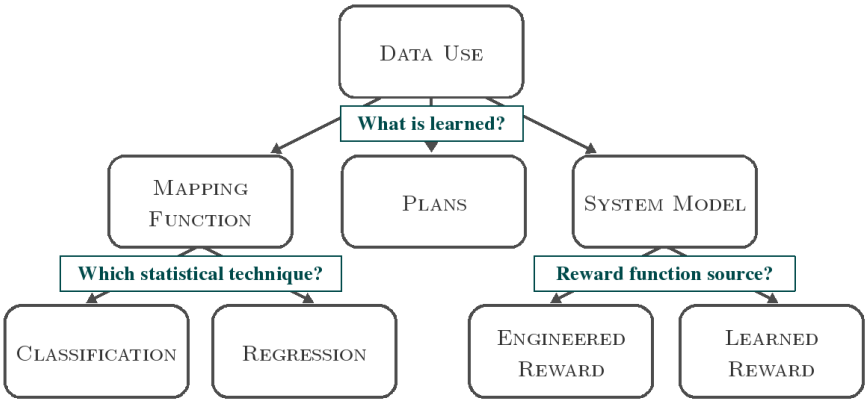

FIGURE 8.5. Categorization of approaches to learning a policy from demonstration data.

Returning again to our box moving example, suppose a mapping function approach is used to derive the policy. A function $f() : Z \rightarrow A$ is learned that maps the observed state of the world, for

example the 3-D location of the robot's end effector, to an action which guides the learner towards the goal state, for example the desired end effector motor speed. Consider instead using a system model approach. Here a state transition model $T(s'|s,a)$ is learned, for example that taking the *pick up* action when in state *box on table* results in state *box held by robot*. Using this model, a policy is derived that indicates the best action to take when in a given state, such that the robot is guided towards the goal state. Finally, consider using a planning approach. The pre- and post-conditions of executing an action $L(\{preC, postC\}|a)$ are learned from the demonstrations. For example, the *pick up* action requires the *box on table* pre-condition, and results in the *box held by robot* post-condition. A planner uses this learned information to produce a sequence of actions that end with the robot in the goal state.

8.2.2. Problem Space Continuity

When selecting a technique for policy derivation, the continuity of the action representation within the demonstration set is a key factor. Action-space continuity is determined both by the task and robot capabilities, and restricts the applicability of the various derivation approaches. For example, a policy with continuous-valued actions can be derived with regression techniques and thus under the mapping function approximation approach, but is unlikely to be represented within a planning domain where each action must be enumerated and associated with pre- and post-conditions.

The question of problem-space continuity plays a prominent role within the context of state and action representation, and many valid representations frequently exist for the same domain. Within our robot box moving example, one option could be to discretize state, such that the environment is represented by boolean features such as *box on table* and *box held by robot*. Alternatively, a continuous state representation could be used in which the state is represented by the 3-D positions of the robot's end effector and the box. Similar discrete or continuous representations may be chosen for the robot's actions. In designing a domain, the continuity of the problem space is influenced by many factors, such as the desired learned behavior, the set of available actions and whether the world is simulated or real.

We additionally note that LfD can be applied at a variety of action control levels, depending on the problem formulation. Here roughly grouped into three control levels, LfD has been applied to: low-level actions for motion control, basic high-level actions (often called action primitives) and complex behavioral actions for high-level control. Note that this is a somewhat different consideration to action-space continuity. For example, a low-level motion could be formulated as discrete or continuous, and so this single action level can map to either continuity space. As a general technique, LfD can be applied at any of these action levels. Most important in the context of policy derivation, however, is whether actions are continuous or discrete, and not their control level.

8.2.3. Mapping Function

The *mapping function* approach to policy learning calculates a function that approximates the state to action mapping, $f() : Z \rightarrow A$, for the demonstrated behavior (Fig. 8.4a).

8.2.3.1. Description. The goal of this type of algorithm is to reproduce the underlying teacher policy, which is unknown, and to generalize over the set of available training examples such that valid solutions are also acquired for similar states that may not have been encountered during demonstration.

The details of function approximation are influenced by many factors. These include: whether the state input and action output are continuous or discrete; whether the generalization technique uses the data to approximate a function prior to execution time or directly at execution time; whether it is feasible or desirable to keep the entire demonstration dataset around throughout learning, and whether the algorithm updates online.

In general, mapping approximation techniques fall into two categories depending on whether the prediction output of the algorithm is discrete or continuous. *Classification* techniques are used to produce discrete output, and *regression* techniques produce continuous output. Many techniques for performing classification and regression have been developed outside of LfD; the reader is referred to Hastie et al. (2001) for a full discussion.

8.2.3.2. Implementations. Mapping function approaches directly approximate the state-action mapping through classification or regression techniques. We begin with a presentation of classification applications, followed by regression. Since regression has been the policy derivation technique employed in this thesis, it will be presented in greater detail than the other techniques.

Classification approaches categorize their input into discrete classes, thereby grouping similar input values together. In the context of policy learning, the input to the classifier is robot states and the discrete output classes are robot actions.

Discrete low-level robot actions include basic commands such as moving forward or turning. Classifiers have been learned for a variety of tasks, including *Gaussian Mixture Models (GMMs)* for simulated driving (Chernova and Veloso, 2007), *decision trees* for simulated airplane flying (Sammut et al., 1992) and *Bayesian networks* for navigation and obstacle avoidance (Inamura et al., 1999).

Discrete high-level robot actions range from basic motion primitives to high-level robot behaviors. Motion primitives learned from demonstration include a manipulation task using k-Nearest Neighbors (k-NN) (Pook and Ballard, 1993), an assembly task using *Hidden Markov Models (HMMs)* (Hovland et al., 1996) and a variety of human torso motions using *vector-quantization* (Matarić, 2002). High-level behaviors are generally developed (by hand or learned) prior to task learning, and demonstration teaches the selection of these behaviors. For example, HMMs classify demonstrations into gestures for a box sorting task with a Pioneer robot (Rybski and Voyles, 1999), and the Bayesian likelihood method is used to select actions for a humanoid robot in a button pressing task (Lockerd and Breazeal, 2004).

Regression approaches map demonstration states to continuous action spaces. Similar to classification, the input to the regressor is robot states, and the *continuous* output are robot actions. Since the continuous-valued output often results from combining multiple demonstration set actions, typically regression approaches are applied to low-level motions, and not to high-level behaviors. A key distinction between methods is whether the mapping function approximation occurs at run time,

or prior to run time. We frame the summary of regression techniques for LfD along a continuum between these two extremes.

At one extreme on the regression continuum lies Lazy Learning (Atkeson et al., 1997), where function approximation does not occur until a current observation point in need of mapping is present. The simplest Lazy Learning technique is k-NN, which is applied to action selection within a robotic marble-maze domain (Bentivegna, 2004). More complex approaches include *Locally Weighted Regression (LWR)* (Cleveland and Loader, 1995). One LWR technique further anchors local functions to the phase of nonlinear oscillators (Schaal and Sternad, 1998) to produce rhythmic movements, specifically drumming (Ijspeert et al., 2002b) and walking (Nakanishi et al., 2004) patterns with a humanoid robot. While Lazy Learning approaches are fast and expend no effort approximating the function in areas which are unvisited during execution time, they do require keeping around all of the training data.

In the middle of the regression continuum lie techniques in which the original data is converted to another, possibly sparsified, representation prior to run time. This *converted* data is then used by Lazy Learning techniques at run time. For example, *Receptive Field Weighted Regression (RFWR)* (Schaal and Atkeson, 1998) first converts demonstration data to a Gaussian and local linear model representation. *Locally Weighted Projection Regression (LWPR)* (Vijayakumar and Schaal, 2000) extends this approach to scale with input data dimensionality and redundancy. Both RFWR and LWPR are able to incrementally update the number of representative Gaussians as well as regression parameters online. Successful robotics applications using LWPR include an AIBO robot performing basic soccer skills (Grollman and Jenkins, 2008) and a humanoid playing the game of air hockey (Bentivegna, 2004). The approaches at this position on the continuum benefit from not needing to evaluate all of the training data at run time, but at the cost of extra computation and generalization prior to execution.

At the opposite extreme on the regression continuum lie approaches which form a complete function approximation prior to execution time. At run time, they no longer depend on the presence of the underlying data (or any modified representations thereof). *Neural Network (NN)* techniques enable autonomous vehicle driving (Pomerleau, 1991) and peg-in-hole task execution with a robot arm (Dillmann et al., 1995). Also possible are statistical approaches that represent the demonstration data in a single or mixture distribution which is sampled at run time, for example a joint distribution for grasping (Steil et al., 2004), *Gaussian Mixture Regression (GMR)* for gestures (Calinon and Billard, 2007) and *Sparse On-Line Gaussian Processes (SOGP)* for soccer skills (Grollman and Jenkins, 2008). The regression techniques in this area all have the advantage of *no* training data evaluations at run time, but are the most computationally expensive prior to execution. Additionally, some of the techniques, for example NN, can suffer from "forgetting" mappings learned earlier, a problem of particular importance when updating the policy online.

8.2.4. System Models

The *system model* approach to LfD policy learning uses a state transition model of the world, $T(s'|s,a)$, and derives a policy $\pi : Z \rightarrow A$ from this model (Fig. 8.4b).

8.2.4.1. Description. This approach is typically formulated within the structure of *Reinforcement Learning (RL)*. The transition function $T(s'|s,a)$ generally is defined from the demonstration data and any additional autonomous exploration the robot may do. To derive a policy from this transition model, a reward function $R(s)$, that associates reward value r with world state s, must be either learned from demonstrations or defined by the user.

The goal of RL is to maximize cumulative reward over time. The expected cumulative future reward of the state under the current policy is represented by further associating each state s with a value according to the function $V(s)$ (or associating state-action pair s, a with a Q-value according to the function $Q(s,a)$). State values may be represented by the *Bellman Equation*:

$$V^{\pi}(s) \;=\; \int_a \pi(s,a) \int_{s'} T(s'|s,a)\,[r(s) + \gamma V^{\pi}(s')]\,ds'da \tag{8.1}$$

where $V^{\pi}(s)$ is the value of state s under policy π, and γ represents a discounting factor on future rewards. How to use these values to update a policy, and how to update them effectively online, is a subject of much research outside of LfD. Unlike function approximation techniques, RL approaches typically do *not* generalize state and demonstrations must be provided for every discrete state. For a full review of RL, the reader is referred to Sutton and Barto (1998).

Reward is the motivation behind state and action selection, and consequently also guides task execution. Defining a reward function to accurately capture the desired task behavior, however, is not always obvious. Approaches may therefore be categorized according to the source of the reward function. Some approaches *engineer* the reward function, as seen in classical RL implementations. Other approaches, *learn* a reward function from the demonstration data.

8.2.4.2. Implementations. Within most applications of the LfD system model approach, RL is employed and the reward function is manually defined by the user. User-defined rewards tend to be sparse, meaning that the reward value is zero except for a few states, such as around obstacles or near the goal. Various demonstration-based techniques therefore have been defined to aid the robot in locating the rewards and to prevent extensive periods of blind exploration.

Demonstration may be used to highlight interesting areas of the state space in domains with sparse rewards, for example using teleoperation to show the robot the reward states and thus eliminating long periods of initial exploration in which no reward feedback is acquired (Smart and Kaelbling, 2002). Both reward and action demonstrations influence the performance of the learner in the supervised actor-critic reinforcement learning algorithm (Sutton and Barto, 1998), applied to a basic assembly task using a robotic manipulator (Rosenstein and Barto, 2004).

Defining an effective reward function for real world systems, however, can be a non-trivial issue. One subfield within RL that addresses this is *Inverse Reinforcement Learning* (Russell, 1998), where the reward function is *learned* rather than hand-defined. Several LfD algorithms examine learning a reward function from demonstration. For example, a reward function is learned by associating greater reward with states similar to those encountered during demonstration, and is combined with RL techniques to teach a pendulum swing-up task to a robotic arm (Atkeson and Schaal, 1997).

Rewarding similarity between teacher demonstrated and robot-executed trajectories is one approach to learning a reward function. Example approaches include a Bayesian model formulation (Ollis et al., 2007), and another that further provides greater reward to positions close to the goal (Guenter and Billard, 2007). Trajectory comparison based on state features guarantees that a correct reward function (Abbeel and Ng, 2004), and furthermore the correct behavior (Ratliff et al., 2006), is learned from feature count estimation. Extensions of this approach decompose the task demonstration into hierarchies for a small legged robot (Kolter et al., 2008; Ratliff et al., 2007), and resolve feature count ambiguities (Ziebart et al., 2008). Both a transition function $T(s'|s,a)$ *and* reward function $R(s)$ are learned for acrobatic helicopter maneuvers (Abbeel et al., 2007).

Finally, real robot applications tend to employ RL in ways not typically seen in the classical RL policy derivation algorithms. This is because even in discrete state-action spaces the cost of visiting every state, and taking every action from each state, becomes prohibitively high and further could be physically dangerous. One approach is to use simulation to seed an initial world model, $T(s'|s,a)$, for example using a planner operating on a simulated version of a marble maze robot (Stolle and Atkeson, 2007). The opposite approach derives a model of robot dynamics from demonstration but then uses the model in simulation, for example to simulate robot state when employing RL to optimize a NN controller for autonomous helicopter flight (Bagnell and Schneider, 2001) and inverted helicopter hovering (Ng et al., 2004). The former implementation additionally pairs the hand-engineered reward with a high penalty for visiting any state not encountered during demonstration, and the latter additionally makes the explicit assumption of a limited class of feasible controllers.

8.2.5. Plans

The *plans* approach to policy learning views demonstration executions as example plans, and uses the demonstration data to model the conditions of these plans. More specifically, the policy is a plan, and is represented as a sequence of actions that lead from the initial state to the final goal state (Fig. 8.4c).

8.2.5.1. Description. Plans are typically described in terms of action pre- and post-conditions. An action *pre-condition* defines the state that must be established before the action may be executed. An action *post-condition* defines the state that results from the action execution. Typically these pre- and post-conditions are learned from the demonstration data. LfD techniques under this approach also frequently have the teacher provide information in the form of *annotations* or *intentions*, in addition to the state-action examples. Algorithms differ based on whether this additional information is provided, as well as how the rules associating action pre- and post-conditions are learned.

8.2.5.2. Implementations. One of the first papers to learn execution plans based on demonstration is Kuniyoshi et al. (1994), in which a plan is learned for object manipulation based on observations of the teacher's hand movements. Some approaches use spoken dialog both as a technique to demonstrate task plans, and also to enable the robot to verify unspecified or unsound parts of a plan through dialog (Lauria et al., 2002; Rybski et al., 2007). Other planning-based methods require teacher annotations, for example to encode plan post-conditions (Friedrich and

Dillmann, 1995), to draw attention to particular elements of the domain (Nicolescu and Matarić, 2003), or to encode information about the task goal (Jansen and Belpaeme, 2006; van Lent and Laird, 2001).

8.3. Summary

Learning from Demonstration (LfD) is an approach for deriving a policy from examples of behavior executions by a teacher. This chapter has introduced a framework (Argall et al., 2009b) for the categorization of typical robot applications of LfD. In particular, under this framework the implementation of an LfD learning system is segmented into two core phases: gathering examples, and deriving a policy from the examples. Design considerations when gathering examples include who controls and executes a demonstration, and how a demonstration is recorded. Each of these decisions influence the introduction of teacher-learner correspondence issues into the learning system. When deriving a policy, one of three core approaches is typically taken, termed here as the mapping function approximation, system model and plans approaches.

CHAPTER 9

Related Work

THE problem of learning a mapping between world state and actions lies at the heart of many robotics applications. This mapping, or policy, enables a robot to select an action based upon its current world state. The development of policies by hand is often a very challenging problem, and as a result many machine learning techniques have been applied to policy development. In this thesis, we consider the particular policy learning approach of *Learning from Demonstration (LfD)*. We focus on its application to low-level motion control domains, and techniques that augment traditional approaches to address common limitations within LfD.

This chapter presents robot LfD literature relevant to the work of this thesis. In summary, we have found within the field a lack within the field (i) of techniques that address particular LfD limitations within continuous, frequently sampled, action-space domains, (ii) of techniques that provide focused performance evaluations, (iii) of LfD applications to motion control for a *mobile* robot (iv) of algorithms that build a policy from primitive behaviors while explicitly addressing LfD limitations and (v) of approaches that incorporate data from multiple demonstration sources. In Section 9.1, LfD applications associated with topics central to the focus of this thesis are presented. Methods for then improving robot performance beyond the capabilities of the teacher examples are discussed in Section 9.2. Algorithms that explicitly address limitations within LfD, typically the cause of poor policy performance, are then presented in Section 9.3.

9.1. LfD Topics Central to this Thesis

In this section, we examine LfD applications particularly relevant to the central topics of this thesis. The topic crucially related to the work of this thesis is whether an algorithm explicitly addresses any limitations in the LfD dataset; we reserve discussion of this topic for the following sections (Secs. 9.2 and 9.3). The central topics considered here include (i) learning low-level motion control from demonstration, (ii) learning and using behavior primitives within a LfD framework and (iii) the incorporation of multiple data sources into a LfD policy.

9.1.1. Motion Control

This section discusses LfD applications to low-level motion control tasks. The focus in this section is on applications where the actual *motion* is learned from demonstration; applications that

learn from demonstration some other aspect of a motion task, for example how to select between hand-coded motion primitives, are not discussed here. We segment approaches into three motor strata: motor poses, motions for a stationary robot and motions for a mobile robot.

We define motor poses as stationary configurations of the robot, that are *formed* through motion. While the motion is learned from demonstration, it merely enables the target behavior. For example, through demonstration an anthropomorphic hand learns the hand formations and rules for the game Rock-Paper-Scissors (Infantino et al., 2004). Using GMR, a humanoid robot is taught basketball referee signals by pairing kinesthetic teachings with human teacher executions recorded via wearable motion sensors (Calinon and Billard, 2007).

The most common application of LfD to motion control is for motion tasks on a stationary robot. Early work by Atkeson and Schaal (1997) has a robotic arm learn pole balancing via stereo-vision and human demonstration. The recorded joint angles of a human teach drumming patterns to a 30-DoF humanoid (Ijspeert et al., 2002a). Motion capture systems utilizing visual markers are applied to teaching human arm motion (Amit and Matarić, 2002; Ude et al., 2004) and manipulation tasks (Pollard and Hodgins, 2002). The combination of 3D marker data with torso movement and joint angle data is used for applications on a variety of simulated and robot humanoid platforms (Matarić, 2002). A human wearing sensors control a simulated human, which maps to a simulated robot and then to a real robot arm (Aleotti and Caselli, 2006). A force-sensing glove is combined with vision-based motion tracking to teach grasping movements (Lopes and Santos-Victor, 2005; Voyles and Khosla, 2001). Generalized dexterous motor skills are taught to a humanoid robot (Lieberman and Breazeal, 2004), and demonstration teaches a humanoid to grasp novel and known household objects (Steil et al., 2004).

For demonstrated motion control on a mobile robot, a seminal work by Pomerleau (1991) used a Neural Network to enable autonomous driving of a van at speed on a variety of road types. Demonstrations contribute to the development of a controller for a robotic helicopter flight, inverted hovering and acrobatic maneuvers (Bagnell and Schneider, 2001; Ng et al., 2004; Abbeel et al., 2007). An AIBO robot is taught basic soccer skills (Grollman and Jenkins, 2008), recorded joint angles teach a humanoid walking patterns within simulation (Nakanishi et al., 2004) and a wheeled robot learns obstacle avoidance and corridor following (Smart, 2002).

9.1.2. Behavior Primitives

To build a policy from multiple smaller policies, or behavior *primitives*, is an interesting development technique to combine with LfD. This technique is attractive for a variety of reasons, including a natural framework for scaling up existing behaviors to more complex tasks and domains, and the potential for primitive reuse within other policies.

Pook and Ballard (1993) classify primitive membership using k-NN, and then recognize each primitive within the larger demonstrated task via *Hidden Markov Models (HMMs)*, to teach a robotic hand and arm an egg flipping manipulation task. Demonstrated tasks are decomposed into a library of primitives for a robotic marble maze (Stolle and Atkeson, 2007), and behavior primitives are demonstrated and then combined by the human teacher into a larger behavior (Saunders et al., 2006). The use of hand-coded behavior primitives to build larger tasks learned from demonstration

include a humanoid playing the game of air-hockey (Bentivegna, 2004) and a Pioneer performing navigation tasks (Nicolescu and Matarić, 2003).

A biologically-inspired framework is presented in Matarić (2002), which automatically extracts primitives from demonstration data, classifies data via vector-quantization and then composes and/or sequences primitives within a hierarchical NN. The work is applied to a variety of test beds, including a 20 DoF simulated humanoid torso, a 37 DoF avatar (a simulated humanoid which responds to external sensors), Sony AIBO dogs and small differential drive Pioneer robots. Under this framework, the simulated humanoid torso is taught a variety of dance, aerobics and athletics motions (Jenkins et al., 2000), the avatar reaching patterns (Billard and Matarić, 2001) and the Pioneer sequenced location visiting tasks (Nicolescu and Matarić, 2001b).

9.1.3. Multiple Demonstration Sources and Reliability

The incorporation of data from multiple demonstration sources is a topic that has received limited attention from the LfD community. Furthermore, assessing the reliability between different demonstration sources has not been explicitly addressed in the literature. A closely related topic, however, is the assessment of the worth and reliability of datapoints within the demonstration set.

Multiple demonstration teachers are employed for the purposes of isolating salient characteristics of the task execution (Pook and Ballard, 1993). Demonstration information is also solicited from multiple other agents to speed up learning (Oliveira and Nunes, 2004). Multiple sources of demonstration data on the whole, however, remains largely unaddressed.

Data worth is addressed by actively removing unnecessary or inefficient elements from a teacher demonstration (Kaiser et al., 1995). The reliability of a dataset in different areas of the state-space is assessed through the use of a confidence measure to identify undemonstrated or ambiguously demonstrated areas of the state-space (Chernova and Veloso, 2007; Grollman and Jenkins, 2007).

9.2. Limitations of the Demonstration Dataset

LfD systems are inherently linked to the information provided in the demonstration dataset. As a result, learner performance is heavily limited by the quality of this information. One common cause for poor learner performance is due to dataset sparsity, or the existence of areas of the state space that have not been demonstrated. A second cause is poor quality of the dataset examples, which can result from a teacher's inability to perform the task optimally.

9.2.1. Undemonstrated State

All LfD policies are limited by the availability of training data because, in all but the most simple domains, the teacher is unable to demonstrate the correct action from every possible state. This raises the question of how the robot should act when it encounters a state for which no demonstration has been provided. Here existing methods for dealing with undemonstrated state are categorized into two approaches: *generalization from existing demonstrations*, and *acquisition of new demonstrations*.

9.2.1.1. Generalization from Existing Demonstrations. Here techniques that use existing data to deal with undemonstrated state are presented as used within each of the core policy derivation approaches.

Within the function mapping approach, undemonstrated state is dealt with through state generalization. Generalizing across inputs is a feature inherent to robust function approximation. The exact nature of this generalization depends upon the specific classification or regression technique used. Some examples are approaches that range from strict grouping with the nearest dataset point state, as in 1-Nearest Neighbor, to soft averaging across multiple states, as in kernelized regression.

Within the system model approach, undemonstrated state can be addressed through state exploration. State exploration is often accomplished by providing the learner with an exploration policy that guides action selection within undemonstrated states. Based on rewards provided by the world, the performances of these actions are automatically evaluated. This motivates the issue of *Exploration vs. Exploitation*; that is, of determining how frequently to take exploratory actions versus following the set policy. Note that taking exploratory steps on a real robot system is often unwise, for safety and stability reasons.

Generalization to unseen states is not a common feature amongst traditional planning algorithms. This is due to the common assumption that every action has a set of known, deterministic effects on the environment that lead to a particular known state. However, this assumption is typically dropped in real-world applications, and among LfD planning approaches several algorithms do explore the generalization of demonstrated sequences.

9.2.1.2. Acquisition of New Demonstrations. A fundamentally different approach to dealing with undemonstrated state is to acquire additional demonstrations when novel states are encountered by the robot. Note that such an approach is not equivalent to providing all of these demonstrations *prior* to learner execution, as that requires the teacher to know beforehand every state the learner might encounter during execution, a generally impossible expectation in real world domains. One set of techniques has the teacher decide when to provide new examples, based on learner performance. Another set has the learner re-engage the teacher to request additional demonstrations. Under such a paradigm, the responsibility of selecting states for demonstration now is shared between the robot and teacher.

9.2.2. Poor Quality Data

The quality of a learned LfD policy depends heavily on the quality of the provided demonstrations. In general, approaches assume the dataset to contain high quality demonstrations performed by an expert. In reality, however, teacher demonstrations may be ambiguous, unsuccessful or suboptimal in certain areas of the state space.

9.2.2.1. Suboptimal or Ambiguous Demonstrations. Inefficient demonstrations may be dealt with by eliminating parts of the teacher's execution that are suboptimal. Suboptimality is most common in demonstrations of low-level tasks, for example movement trajectories for robotic arms. In these domains, demonstrations serve as guidance instead of offering a complete solution.

Other approaches smooth or generalize over suboptimal demonstrations in such a way as to improve upon the teacher's performance.

Dataset ambiguity may be caused by uncaptured elements of the record mapping. Additionally, dataset ambiguity may be caused by teacher demonstrations that are inconsistent between multiple executions. This type of ambiguity can arise due to the presence of multiple equally applicable actions, and the result is that a single state maps to different actions. In the case of discrete actions, this results in inconsistent learner behavior. In the case of continuous actions, depending on the regression technique, this can result in an averaged behavior that is dangerously dissimilar to either demonstrated action.

9.2.2.2. Learning from Experience. An altogether different approach to dealing with poor quality data is for the student to learn from experience. If the learner is provided with feedback that evaluates its performance, this may be used to update its policy. This evaluation is generally provided via teacher feedback, or through a reward as in RL.

Learning from experience under the mapping function approximation approach is relatively uncommon. If present, however, performance evaluations can be used to modify the state-action mapping function. This occurs either by modifying the contents of the dataset, or modifying the manner of function approximation. Performance evaluations are generally provided automatically by the world or through teacher feedback. Beyond providing just an evaluation of performance, a human teacher may also provide a *correction* on the executed behavior. For the most part, approaches that employ teacher corrections do so within discrete-action domains.

Most use of RL-type rewards occurs under the system model approach. To emphasize, these are rewards seen during *learner* execution with its policy, and not during teacher demonstration executions. State rewards are typically used to update state values $V(s)$ (or Q-values $Q(s,a)$). Within a system model LfD formulation, student execution information may also be used to update a learned dynamics model $T(s'|s,a)$.

Important to underline is the advantage of providing demonstrations and employing LfD *before* using traditional policy learning from experience techniques. Many traditional learning approaches, such as exploration-based methods like RL, are not immediately applicable to robots. This is due largely to the constraints of gathering experience on a real system. To physically execute *all* actions from *every* state will likely be infeasible, may be dangerous, and will not scale with continuous state spaces. However, the use of these techniques to evaluate and improve upon the experiences gained through an LfD system has proven very successful.

9.3. Addressing Dataset Limitations

The quality of a policy learned from demonstration is tightly linked to the quality of the demonstration dataset. Many LfD learning systems therefore have been augmented to enable learner performance to improve beyond what was provided in the demonstration dataset. One technique for addressing dataset limitations, discussed in Section 9.3.1, is to provide the learner with more teacher demonstrations. The intent here is to populate undemonstrated state-space areas, or to clarify ambiguous executions.

There are LfD limitations, however, that are *not* able to be addressed through demonstration, for example poor teacher-to-learner correspondence or suboptimal teacher ability. Learning from experience is one approach for addressing poor policy performance that does not depend on teacher demonstration. Approaches that extend LfD to have the robot update its policy based on its execution experience are presented in Sections 9.3.2 and 9.3.3. We note that learning from experience is a rich research area outside of the realm of LfD and a variety of techniques exist, the most popular of which is RL. We focus this presentation, however, to applications of experience learning techniques within LfD particularly.

9.3.1. More Demonstrations

Providing the learner with more teacher demonstrations can address the limitation of undemonstrated or ambiguously demonstrated areas of the state-space. This technique is most popular amongst approaches that derive policies by approximating the underlying state→action mapping function. Note that under the mapping function approach, to a certain extent the issue of undemonstrated state is already addressed through the state generalization of the classification or regression policy derivation techniques. State generalization is automatically performed by the statistical techniques used to derive the policy, ranging from the simplest which group unseen states to the nearest dataset state (e.g. *k-NN*) to more complex approaches such as state mixing. In undemonstrated state-space areas, however, this generalization may not be correct.

One set of approaches acquire new demonstrations by enabling the learner to evaluate its confidence in selecting a particular action, based on the confidence of the underlying classification algorithm. In Chernova and Veloso (2007, 2008a) the robot requests additional demonstration in states that are either very different from previously demonstrated states or in states for which a single action can not be selected with certainty. In Grollman and Jenkins (2007) the robot indicates to the teacher its certainty in performing various elements of the task, and the teacher may choose to provide additional demonstrations indicating the correct behavior to perform. More demonstration is provided through kinesthetic teaching, where passive humanoid joints are moved by the teacher through the desired motions, at the teacher's discretion in Calinon and Billard (2007).

9.3.2. Rewarding Executions

Providing the learner with an evaluation of policy performance is another approach to addressing dataset limitations. By incorporating this evaluation into a policy update, performance may improve.

Policy evaluations are most commonly provided through state rewards. This technique is therefore popular amongst policies formulated under the system model approach, which are typically derived using RL techniques. Note that under this policy derivation formulation, state generalization does not occur. Since the goal of RL is to maximize cumulative reward over time, by updating the state values with rewards received, a policy derived under this paradigm also updates. We emphasize that these are rewards seen during *learner* execution with its policy, and not during teacher demonstration executions. Note that reward functions tend to be sparse and thus credit performance only in key states, such as those near a goal or a point of failure. Earlier performance that *leads* to poor performance is therefore not always credited.

Smart (2002) seeds the robot's reward and transition functions with a small number of suboptimal demonstrations, and then optimizes the policy through exploration and RL. A similar approach is taken by Peters and Schaal (2008) in their use of the Natural Actor Critic algorithm, a variant of RL, to learn a motor primitives for a 7-DoF robotic arm. In Stolle and Atkeson (2007), the world model is seeded with trajectories produced by a planner operating on a simulated version of their robot, and learner execution results in state values being updated. In addition to updating state values, student execution information may be used to update a learned dynamics model $T(s'|s,a)$. This approach is taken by Abbeel and Ng (2005), who seed their world model with teacher demonstrations. Policy updates then consist of rederiving the world dynamics after adding learner executions with this policy (and later updated policies) to the demonstration set.

In Bentivegna (2004), a policy *derived* under the mapping function approximation approach is *updated* using RL techniques. The function approximation takes into consideration Q-values associated with paired query-demonstration points, and these Q-values are updated based on learner executions and a developer-defined reward function. Another atypical use of reward is the work of Jansen and Belpaeme (2006), where a student uses binary policy evaluations from a human teacher to prune the set of inferred goals that it maintains within a planning framework.

9.3.3. Correcting Executions

An additional approach to addressing LfD limitations is to directly correct poor predictions made during a policy execution. Policy corrections formulated in this manner do not depend on teacher demonstrations. Furthermore, correcting poor predictions provides more focused and detailed improvement information than overall performance evaluations or state rewards.

Policy correction has seen limited attention within LfD however. The selection of a policy correction is sufficiently complex to preclude it being provided with a simple sparse function, as in the case of most state rewards. Approaches that do correct policy predictions therefore provide corrections through human teachers. Furthermore, since the correction likely indicates a preferred state or action, within the existing literature corrections are only provided within action spaces where the actions are discrete and of significant time duration, and therefore sampled relatively infrequently. The correct action from a discrete set is provided by a human teacher in Nicolescu and Matarić (2003), and this information updates the structure of a hierarchical Neural Network of robot behaviors. Discrete action corrections from a human teacher update an action classifier in Chernova and Veloso (2008b).

9.3.4. Other Approaches

There are other techniques that address LfD limitations but do not fall within any of the broader technique categories already discussed. We first present approaches that consider suboptimal or ambiguous demonstrations, followed by those that consider undemonstrated state-space areas.

Suboptimal demonstrations are addressed in Kaiser et al. (1995), who present a method for actively identifying and removing elements of the demonstration that are unnecessary or inefficient, by removing actions that do not contribute to the solution to the task. Other approaches smooth or

generalize over suboptimal demonstrations in such a way as to improve upon the teacher's performance. For example, when learning motion control, data from multiple repeated demonstrations is used to obtain a smoothed optimal path (Aleotti and Caselli, 2006; Delson and West, 1996; Yeasin and Chaudhuri, 2000), and data from multiple teachers encourages more robust generalization (Pook and Ballard, 1993).

Ambiguous demonstrations are addressed in Friedrich and Dillmann (1995), where the teacher specifies the level of generality that was implied by a demonstration, by answering the learner's questions. Other approaches focus on the development of generalized plans with flexible action order (van Lent and Laird, 2001; Nicolescu and Matarić, 2003). Differences in teacher and learner perspectives during externally recorded demonstrations are accounted for by having the robot maintain separate belief estimates for itself and the human demonstrator (Breazeal et al., 2006). Teacher demonstrations that are inconsistent between multiple executions may be handled by modeling equivalent actions choices explicitly in the robot policy (Chernova and Veloso, 2008b).

To visit and evaluate new states not seen in the demonstration set, an exploration policy may be employed, though we note that taking exploratory steps on a real robot can be inefficient and even dangerous. Executing with an exploration policy may provide information about the world dynamics, and additionally the value of state-action pairs if it is coupled within an approach that evaluates policy performance, for example RL. A safe exploration policy is employed by Smart (2002) for the purposes of gathering robot demonstration data, both to seed and then update a state transition model and reward function. To guide exploration within this work, initially a suboptimal controller is provided, and once a policy is learned small amounts of Gaussian noise are randomly added to the greedy actions of this policy. Execution experience also updates a learned transition function without taking explicit exploration steps (Abbeel and Ng, 2005).

9.4. Summary

In summary, Learning from Demonstration is an effective policy development approach that has been validated on a variety of robotic applications. Algorithms that use demonstration to teach low level motion control tasks for *mobile* robots are less common, but do exist.

Given that there are limitations within the LfD paradigm, approaches that provide techniques to improve beyond the demonstration dataset are noted to be particularly valuable. Popular techniques either provide more demonstration data or state rewards. More demonstration, however, is not able to address all sources of LfD limitations and requires revisiting states, while state rewards are often unable to credit early performance that leads to later poor performance and furthermore give no indication of the preferred policy behavior. In this thesis, we identify techniques that address these additional causes of LfD limitations, and provide more focused informative performance information, to be especially useful.

In particular, the approach of correcting policy predictions is identified to be a promising technique that *does* address both of these concerns. Challenges to providing policy corrections, however, have prevented its inclusion the majority of LfD applications that address dataset limitations. Furthermore, approaches that correct policies within *continuous* action-spaces sampled at high frequency are entirely absent from the literature. These considerations have prompted our

development of multiple feedback forms, in particular of our corrective advice-operator technique, as well as our F3MRP feedback framework that is suitable for rapidly sampled policy domains.

We also identify techniques that build and incorporate primitive policies within LfD frameworks to be useful, given the potential for policy reuse. Only one existing approach however (Bentivegna, 2004) combines a primitive policy framework with techniques that also address dataset limitations. Additionally, the incorporation of data from multiple sources is identified as an interesting and relevant area that has received limited attention within the current LfD literature. Algorithms contributed in this thesis, FPS and DWL respectively, address each of these topics.

CHAPTER 10

Conclusions

THIS thesis has contributed techniques for the development of motion control algorithms for mobile robots. Our introduced approaches use human teacher feedback to build and refine policies learned from demonstration. The most distinguishing characteristics of our feedback techniques are the capability of providing policy corrections within continuous state-action domains, without needing to revisit the state being corrected, and to provide feedback on multiple execution points at once. With these characteristics, our techniques are appropriate for the improvement of low-level motion control policies executed by mobile robots. We use feedback to refine policies learned from demonstration, as well as to build new policies able to accomplish complex motion tasks. In particular, our scaffolding technique allows for the building of complex behaviors from simpler motions learned from demonstration.

Algorithms were introduced that use feedback to build and refine motion control policies for mobile robots. In particular, the algorithms *Binary Critiquing* and *Advice-Operator Policy Improvement* used binary performance flags and corrective advice as feedback, respectively, to refine motion control policies learned from demonstration. The algorithm *Feedback for Policy Scaffolding* used multiple types of feedback to refine primitive motion policies learned from demonstration, and to scaffold them into complex behaviors. The algorithm *Demonstration Weight Learning* treated different feedback types as distinct data sources, and through a performance-based weighting scheme combined data sources into a single policy able to accomplish a complex task.

A variety of techniques were developed to enable the transfer of teacher evaluations of learner execution performance into policy updates. Of particular note is the *advice-operator* formulation, which was a mechanism for correcting motion control policies that exist in continuous action spaces. The *Focused Feedback for Mobile Robot Policies* framework furthermore enabled the transfer of teacher feedback to the learner for policies that are sampled at a high frequency.

All contributed feedback mechanisms and algorithms were validated empirically, either on a Segway RMP robot or within simulated motion control domains. In depth discussions of the conclusions drawn from each algorithm implementation, and the identification of directions for future research, were presented throughout the document in each respective algorithm chapter. In the following sections, the contributions of this thesis are highlighted and summarized.

10.1. Feedback Techniques

The core element of this thesis is the use of teacher feedback to build and refine motion control policies for mobile robots. Towards this end, multiple feedback forms were developed and employed in this work. The most noteworthy of these was the advice-operator formulation, that allowed for the correction of low-level motion control policies. A framework was also contributed, Focused Feedback for Mobile Robot Policies, through which feedback was provided on motion control executions and incorporated into policy updates. The combination of these techniques enabled the correction of policies within (i) continuous state-action spaces and (ii) with actions of short duration. Both of these topics were previously unaddressed within the LfD literature.

10.1.1. Focused Feedback for Mobile Robot Policies

With the Focused Feedback for Mobile Robot Policies (F3MRP) framework, we contributed a technique that enables feedback from a teacher to be provided to the learner and incorporated into a policy update. Key is the application of a single piece of feedback to *multiple* execution points, which is necessary for providing focused feedback within rapidly sampled policy domains. In such domains behavior spans over multiple execution points, and so to select each point individually would be tedious and inefficient. The application of feedback to multiple points is accomplished through a visual display of the mobile robot execution ground path and the segment selection technique of F3MRP.

To associate feedback with the execution, the teacher selects segments of a graphically represented execution trace. The feedback interface provides the human teacher with an indication of dataset support, which the teacher uses when selecting execution segments to receive good performance flags. In this work, a discussion of the details involved in developing a feedback framework was also provided. Key design decisions were identified as: the type of feedback provided, the feedback interface and the manner of feedback incorporation into the policy.

10.1.2. Advice-Operators

Advice-operators were contributed as a mechanism for correcting low-level motion control policies. Paired with the segment selection technique of the F3MRP framework, the advice-operator formulation allows a *single* piece of advice to provide *continuous*-valued corrections on *multiple* execution points. Advice-operators are thus appropriate for correcting continuous-valued actions sampled at high frequency, as in low-level motion control domains.

Advice-operators function as mathematical computations on the data points resulting from a learner execution. They synthesize new advice-modified data points, which are added to the demonstration dataset and consequently update a policy derived from this set. There are many key advantages to using advice to correct a policy. One advantage is that a state does not need to be revisited to receive a correction, which is particularly useful when revisiting states would involve executing physical actions on a mobile robot. Another advantage is that actions produced by advice-operators are not restricted to those already present in the dataset, and thus are not limited by suboptimal teacher performance or poor teacher-to-learner correspondence.

The success of the advice-operator technique depends on the definition of effective operators. To this end, we furthermore have contributed a principled approach for the development of action advice-operators. Under this paradigm, a baseline set of operators are automatically defined from the available robot actions. A developer may then build new operators, by composing and sequencing existing operators, starting with the baseline set. The new data points produced from advice are additionally constrained by parameters automatically set based on the robot dynamics, to firmly ground the synthesized data on a real execution and the physical limits of the robot. Directions for future work with advice-operators were identified, and included their application to more complex action spaces and the further development of observation-modifying operators.

10.2. Algorithms and Empirical Results

This thesis contributed multiple algorithms that build motion control policies for mobile robots through a combination of demonstration and teacher feedback. Each algorithm was empirically validated, either on a Segway RMP robot or within a simulated motion control domain. A summary of these algorithms, and their empirical results, are presented here.

10.2.1. Binary Critiquing

The Binary Critiquing (BC) algorithm provided an approach for crediting policy performance with teacher feedback in the form of negative performance flags. The algorithm was implemented within a simulated robot motion interception domain. Empirical results showed improvements in success and efficiency as a consequence of the negative credit feedback. Interestingly, the performance of the final learned policy exceeded the performance of the demonstration policy, without modifying the contents of the demonstration dataset. Instead, teacher feedback modified the use of the existing demonstration data by the regression techniques employed for policy derivation.

This work also provided a discussion of the benefits of using a human for credit assignment. In particular, the human selected portions of an execution to flag as poorly performing. Rather than just crediting poor performance where it occurs, segment selection allows for behaviors that *lead* to poor performance to receive poor credit as well. Future directions for the BC algorithm included directional critiques that consider query point orientation and real-valued critiques that impart a measure of relative, rather than just binary, performance. Feedback that is corrective was also identified as a potential improvement, motivating the development of advice-operators.

10.2.2. Advice-Operator Policy Improvement

With the Advice-Operator Policy Improvement (A-OPI) algorithm, an approach was provided that improves policy performance through corrective teacher feedback. A noteworthy contribution of this work is that corrections were efficiently offered in continuous state-action domains sampled at high frequency, through the use of advice-operators.

The algorithm was implemented on a Segway RMP robot performing planar motion tasks. The initial case study implementation used a simple set of advice-operators to correct a spatial trajectory following task. Empirical results showed improvements in precision and efficiency, and both beyond the abilities of demonstrator. Corrective advice was also found to enable the emergence of novel

behavior characteristics, absent from the demonstration set. The full task implementation used a complex set of advice-operators to correct a more difficult spatial positioning task. Empirical results showed performance improvement, measured by success and accuracy. Furthermore, the A-OPI policies displayed similar or superior performance when compared to a policy developed from exclusively more teleoperation demonstrations. The A-OPI approach also produced noticeably smaller datasets, without a sacrifice in policy performance, suggesting that the datasets were more focused and contained smaller amounts of redundant data.

Also presented was discussion of the features of the A-OPI approach that facilitate providing corrections within low-level motion control domains. In particular, the segment selection technique allows multiple actions to be corrected by a single piece of advice. The advice-operator mechanism translates high-level advice into continuous-valued corrections. Selection of an operator is reasonable for a human to perform, since the set of advice-operators is discrete; by contrast, selection from an infinite set of continuous-values corrections would not be reasonable. Future directions for the A-OPI algorithm were identified to include interleaving the policy improvement techniques of corrective advice and more demonstrations, as well as additional advice formulations.

10.2.3. Feedback for Policy Scaffolding

The Feedback for Policy Scaffolding (FPS) algorithm contributed an approach that uses feedback to build complex policy behaviors. More specifically, the algorithm operates in two phases. The first phase develops primitive motion behaviors from demonstration and feedback provided through the F3MRP framework. The second phase scaffolds these primitive behaviors into a policy able to perform a more complex task. F3MRP feedback is again employed, in this case to assist the scaffolding and thus enable the development of a sophisticated motion behavior.

The FPS algorithm was implemented within a simulated motion domain that tasked a differential drive robot with driving a racetrack. Primitive behavior policies, which represent simple motion components of this task, were successfully learned through demonstration and teacher feedback. A policy able to drive the full track also was successfully developed, from the primitive behaviors and teacher feedback. In both phases, empirical results showed policy performance to improve with teacher feedback. Furthermore, all FPS policies outperformed the comparative policies developed from teacher demonstrations alone. While these exclusively demonstrated policies were able to produce the primitive behaviors, albeit less successfully than the FPS policies, they were not able to perform the more complex task behavior.

Within this work we also discussed the advantages of reusing primitive policies learned from demonstration. The benefit of not needing to revisit states to provide corrections was also highlighted, as the correction techniques of the F3MRP framework were much more efficient than demonstration at developing the desired robot behaviors. Directions for future research were identified as alternate ways to select between and scaffold primitive behavior policies, as well as other approaches for feedback incorporation into the complex policy.

10.2.4. Demonstration Weight Learning

With the Demonstration Weight Learning (DWL) algorithm, an approach was introduced that treats different types of feedback as distinct data sources and, through a weighting scheme, incorporates them into a policy able to perform a complex task. Data sources are selected through an expert learning-inspired paradigm, and their weights automatically determined based on execution performance. This paradigm thus allows for the evaluation of multiple feedback sources, and furthermore of various feedback types. An additional motivation for this algorithm was our expectation of multiple demonstration teachers within real world applications of LfD.

The DWL algorithm was implemented within a simulated racetrack driving domain. Empirical results confirmed an unequal reliability between data sources, and source weighting was found to improve policy performance. Furthermore, the source weights learned under the DWL paradigm were consistent with the respective performance abilities of each individual expert. A discussion of the differing reliability and weighting of data sources was presented. Future research directions identified were the development of alternate techniques for weighting data sources, and the application of DWL to complex domains requiring multiple demonstration teachers with varied skill sets.

10.3. Contributions

The aim of this thesis was to address two research questions. The first question, *How might teacher feedback be used to address and correct common Learning from Demonstration limitations in low-level motion control policies?*, has been addressed through the following contributions:

- The introduction of the *Focused Feedback for Mobile Robot Policies* framework as a mechanism for providing focused feedback on multiple execution points, used for a policy update.
- The development of *advice-operators* as a technique for providing corrections within continuous state-action spaces, in addition to the development of other *novel feedback forms*.
- The introduction of the *Binary Critiquing* and *Advice-Operator Policy Improvement* algorithms, that refine motion control policies learned from demonstration through the incorporation these novel feedback forms.

The second question, *In what ways might the resulting feedback techniques be incorporated into more complex policy behaviors?* has been addressed by these contributions:

- The introduction of the *Feedback for Policy Scaffolding* algorithm, that uses multiple feedback types both to refine motion behavior primitives learned from demonstration, and to scaffold them into a policy able to accomplish a more complex behavior.
- The introduction of the *Demonstration Weight Learning* algorithm, that combines multiple demonstration and feedback sources into a single policy through a performance-based weighting scheme, that consequently evaluates the differing reliability between the various demonstration teachers and feedback types, using our contributed feedback framework.

Furthermore, all contributed algorithms and techniques have been empirically validated, either on a Segway RMP robot platform or within simulated motion control domains.

In conclusion, this thesis has contributed multiple algorithms that build and refine low-level motion control policies for mobile robots. The algorithms develop policies through a combination of demonstration and focused, at times corrective, teacher feedback. Novel feedback forms were introduced, along with a framework through which policy evaluations by a human teacher are transformed into focused feedback usable by the robot learner for a policy update. Through empirical validations, feedback was shown to *improve* policy performance on simple behaviors, and *enable* policy execution of more complex behaviors. All contributed algorithms and feedback techniques are appropriate for continuous-valued action domains sampled at high frequency, and thus for low-level motion control on mobile robots.

BIBLIOGRAPHY

P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21st International Conference on Machine Learning (ICML '04)*, 2004.

P. Abbeel and A. Y. Ng. Exploration and apprenticeship learning in reinforcement learning. In *Proceedings of the 22nd International Conference on Machine Learning (ICML '05)*, 2005.

P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Proceedings of Advances in Neural Information Proccessing (NIPS '07)*, 2007.

J. Aleotti and S. Caselli. Robust trajectory learning and approximation for robot programming by demonstration. *Robotics and Autonomous Systems, Special Issue on The Social Mechanisms of Robot Programming by Demonstration*, 54(5):409–413, 2006.

R. Aler, O. Garcia, and J. M. Valls. Correcting and improving imitation models of humans for robosoccer agents. *Evolutionary Computation*, 3(2-5):2402–2409, 2005.

A. Alissandrakis, C. L. Nehaniv, and K. Dautenhahn. Towards robot cultures? *Interaction Studies: Social Behaviour and Communication in Biological and Artificial Systems*, 5(1):3–44, 2004.

R. Amit and M. J. Matarić. Learning movement sequences from demonstration. In *Proceedings of the 2nd International Conference on Development and Learning (ICDL '02)*, 2002.

B. Argall, Y. Gu, B. Browning, and M. Veloso. The first segway soccer experience: Towards peer-to-peer human-robot teams. In *First Annual Conference on Human-Robot Interactions (HRI '06)*, 2006.

B. Argall, B. Browning, and M. Veloso. Learning from demonstration with the critique of a human teacher. In *Second Annual Conference on Human-Robot Interactions (HRI '07)*, 2007a.

B. Argall, B. Browning, and M. Veloso. Learning to select state machines on an autonomous robot using expert advice. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '07)*, 2007b.

B. Argall, B. Browning, and M. Veloso. Learning robot motion control with demonstration and advice-operators. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '08)*, 2008.

B. Argall, B. Browning, and M. Veloso. Automatic weight learning for multiple data sources when learning from demonstration. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '09)*, 2009a.

B. Argall, S. Chernova, B. Browning, and M. Velosos. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009b.

C. G. Atkeson and S. Schaal. Robot learning from demonstration. In *Proceedings of the Fourteenth International Conference on Machine Learning (ICML '97)*, 1997.

C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11:11–73, 1997.

P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. Gambling in a rigged casino: The adversarial multiarm bandit problem. In *36th Annual Symposium on Foundations of Computer Science*, 1995.

J. A. Bagnell and J. G. Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '01)*, 2001.

D. C. Bentivegna. *Learning from Observation Using Primitives*. PhD thesis, College of Computing, Georgia Institute of Technology, Atlanta, GA, 2004.

A. Billard and M. J. Matarić. Learning human arm movements by imitation: Evaluation of biologically inspired connectionist architecture. *Robotics and Autonomous Systems*, 37(2-3):145–160, 2001.

A. Billard, S. Calinon, and F. Guenter. Discriminative and adaptive imitation in uni-manual and bi-manual tasks. *Robotics and Autonomous Systems, Special Issue on The Social Mechanisms of Robot Programming by Demonstration*, 54(5):370–384, 2006.

M. Bowling and M. Veloso. Convergence of gradient dynamics with a variable learning rate. In *Proceedings of the 18th International Conference on Machine Learning (ICML '01)*, 2001.

C. Breazeal, M. Berlin, A. Brooks, J. Gray, and A. L. Thomaz. Using perspective taking to learn from ambiguous demonstrations. *Robotics and Autonomous Systems, Special Issue on The Social Mechanisms of Robot Programming by Demonstration*, 54(5):385–393, 2006.

B. Browning, L. Xu, and M. Veloso. Skill acquisition and use for a dynamically-balancing soccer robot. In *Proceedings of 19th National Conference on Artificial Intelligence (AAAI '04)*, 2004.

S. Calinon and A. Billard. Incremental learning of gestures by imitation in a humanoid robot. In *Proceedings of the 2nd ACM/IEEE International Conference on Human-Robot Interactions (HRI '07)*, 2007.

A. Chella, H. Dindo, and I. Infantino. A cognitive framework for imitation learning. *Robotics and Autonomous Systems, Special Issue on The Social Mechanisms of Robot Programming by Demonstration*, 54(5):403–408, 2006.

J. Chen and A. Zelinsky. Programing by demonstration: Coping with suboptimal teaching actions. *The International Journal of Robotics Research*, 22(5):299–319, 2003.

S. Chernova and M. Veloso. Confidence-based learning from demonstration using Gaussian Mixture Models. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS '07)*, 2007.

S. Chernova and M. Veloso. Multi-thresholded approach to demonstration selection for interactive robot learning. In *Proceedings of the 3rd ACM/IEEE International Conference on Human-Robot Interaction (HRI '08)*, 2008a.

S. Chernova and M. Veloso. Learning equivalent action choices from demonstration. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '08)*, 2008b.

W. S. Cleveland and C. Loader. Smoothing by local regression: Principles and methods. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, 1995.

J. A. Clouse. *On integrating apprentice learning and reinforcement learning.* PhD thesis, University of Massachusetts, Department of Computer Science, 1996.

K. Dautenhahn and C. L. Nehaniv, editors. *Imitation in animals and artifacts.* MIT Press, Cambridge, MA, USA, 2002.

N. Delson and H. West. Robot programming by human demonstration: Adaptation and inconsistency in constrained motion. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '96)*, 1996.

Y. Demiris and G. Hayes. Imitation as a dual-route process featuring predictive and learning components: a biologically-plausible computational model. In Dautenhahn and Nehaniv (2002), chapter 13.

Y. Demiris and B. Khadhouri. Hierarchical attentive multiple models for execution and recognition of actions. *Robotics and Autonomous Systems, Special Issue on The Social Mechanisms of Robot Programming by Demonstration*, 54(5):361–369, 2006.

M. B. Dias, B. Kannan, B. Browning, E. G. Jones, B. Argall, M. F. Dias, M. B. Zinck, M. Veloso, and A. Stentz. Sliding autonomy for peer-to-peer robot teams. In *Proceedings of the 10th International Conference on Intelligent Autonomous Systems (IAS '08)*, 2008.

R. Dillmann, M. Kaiser, and A. Ude. Acquisition of elementary robot skills from human demonstration. In *International Symposium on Intelligent Robotic Systems (SIRS '95)*, 1995.

J. W. Eaton. *GNU Octave Manual.* Network Theory Limited, 2002. ISBN 0-9541617-2-6.

H. Friedrich and R. Dillmann. Robot programming based on a single demonstration and user intentions. In *3rd European Workshop on Learning Robots at ECML '95*, 1995.

D. H. Grollman and O. C. Jenkins. Dogged learning for robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '07)*, 2007.

D. H. Grollman and O. C. Jenkins. Sparse incremental learning for interactive robot control policy estimation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '08)*, 2008.

F. Guenter and A. Billard. Using reinforcement learning to adapt an imitation task. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '07)*, 2007.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, New York, NY, USA, 2001.

G. Hovland, P. Sikka, and B. McCarragher. Skill acquisition from human demonstration using a hidden markov model. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '96)*, 1996.

A. J. Ijspeert, J. Nakanishi, and S. Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '02)*, 2002a.

A. J. Ijspeert, J. Nakanishi, and S. Schaal. Learning rhythmic movements by demonstration using nonlinear oscillators. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '02)*, 2002b.

T. Inamura, M. Inaba, and H. Inoue. Acquisition of probabilistic behavior decision model based on the interactive teaching method. In *Proceedings of the Ninth International Conference on Advanced Robotics (ICAR '99)*, 1999.

I. Infantino, A. Chella, H. Dzindo, and I. Macaluso. A posture sequence learning system for an anthropomorphic robotic hand. *Robotics and Autonomous Systems*, 42:143–152, 2004.

B. Jansen and T. Belpaeme. A computational model of intention reading in imitation. *Robotics and Autonomous Systems, Special Issue on The Social Mechanisms of Robot Programming by Demonstration*, 54(5):394–402, 2006.

O. C. Jenkins, M. J. Matarić, and S. Weber. Primitive-based movement classification for humanoid imitation. In *Proceedings of the 1st IEEE-RAS International Conference on Humanoid Robotics*, 2000.

E. G. Jones, B. Browning, M. B. Dias, B. Argall, M. Veloso, and A. Stentz. Dynamically formed heterogeneous robot teams performing tightly-coordinated tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '06)*, 2006.

M. Kaiser, H. Friedrich, and R. Dillmann. Obtaining good performance from a bad teacher. In *Workshop on Programming by Demonstration at ICML '95*, 1995.

J. Z. Kolter, P. Abbeel, and A. Y. Ng. Hierarchical apprenticeship learning with application to quadruped locomotion. In *Proceedings of Advances in Neural Information Proccessing (NIPS '08)*, 2008.

Y. Kuniyoshi, M. Inaba, and H. Inoue. Learning by watching: Extracting reusable task knowledge from visual observation of human performance. In *IEEE Transactions on Robotics and Automation*, volume 10, pages 799–822, 1994.

S. Lauria, G. Bugmann, T. Kyriacou, and E. Klein. Mobile robot programming using natural language. In *Robotics and Autonomous Systems*, volume 38, pages 171–181, 2002.

J. Lieberman and C. Breazeal. Improvements on action parsing and action interpolation for learning through demonstration. In *4th IEEE/RAS International Conference on Humanoid Robots*, 2004.

A. Lockerd and C. Breazeal. Tutelage and socially guided robot learning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '04)*, 2004.

M. Lopes and J. Santos-Victor. Visual learning by imitation with motor representations. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 35(3):438–449, 2005.

M. J. Matarić. Sensory-motor primitives as a basis for learning by imitation: Linking perception to action and biology to robotics. In Dautenhahn and Nehaniv (2002), chapter 15.

J. Nakanishi, J. Morimoto, G. Endo, G. Cheng, S. Schaal, and M. Kawato. Learning from demonstration and adaptation of biped locomotion. *Robotics and Autonomous Systems*, 47:79–91, 2004.

C. L. Nehaniv and K. Dautenhahn. The correspondence problem. In Dautenhahn and Nehaniv (2002), chapter 2.

U. Nehmzow, O. Akanyeti, C. Weinrich, T. Kyriacou, and S. Billings. Robot programming by demonstration through system identification. In *Proceedings of the IEEE/RSJ International Conference*

on Intelligent Robots and Systems (IROS '07), 2007.

A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang. Inverted autonomous helicopter flight via reinforcement learning. In *International Symposium on Experimental Robotics*, 2004.

H. G. Nguyen, J. Morrell, K. Mullens, A. Burmeister, S. Miles, K. Thomas, and D. W. Gage. Segway robotic mobility platform. In *SPIE Mobile Robots XVII*, 2004.

M. N. Nicolescu and M. J. Matarić. Experience-based representation construction: learning from human and robot teachers. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '01)*, 2001a.

M. N. Nicolescu and M. J. Matarić. Learning and interacting in human-robot domains. *Special Issue of IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, 31(5): 419–430, 2001b.

M. N. Nicolescu and M. J. Matarić. Methods for robot task learning: Demonstrations, generalization and practice. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS '03)*, 2003.

M. Ogino, H. Toichi, Y. Yoshikawa, and M. Asada. Interaction rule learning with a human partner based on an imitation faculty with a simple visuo-motor mapping. *Robotics and Autonomous Systems, Special Issue on The Social Mechanisms of Robot Programming by Demonstration*, 54 (5):414–418, 2006.

E. Oliveira and L. Nunes. *Learning by exchanging Advice*. Springer, 2004.

M. Ollis, W. H. Huang, and M. Happold. A bayesian approach to imitation learning for robot navigation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '07)*, 2007.

J. Peters and S. Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190, 2008.

N. Pollard and J. K. Hodgins. Generalizing demonstrated manipulation tasks. In *Workshop on the Algorithmic Foundations of Robotics (WAFR '02)*, 2002.

D. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

P. K. Pook and D. H. Ballard. Recognizing teleoperated manipulations. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '93)*, 1993.

B. Price and C. Boutilier. Accelerating reinforcement learning through implicit imitation. *Journal of Artificial Intelligence Research*, 19:569–629, 2003.

R. P. N. Rao, A. P. Shon, and A. N. Meltzoff. A bayesian model of imitation in infants and robots. In K. Dautenhahn and C. Nehaniv, editors, *Imitation and Social Learning in Robots, Humans, and Animals: Behavioural, Social and Communicative Dimensions*, chapter 11. Cambridge University Press, Cambridge, UK, 2004.

N. Ratliff, J. A. Bagnell, and M. A. Zinkevich. Maximum margin planning. In *Proceedings of the 23rd International Conference on Machine Learning (ICML '06)*, 2006.

N. Ratliff, D. Bradley, J. A. Bagnell, and J. Chestnutt. Boosting structured prediction for imitation learning. *Proceedings of Advances in Neural Information Processing Systems (NIPS '07)*, 2007.

H. Robbins. Some aspects of the sequential design of experiments. *Bulletin American Mathematical Society*, 55:527–535, 1952.

M. T. Rosenstein and A. G. Barto. Supervised actor-critic reinforcement learning. In J. Si, A. Barto, W. Powell, and D. Wunsch, editors, *Learning and Approximate Dynamic Programming: Scaling Up to the Real World*, chapter 14. John Wiley & Sons, Inc., New York, NY, USA, 2004.

S. Russell. Learning agents for uncertain environments (extended abstract). In *Proceedings of the eleventh annual conference on Computational learning theory (COLT '98)*, 1998.

P. E. Rybski and R. M. Voyles. Interactive task training of a mobile robot through human gesture recognition. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '99)*, 1999.

P. E. Rybski, K. Yoon, J. Stolarz, and M. M. Veloso. Interactive robot task training through dialog and demonstration. In *Proceedings of the 2nd ACM/IEEE International Conference on Human-Robot Interactions (HRI '07)*, 2007.

C. Sammut, S. Hurst, D. Kedzier, and D. Michie. Learning to fly. In *Proceedings of the Ninth International Workshop on Machine Learning*, 1992.

J. Saunders, C. L. Nehaniv, and K. Dautenhahn. Teaching robots by moulding behavior and scaffolding the environment. In *Proceedings of the 1st ACM/IEEE International Conference on Human-Robot Interactions (HRI '06)*, 2006.

S. Schaal and C. G. Atkeson. Constructive incremental learning from only local information. *Neural Computation*, 10(8):2047–2084, 1998.

S. Schaal and D. Sternad. Programmable pattern generators. In *3rd International Conference on Computational Intelligence in Neuroscience*, 1998.

R. Simmons and D. Apfelbaum. A task description language for robot control. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '98)*, 1998.

W. D. Smart. *Making Reinforcement Learning Work on Real Robots*. PhD thesis, Department of Computer Science, Brown University, Providence, RI, 2002.

W. D. Smart and L. P. Kaelbling. Effective reinforcement learning for mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '02)*, 2002.

R. Stefani, B. Shahian, C. Savant, and G. Hostetter. *Design of Feedback Control Systems*. Oxford University Press, 2001.

J. Steil, F. Røthling, R. Haschke, and H. Ritter. Situated robot learning for multi-modal instruction and imitation of grasping. *Robotics and Autonomous Systems, Special Issue on Robot Learning by Demonstration*, 2-3(47):129–141, 2004.

M. Stolle and C. G. Atkeson. Knowledge transfer using local features. In *Proceedings of IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL '07)*, 2007.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, London, England, 1998.

J. D. Sweeney and R. A. Grupen. A model of shared grasp affordances from demonstration. In *Proceedings of the IEEE-RAS International Conference on Humanoids Robots*, 2007.

A. Ude, C. G. Atkeson, and M. Riley. Programming full-body movements for humanoid robots by observation. *Robotics and Autonomous Systems*, 47:93–108, 2004.

M. van Lent and J. E. Laird. Learning procedural knowledge through observation. In *Proceedings of the 1st international conference on Knowledge capture (K-CAP '01)*, 2001.

S. Vijayakumar and S. Schaal. Locally weighted projection regression: An o(n) algorithm for incremental real time learning in high dimensional space. In *Proceedings of 17th International Conference on Machine Learning (ICML '00)*, 2000.

R. M. Voyles and P. K. Khosla. A multi-agent system for programming robots by human demonstration. *Integrated Computer-Aided Engineering*, 8(1):59–67, 2001.

M. Yeasin and S. Chaudhuri. Toward automatic robot programming: Learning human skill from visual data. *IEEE Transactions on Systems, Man and Cybernetics – Part B: Cybernetics*, 30, 2000.

B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of 23rd National Conference on Artificial Intelligence (AAAI '08)*, 2008.

Document Log:

Manuscript Version 3
Typeset by $\mathcal{AMS}$-LaTeX — 21 August 2012
Template from Nicholas Roy.

Brenna D. Argall

Robotics Institute, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213, USA, *Tel.* : (412) 268-9923
E-mail address: bargall@@ri.cmu.edu