Abstract of "Teaching Old Dogs New Tricks: Incremental Multimap Regression for Interactive Robot Learning from Demonstration" by Daniel H Grollman, Ph.D., Brown University, May 2010.

We consider autonomous robots as having associated control policies that determine their actions in response to perceptions of the environment. Often, these controllers are explicitly transferred from a human via programmatic description or physical instantiation. Alternatively, Robot Learning from Demonstration (RLfD) can enable a robot to learn a policy from observing only demonstrations of the task itself. We focus on interactive, teleoperative teaching, where the user manually controls the robot and provides demonstrations while receiving learner feedback. With regression, the collected perception-actuation pairs are used to directly estimate the underlying policy mapping.

This dissertation contributes an RLfD methodology for interactive, mixed-initiative learning of unknown tasks. The goal of the technique is to enable users to implicitly instantiate autonomous robot controllers that perform desired tasks as well as the demonstrator, as measured by task-specific metrics. With standard regression techniques, we show that such "on-par" learning is restricted to policies typified by a many-to-one mapping (a unimap) from perception to actuation. Thus, controllers representable as multi-state Finite State Machines (FSMs) and that exhibit a one-to-many mapping (a multimap) cannot be learnt. To be able to do so we must address the three issues of model selection (how many subtasks or FSM states), policy learning (for each subtask), and transitioning (between subtasks). Previous work in RLfD has assumed knowledge of the task decomposition and learned the subtask policies or the transitions between them in isolation.

We instead address both model selection and policy learning simultaneously. Our presented technique uses an infinite mixture of experts and treats the multimap data from an FSM controller as being generated from overlapping unimaps. The algorithm automatically determines the number of unimap experts (model selection) and learns a unimap for each one (policy learning). On data from both synthetic and robot soccer multimaps we show that the discovered subtasks can be used (switched between) to reperform the original task. While not at the same level of skill as the demonstrator, the resulting approximations represent significant improvement over ones for the same tasks learned with unimap regression.

Teaching Old Dogs New Tricks: Incremental Multimap Regression for Interactive Robot Learning from Demonstration

by
Daniel H Grollman
Sc.M Computer Science, Brown University
B.S. Electrical Engineering and Computer Science, Yale University

A dissertation submitted in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy
in the Department of Computer Science at Brown University

Providence, Rhode Island
May 2010

This dissertation by Daniel H Grollman is accepted in its present form by the Department of Computer Science as satisfying the dissertation requirement for the degree of Doctor of Philosophy.

Date _____ _____

Odest Chadwicke Jenkins, Advisor

Recommended to the Graduate Council

Date _____ _____

Tom Dean, Reader
Google Inc.

Date _____ _____

Manuela Veloso, Reader
Carnegie Mellon University

Date _____ _____

Daniel Lee, Reader
University of Pennsylvania

Approved by the Graduate Council

Date _____ _____

Sheila Bonde
Dean of the Graduate School

Curricula Vita

Daniel H Grollman was born June 28, 1981 in New York, NY. A product of the NYC public school system, he was "magnetized" in fifth grade, and eventually attended The Bronx High School of Science. While in high school he began doing academic, scientific research at Cornell Medical School in 1998, investigating the role of adhesion molecules in inflammation. At Yale University he switched to Electrical Engineering and Computer Science and graduated with a combined B.S. in 2003, after building a robotic rat for his senior project. From there he enrolled in the Brown University Department of Computer Science's PhD program, and obtained his Master's en route in 2005 for work in robotic perception. In addition to these locations, Dan has gained experience working for the University of Edinburgh, Microsoft Corporation, iRobot Corporation, and the Fraunhofer Institute.

While at Brown, Dan's research has led to publications in several conferences, workshops and journals, including the International Conference on Robotics and Automation, the International Conference on Development and Learning, Neural Information Processing Systems, the International Conference on Intelligent Robots and Systems, and the Journal of Field Robotics. His work has led him to be selected as a Young Pioneer in Human-Robot Interaction twice, and garnered a best video and best poster award. Further, he has appeared in the Brown Annual Report, the Brown Alumni Magazine, and on the Cartoon Network.

While a graduate student, Dan has mentored several high school interns, undergraduates and master's students, aiding in their selection and development of various projects. In a classroom setting, Dan has given guest lectures and demonstrations to current and prospective students, both at Brown and other institutions. He has also assisted in the development and teaching of graduate-level courses on Machine Learning and Cryptography. After completing his degree, Dan will be serving as a post-doc at the École Polytechnique Fédérale de Lausanne, continuing research into robot learning from demonstration.

Preface and Acknowledgments

I, literally, could not have completed this dissertation without the help, support, and prior work of many. As an overview of the academic work related to my dissertation is given in Chapter 2, I will take this space to point out the less academic, but no less important, contributions of others.

Firstly, my family, who have been supporting my curiosity from early on, allowing me to disassemble and explore whatever I could get my hands on, and providing books and classes for those things I could not. Schoolteachers as well, too numerous to mention, fanned the flames of inquiry, but I must especially thank Mr. Rockfeld, who let me run wild in the computer lab, and Mrs. Reidy, who threatened to fail me unless I committed a year of my life to research.

Doing that research with Dr. Muller exposed me to rigorous scientific experimentation for the first time. He and the other researchers taught me many of my good lab habits, such as not eating at the workbench. At college, I continued academic work with Drs. Hudak, Peterson, McDermott and Scassellati, who gave me my first taste of robotics. I've since been hooked on using computers to interact with the physical world, and thank them for getting me involved in such an exciting field.

At Yale and Brown both, I have been able to surround myself with like-minded individuals who have fostered my intellectual development, in addition to taking classes that extended my sphere of knowledge outside of my discipline. Particularly, without the members of the Yale Precision Marching Band, The Purple Crayon of Yale, Brown University Gilbert and Sullivan, and the Brown Tae Kwon Do club, I do not think I would be where or who I am today.

Here in graduate school, from the initial days of the Albino Kangaroo Society, my friends and coworkers have been one and the same. My officemates and housemates put up with me daily, and the members of RLAB and Brown # have provided unfailing support, particularly Jesse Butterfield, who came through in a pinch. Special thanks also go to Micah Lapping-Carr and Daniel Byers, who developed the graphical front end for my system, as well as providing some sweet sax playing and videography services. Likewise, my cohorts on the machine learning side were Sharon Goldwater and Frank Wood, to whom nonparametric, Bayesian models are like mother's milk.

The departmental tstaff and astaff have provided exemplary support for my work. Over the years they have assisted me with a wide variety of practical requests, from installing shelves to buying scotch. My committee, particularly my adviser, likewise provided wide-ranging academic support, allowing me to explore my own ideas, while keeping me from going off the deep end. Finally, this work was made possible in part by the NSF (IIS-0534858), Brown Salomon, and readers like you.

Contents

⋆ The use herein of photographs, algorithms, figures and quotations does not constitute support or approval of this work by the copyright holders. This dissertation was almost titled "Rogering the Dog," but eventually good taste won out.

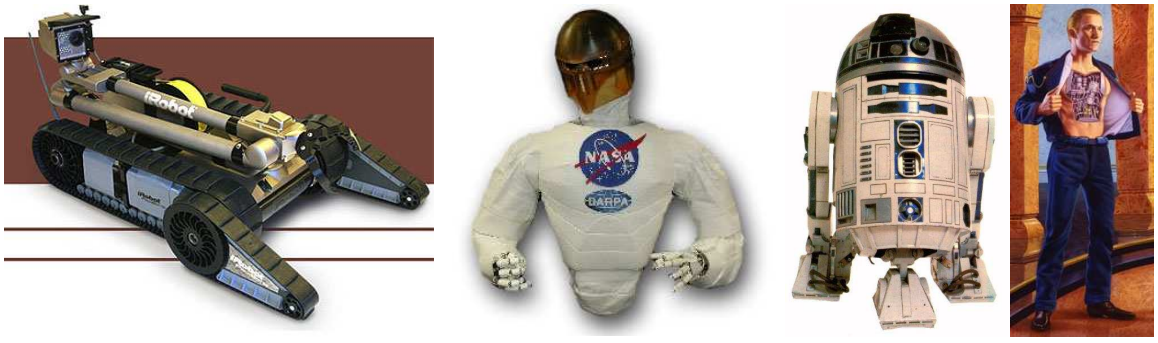
Chapter 1

Introduction

> *In ten years Rossum's Universal Robots will produce so much corn, so much cloth, so much everything, that things will be practically without price. There will be no poverty. All work will be done by living machines. Everbody will be free from worry and liberated from the degradation of labor. Everbody will live only to perfect himself.*
>
> Karel Čapek, R.U.R, 1921, page 15

We take as a goal of robotics the extension of our computing capabilities into the physical world. That is, as modern computers allow non-specialist users to automate the manipulation and management of digital data, robotics aims to enable those same users to exert similar control over their physical environment. Consider, for example, recent video and image editing software, which enable novice users to modify their data in ways that, previously, only experts could. We likewise seek, in this dissertation, to aid these users in controlling their environment, through robots, in ways that currently only skilled roboticists can.

Currently, decades of work in artificial intelligence and robotics has lead to high-end consumer



(a) Packbot (b) Robonaut (c) R2D2 (d) Olivaw

Figure 1.1: A sampling of robots both real (a,b) and fictional (c,d).

and research robots such as those in Figures 1.1a and 1.1b. However, these robots tend to be domain specific, exhibit limited autonomy, and be difficult to adapt to new applications. In other words, while there exist robots that perform complicated and useful tasks, they often perform only in a particular environment, and/or possibly require excessive human supervision or special training to operate. These shortcomings can be traced to issues of robot behavior, as opposed to robot embodiment. The very fact that robots *can* perform these tasks (under supervision) argues for the sufficiency of their forms. What is lacking must then lie in how that form is used, in the perceptual, actuational, and decision making systems that map from raw information extracted from the environment to robot-effected changes. Terming this aspect the robot's control system, we argue that while robots may be physically capable of performing many desired tasks, the autonomous control systems necessary for executing them in varying conditions are difficult to develop, and often lacking. We therefore seek improved techniques for instantiating autonomous robot control policies.

More generally, we may be moving towards so-called "universal" robots: Multi-purpose, robust, adaptable systems that interact with humans in an intuitive manner and perform a wide variety of tasks. Examples of such robots abound in fiction, such as those seen in Figures 1.1c and 1.1d. These robots are able to assist humans in a multitude of settings, receiving input from speech and gestures just as a human would, and acting safely alongside us. They further exhibit decision making and problem solving, often determining the correct sequence of actions to bring about a desired result. In addition to such autonomy, over their (long) lifetimes universal robots exhibit adaptability to new situations and tasks, perhaps by learning new skills or discovering additional uses for old ones.

However, while autonomous control policies themselves are required, it is not so that they must exhibit learning and adaptation. It is conceivable that all the behaviors and autonomy required by the robot over its lifetime can be designed (hardwired or preprogrammed) into the robot, and never require updating or modification. These fixed behaviors would then determine the robot's every move, in response to environmental stimuli. They may also enable robots to adapt to changes in their physical structure, due to damage or willful modification. This approach to robot control can be seen as analogous to innate behaviors or instincts in biology. For some lifeforms, these born-in abilities are all that is necessary for a full life, similar to a single-purpose, preprogrammed robot. For other species, such as ourselves, and for robots that will have to deal with a wide variety of unknown situations, the ability to develop new behaviors may be advantageous.

Even if not necessary for the tasks themselves, learning may greatly simplify the challenge of developing autonomous control policies for robots. We take this view here, and point out two ways in which it may occur. Firstly, by using learning from demonstration, the field of robot control policy instantiation may become accessible to users other than traditional programmers. More varied types of users may lead to new approaches to difficult problems and result in higher quality solutions than those derived by scientists [64]. Secondly, interactive learning, where the user corrects improper behavior and the robot request more demonstrations as needed, may enable the development of controllers that address the particular situations that arise, and perform well enough in most cases. These controllers could then be extended to add support for rarer edge cases when and if they occur.

We thus contribute a framework for performing interactive robot learning from demonstration, or *tutelage*, as referred to herein. The framework is designed to be run over the entire lifetime of the robot, blending autonomous behavior with the learning of new tasks. Within this framework we further contribute a new learning algorithm for multivalued regression to avoid requiring that users pre-segment their desired task into distinct subtasks. Performing such segmentation may require analysis beyond that of the end user, either due to a lack of the necessary skills, or the complexity of the task. Our algorithm instead automatically determines an appropriate number of subtasks and learns individual policies for them. We cast this process as learning the individual machine states of a finite state machine controller, where previous approaches have taken these states as given.

In the rest of this chapter, we discuss some of the challenges facing robot design, both of the physical form and the control system. The issues of human-robot policy transfer and multimap policies are introduced, and our approaches sketched out. We conclude with a summary of our contributions and an outline of this dissertation.

1.1 Dissertation Scope

When designing a robot, there are many issues to consider and challenges that must be overcome, such as the robot's physical form, computational architecture, and control policy. This dissertation only addresses one of them, that of **Human-Robot Policy Transfer** (HRPT), the transitioning of a control policy from a human user's mind onto a robot. Our approach is to use a combination of **interactive learning from demonstration** and **multimap regression**. Interactive learning from demonstration is a technique where the robot learns the task as demonstration takes place and provides feedback on the learned policy to the user. That is, the user can teach a task, observe robot performance of it, and provide additional, corrective demonstrations at interactive rates. A tutelage paradigm may allow for the generation of more targeted demonstration data, where portions of the task that are harder to learn are demonstrated more often than those that are learnt faster.

We use multimap regression to address the issue of **perceptual aliasing** (PA) in the demonstration data, where the current perceived state does not map to a unique correct action. PA can arise when portions of the state, required for the task at hand, are unobservable, or hidden. This hidden state may be due to inconsistent demonstrations, insufficient perception, or a change in task objective. A change in objective could be associated with a switch to a new subtask or reward function. Rather than combining the multiple observed outputs into one, we instead learn a one-to-many mapping directly in perception-actuation space.

To focus on the decision making aspects of robot control responsible for goal-directed behavior, this dissertation takes as given a complete robot, both physical and computational systems. With a known physical embodiment, sensors and perceptual capabilities, effectors and actuation control systems, we are interested in how the control policy can be transitioned from a human user onto the robot such that the robot performs the desired task in the manner expected. We focus on robots that are able to perform a variety of tasks that are demonstrable by humans, and thus do not consider

tasks that that humans themselves cannot do. While this focus may limit the applicability of our proposed techniques, future development may expand our contributions to the learning of tasks that humans themselves cannot perform well. Further, we note that many common desired tasks *are* human demonstrable, it is just that humans would rather not perform them themselves.

1.1.1 Beyond Consideration

As one of our goals is to develop a system that can be used to teach robots to perform previously unknown tasks, we strive to avoid incorporating task-specific knowledge. However, we do make the assumption that we have a robot that can physically perform tasks as desired by the user and on which we can instantiate desired control policies. An issue that we then do not address here is that of robot *embodiment*. This topic relates to the precise shape of the robot and the materials it is made of, and we particularly make no claims as to what these properties of the robot should be.

Research in related fields such as materials science, physics, and chemistry has resulted in an increase in the number of options available to robot designers. Specifically, there is now a wider variety of robots in terms of size, shape, sensors and effectors than there was before. While once "trash-can" robots with three or four wheels and sonar rings were state of the art, we now have robots that can climb walls [108] and utilize trinocular vision [120]. The increase in available embodiment options has led to a corresponding increase in the possible tasks that robots can perform. Additionally, research in psychology has lead to a better understanding of how humans perceive robots, and changed their outward appearance and behavior. Faces and other affective indicators are common [20], and studies have indicated how robot behaviors can be adjusted to suit individual personalities [136]. Thus *how* the robot performs a task is also a concern rife with possibilities.

However, no matter what the physical structure of the robot, the task it performs, and how it should be performed, the robot must be supplied with some method of control. That is, there must be some "cognitive" procedures that generate the control signals for the effectors based on observations or sensory signals. The mapping between the observations and control may be arbitrarily complicated, and the control system may make use of information in addition to the instantaneous sensory readings. Designing the controller can itself be a challenging task, and there are many different paradigms that may be applicable. At a low level, there are a plethora of possible computational architectures that can run on a robot, and more are being developed [76]. As with physical form, the choice of computing architecture and controller paradigm may make it easier for a robot to perform certain tasks over others, and novel desired tasks may require the development of new techniques.

Often, the robot's task and the environment in which it will be performed will inform the embodiment, and a robot's physicality is sometimes designed with certain tasks in mind. For instance, a home care robot for the elderly will likely look and feel much different from an interplanetary explorer robot. The former may be socially expressive, humanoid, and physically compliant, while the latter needs none of those properties and may instead be boxy and shielded against radiation. These are only two of the possible desired robot tasks, which may go beyond the stereotypical "Dirty, Dangerous and Dull" and change with time as perceptions of robots alter [137].

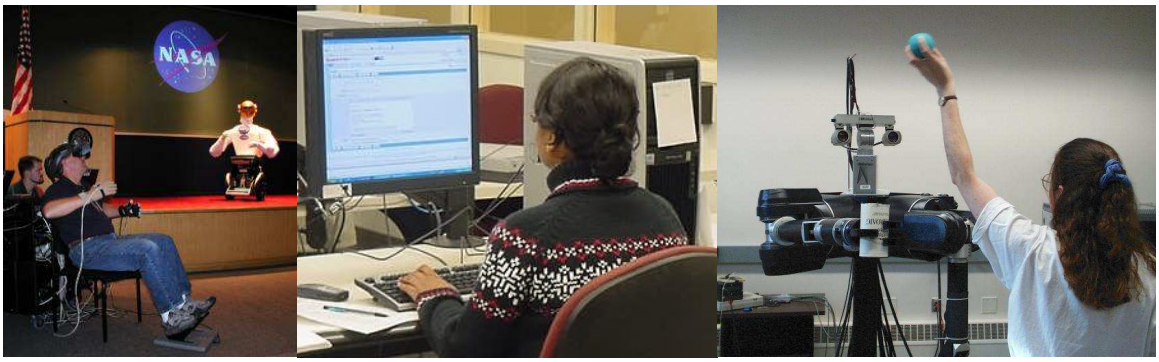
The interaction of a robot with a user can be broadly considered under the umbrella of *usability*, another topic which we do not directly address. In particular, as robots interact more with humans, intuitive interfaces and socially-correct interaction may become increasingly important. In order to guage the public's perception of robotic systems, we require user studies that have naive users interacting with and evaluating new designs in robots and interfaces. While such a user study is outside of the scope of this dissertation, we note that a desire for a more intuitive HRPT interface does guide our work, in our focus on learning from demonstration.

In terms of learning control policies themelves, we focus on interpolation between demonstrations to related situations, rather than extrapolation to completely novel scenarios. In order to do so, we depend on having demonstration data that covers the state and action space related to the task. Gathering the data itself is not a main focus of this dissertation, but we will discuss a distributed internet-based approach in Section 3.5.1. Inspired by "crowdsourcing" and "human computation," we aim to eventually collect the necessary massive datasets from a distributed userbase [6].

1.2 Human-Robot Policy Transfer

We roughly categorize HRPT techniques into 3 types, illustrated in Figure 1.2: teleoperation, procedural specification (which we will refer to as coding[1]), and learning, all of which are valid means of instantiating control policies for desired tasks onto robots. In any approach, a policy is taken to be a mapping from perceived state of the world to robot actions, and we assume that a human user has an appropriate control policy for the task and seeks to transition it onto the robot. We say that the policy itself is *latent* in the user's mind, and the user requires some means to computationally

[1] All approaches may require some sort of explicit coding, to instantiate the learning system or teleoperative interface. What we refer to as coding here is that on the part of the end user, not the robot designer/developer.



(a) Teleoperation (b) Coding (c) Learning

Figure 1.2: Three methods for Human-Robot Policy Transfer. Teleoperation requires the user's continuous attention and Coding requires mastery of a secondary skill set. Learning may combine the ease of teleoperation with the autonomy of coding while avoiding these disadvantages.

express it. The different techniques for HRPT can be seen as leveraging different information from the user, and each have their their own strengths and weaknesses as discussed below:

1. Teleoperation (Figure 1.2a) - A human directly controls the robot at all times by manually specifying actions that cause the robot perform the task.

2. Coding (Figure 1.2b) - A human explicitly writes a computer program or otherwise specifies some technique to automatically map robot perception to actuation.

3. Learning (Figure 1.2c) - The robot's control policy is implicitly derived from experience, guided in part by the human user.

1.2.1 Teleoperation

In teleoperation, a human is directly supervising the robot's decision making at all times. While there are many issues facing interface design and control mapping [117], the basic concept is that the robot's state (perhaps as extracted from sensor information) is presented to a human user, who then provides the control signals for the robot's actuators. This approach to HRPT is often the most manageable to set up and operate, as there are almost no secondary skills such as programming that must be learnt before such a system can be used. Once the user learns the teleoperative interface, they are able to control the robot to perform the task. Often, users are already familiar with teleoperative control from experience with remote controlled toys or video games [86]. However, more complicated interfaces are possible and may require more time to master [91].

Using teleoperation, it is often easy for the user to adjust the robot's behavior, as feedback is immediate. By directly observing the effects of their control decisions, the user can quickly make adjustments. This property is often desirable in research in human-robot interaction, where the human's response is the major focus of study. So called "Wizard of Oz studies" enable researchers to test out robot behaviors without having to explicitly program them. However, a major disadvantage to teleoperation is that it requires that the user be present at all times when the robot is in operation. For long periods of activity, exhaustion is an issue. Further, while operating the robot, the user must gain and maintain situation awareness of the robot's state and environment. Becoming immersed in the robot's situation can often lead to losing cognizance of the user's own environment, which can in turn lead to unintended, and perhaps undesirable outcomes [28].

In terms of the amount of knowledge that must be explicitly obtained from the user, teleoperation is the most minimal of the three approaches to HRPT. Instead of directly defining the control policy, users instead implicitly specify it by providing the appropriate control signals to perform the task. Humans may thus use teleoperation for HRPT without having to analyze the task itself.

1.2.2 Coding

At the other end of the spectrum, in multiple senses, are coding techniques. The first difference between the two is that coding approaches attempt to make the robot behave autonomously, or

without the constant human supervision required in teleoperation. In this scenario, the robot is designed to operate by themselves, allowing the human user to attend to other tasks. This freedom is paid for, however, by the high set-up costs associated with this type of policy transfer. Instead of using a teleoperative interface, autonomous behavior through coding requires that the policy must be instantiated in some robot executable/computable form. Often these policies take the form of a program, or computer code, but alternative forms such as hardwired circuits are possible. For the case of a program, writing good control code can be an arduous procedure, involving multiple rounds of testing and debugging. Adjusting the behavior to fit new circumstances may then involve rewriting code, which can incur significant downtime.

A further difference between teleoperation and coding is the amount of time that needs to be invested in learning the policy specification interface in the first place. While teleoperative devices are somewhat mainstream, and arguably intuitive, the same cannot be said of coding environments. Current techniques such as visual [39] or verbal [115] programming may alleviate this issue somewhat, but cannot remove it entirely. Achieving proficiency in coding in general, and a specific language and robot architecture in particular, can take years. Currently, only a subset of the population (computer scientists and hobbyists) have made this investment.

Lastly, coding requires that the user think about their latent control policy in an analytical, mathematical manner, in further contrast with teleoperation. Again, only a small subset of the general population has the skills necessary for this analysis, limiting the utility of this approach to HRPT. However, despite these drawbacks, coding is still arguably the preferred method of the three. Major contributing factors include the autonomous execution of behaviors and the robustness of well-designed policies. Looking forward, as robots and their policies become more complicated, performing the analysis required to write suitable controllers may become more difficult and alternate methods of HRPT may be more desirable.

1.2.3 Learning

Learning techniques span the distance between these two extremes, as illustrated in Figure 1.3, with various learning techniques making different tradeoffs between explicit policy information and initial supervision. With a learning architecture, a user can interact with a robot and teach it to eventually perform an (unknown) task autonomously. This interaction phase may be the analytical specification of goals and observation of eventual performance, or constant attention and control, as in teleoperation. However, once the task has been learned, the user can attend elsewhere as in the coding scenario. Note, there is still the set-up cost of programming the learning system, but this is incurred by the robot manufacturer, not the end user. Further, it is possible to enable a user to teach tasks in manners that do not require the user to master a large set of secondary skills, thus relieving them of all explicit coding responsibility. For instance, tasks may be taught simply by performing them, or *demonstrating* [14], while the robot observes. Alternatively, the user may *reward* [60] the robot when it behaves desirably.

Reward-based learning, or reinforcement learning (RL) [123], is a popular learning paradigm,

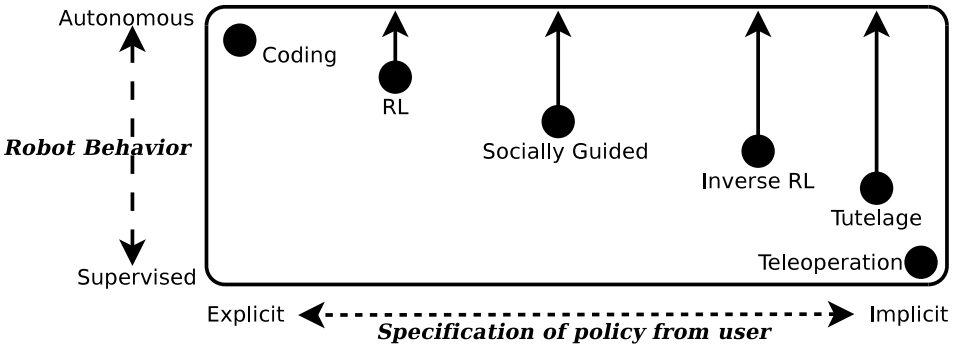


Figure 1.3: An illustration of the different approaches to HRPT. Teleoperation requires constant user interaction, but no explicit definition of the policy. Coding is very much the opposite, allowing for autonomous behavior, but requiring both skills and analysis. Learning techniques can span the difference, mixing different levels of explicit information and initial supervision, and eventually resulting in autonomous behavior.

where the robot attempts to determine a policy that is optimal with respect to rewards and punishments. An example in the case of navigation would be a robot that is rewarded when it reaches the goal, but punished for each step it takes. The robot is then incentivized to reach the goal as quickly as possible. In this scenario, users must supply the rewards and punishments. They can either do this directly, by observing the robot and administering the appropriate signals, or indirectly, by determining a mapping from the robot's state to rewards and punishments. However, specifying an appropriate reward function may be non-trivial and require the same sort of analysis that writing the complete policy would entail. Conversely, administering rewards directly may require the kind of focused attention that teleoperation does.

Learning from demonstration seeks to leverage more of the information obtainable from the user, by asking them not for scalar rewards, but for indications of what the appropriate action is to perform at a given time. However, this approach to HRPT has as a drawback that only tasks demonstrable by the user can be learned. In contrast, reinforcement learning can enable robots to learn tasks that are not directly performable by humans. Hybrid combinations of the two techniques may leverage the advantages of both, while overcoming their limitations.

1.2.4 Comparison

We do not claim that, in general, one of these approaches to HRPT is better than the others. Of the three, however, we note that work in learning is relatively less mature. Coding techniques, specifically programming, have several decades of computing behind them, and teleoperation interfaces are standard procedure for may robot installations.

Which of the techniques is ultimately used for a given application depends on a number of factors, including the tasks being performed, the robot being used, and the user in question. We can view general users as following the "path of least resistance," or doing the least amount of

work necessary to get an acceptably acting robot. This view leads us to consider the *return on investment* (ROI) as a measure of HRPT, which measures the ratio of robot performance to user time: $\texttt{ROI}(\pi) = V(\pi)/T(\pi)$. Informally, we will consider an abstract, unitless ROI, where a value of 1 represents "equality" in return and investment. In terms of performing some task, this equality point corresponds to the human user doing the task themselves.

Using ROI, we can sketch out an argument as to how improved robot learning techniques can lead to broader adoption and development of autonomous robots. With ROI it is important to see that it is not just the accuracy of the robot's control policy that matters, but also the amount of time it takes to develop it. Let us, for argument's sake, assume that for any learned policy (π_L) there is a coded policy (π_C) that outperforms it with regard to some measure of value, or $\exists \pi_C : V(\pi_C) > V(\pi_L), \forall \pi_L$. We can further suppose that the time it takes to code π_C for a skilled coder is less than the amount of time it takes to learn π_L: $T_{\texttt{code}}(\pi_C) < T_{\texttt{learn}}(\pi_L)$. The choice then appears clear, coding is always preferable to learning, as in terms of ROI, $V(\pi_C)/T(\pi_C) > V(\pi_L)/T(\pi_L)$.

However, the above case only holds for users who know how to code. For others, the time it takes to develop π_C would also have to include the amount of time it takes to learn to code in the first place. For some users, the resulting ROI may be too low, $V(\pi_C)/(T_{\texttt{learn}}(\texttt{code}) + T_{\texttt{code}}(\pi_C)) < 1$, and they would instead prefer to perform the task themselves (possibly using a robot via teleoperation). These users might, instead, choose to use learning to implement an autonomous robot control policy, if learning resulted in control policies whose value was high enough to make $V(\pi_L)/T_{\texttt{learn}}(\pi_L) > 1$.

Currently, average users whose ROI for learning is less than 1 are limited in what their robots can do, relying on teleoperation or autonomous controllers developed by others. They are further unable to modify the autonomous policies in ways not thought of by the original designers. Particularly, while built-in flexibility may allow for some adaptation, the coders may not have sufficiently anticipated the user's desired changes. As only a subset of the population has the skills necessary for coding, the onus for instantiating and editing autonomous robot control policies rests on them, creating a bottleneck. As education norms change, it may be that more and more users have the needed secondary skills and this bottleneck may disappear. Still, we argue that learning is an alternative that could emerge as the paradigm of choice.

Additionally, if this bottleneck is (partially) to blame for the current lack of multi-purpose autonomous robots, enabling more people to design and develop robot control policies may help spur growth in the field. Thus, we see this work as attempting to open up robotics to a wider audience. In this endeavor, we take inspiration from the synergy between computer graphics research and the graphic arts. There, computer scientists have provided the tools to perform 3D rendering, texture mapping, and the like, and a broad spectrum of content developers (companies, students, researchers in other fields, etc) use these tools to create high-end content. Similarly, we believe that, given the ability to perform HRPT, new developers will come forth and the field will flourish.

1.3 Learning from Demonstration

We investigate Learning from Demonstration (LfD) for HRPT, where a robot learns a task from an expert performer [61]. During the user's demonstration of the task, the robot collects state-action pairs and infers the underlying control policy, or mapping from states to actions. As mentioned, one benefit of LfD for HRPT is that that no further training for the user is necessary beyond that necessary to perform the task itself; if the user can perform the behavior, they can attempt to teach a robot to do the same. Whether or not they are able to depends on many issues, some of which are discussed below. We also note that LfD as described here is not meant to be a universal solution for instantiating autonomous control policies. For certain robots, tasks, and users, other options, such as explicit coding or statement of objectives, may be easier to perform and lead to better policies. LfD is another choice, that in some cases may be a useful alternative. Our research seeks to make it applicable in more cases.

1.3.1 Correspondences

Many of the issues that must be dealt with when using LfD relate to establishing *correspondences* between the demonstrator and learner. Some are purely physical, such as knowing which part of the robot (the gripper) corresponds to which part of the demonstrator (a hand). Other correspondences can be behavioral, associating observed poses and motions with available motion primitives. Actually recognizing the appropriate parts and motions of the demonstrator is an issue in its own right as well [69]. Common approaches outfit the demonstrator with sensors to reduce ambiguity, but may hinder free performance of tasks, although such systems are improving [151]. Generally, the physical and behavioral correspondences can be given a priori, or learned through experience. However, if they are not sufficient, it is possible that the demonstrator can perform tasks that the robot cannot.

Additionally, there are perceptual correspondences that must exist. Primarily, it must be the case that the learner is capable of extracting from the environment the information necessary to perform the task. For example, if the task is temperature dependent, the learner must have means of sensing temperature. Additionally, if the learner and demonstrator have differing viewpoints, ambiguities can result. Perspective matching [24, 144] may help resolve them, but differences in physical location and observable properties of the world may result in the demonstrator knowing information that is necessary for control, but not available to the learner.

We seek to avoid correspondence issues by using a teleoperative interface, where a user controls the robot to perform the desired task whilst only observing a representation of the robot's perceptual space. Such a setup aims to ensure that the task is robot performable (in that the robot itself performs the task) and robot decidable (in that the information present from the perceptual processes of the robot are sufficient). In the nomenclature of [10], both our record and embodiment mappings are the identity mapping, in that we assume no further processing needs to be done on the data to make it suitable for learning. Using this sort of scenario effectively combines the accessibility of teleoperative interfaces with the autonomous behavior of explicitly programmed robots. That is,

teleoperation is all that is needed to develop autonomous control policies. While it is true that this setup may require some training of the user before they can demonstrate tasks, many potential users are already familiar with our interfaces from current consumer products. In addition, our learning approach does not rely on the teleoperative interface, so as systems are developed that address the correspondence issues discussed above, they can be incorporated into our framework.

1.3.2 Learning Approaches

Having sidestepped issues of correspondence, we focus on the learning algorithm itself. Given appropriate data, we need to specify how the control policy is actually estimated. While there are many possible methods, we choose to attempt to learn the robot control policy using **Direct Policy Approximation** (DPA) techniques [118]. Taking the policy as a mapping between perception and action spaces, we seek to form an approximation of it directly from the data.

An alternate approach would be to apply reinforcement learning and use the demonstrations to place constraints on an underlying latent reward signal that the user is assumed to attempt to maximize. Sometimes called *Inverse Reinforcement Learning* (IRL), there are several techniques that can be used to find a policy that maximizes this discovered reward in the case where the reward is linear with respect to observable features [95]. An advantage to using RL techniques is that by modeling the reward function directly, robots can learn to outperform their demonstrators, in terms of that reward. Ideally, the learned reward function captures the desired task, and the task performance itself is also improved.

Within DPA, one approach is to take as known a parametric model of the task, and estimate the parameters that best fit a user's demonstration. Model approximations may be dealt with by learning additional higher-level task-specific parameters [13]. However, when learning unknown tasks models are, by definition, unavailable. Further, unknown tasks may have unknown or unlimited parameter spaces. We therefore instead use nonparametric non-linear approximation techniques, in an attempt to place fewer restrictions on the form of policies that can be learnt from demonstration. Specifically, they need not be linear in the perceptual features, or conform to a known parametric model.

1.3.3 Data Collection

We chose an incremental approach to the collection of training data, noting that it is often difficult to determine, in advance, what data will be sufficient for learning. For a given task, it may be that portions of the task are harder to learn than others, and thus require more data to learn, in that multiple demonstrations, each slightly different, may be required for sufficient generalization over perception-action space. This difficulty may arise due to the particular combination of task, platform, and learning, and be impossible to predict without a thorough understanding of all three.

In a traditional batch data collection scenario, all data is gathered before learning. However, without the full understanding of the complete system, we must rely upon user intuition as to which parts of the task require more data. In addition to the danger that the resultant dataset

is lacking necessary information, it could also contain extraneous points as well. This additional data is wasteful twice over, as the user must first spend time generating it, and then wait while the learning system processes it before the robot is usable.

To generate only sufficient and necessary data, we could instead build up the training dataset incrementally, evaluating the learned policy repeatedly along the way. The idea is that by observing the learned performance, the user can target additional data on the parts of the policy that are not yet learned, while not demonstrating the parts that are. The training paradigm then becomes a series of "demonstrate, train, test" cycles, where the user gathers data, then the learner processes, and the user uses an evaluation of the (updated) policy to guide their next demonstration. Such an approach could result in smaller overall datasets that are targeted with respect to the desired task and lead to reduced learning times.

In the extreme, a learner could update their learned policy after each datapoint is generated. If the policy update technique operated as fast as the data generation system, learning would occur in realtime, and the learner would be ready to use as soon as demonstration ended. The resulting dataset would be targeted in the sense of incremental generation, and also eliminate the wait associated with processing. The time it takes to develop a fully autonomous policy using this approach may thus be less than in both the batch and incremental frameworks. We call this method of data generation interactive training, or **tutelage**.

For tutelage to take place, inference, or the policy update stage, must be as fast as or faster than data generation. Therefore, we initially focus on incremental learning techniques that build up learned policies by considering each data point in turn. Given a current policy estimate $\hat{\pi}_t$, we want an approach that updates it using only the next datapoint $\mathbf{x}_{t+1}$ to produce a new policy estimate $\hat{\pi}_{t+1}$. We could also use batch learning and learn the complete policy anew after each datapoint is generated, if the full processing is fast enough. As computational hardware develops and improves, we expect that more processing will be possible while retaining realtime interaction.

Tutelage Concerns

As mentioned, we use a teleoperative interface to directly control the robot to perform a task and avoid correspondence issues. Providing more demonstration data during tutelage then involves taking over physical control of the robot. Interacting with the learned autonomy in this manner can be seen as a form of sliding autonomy [40], where the degree of autonomy of the robot is controlled by the user. In particular, during demonstration the autonomy of the robot is shut off entirely, as the user makes all of the actuation decisions. An alternative would be to allow mixed autonomy, where the learned system's outputs still effect the robot's behavior in some fashion.

Another concern is that simple observation of the robot's behavior may not include enough information to guide the user in creating more demonstration data. Specifically, the user may be aided by feedback as to what the robot has learned [25]. By revealing the internal state of the learner, for instance, a teacher can more accurately understand what errors are being made [38]. Alternatively, the learner can report a confidence score, indicating how sure it is that it has learned

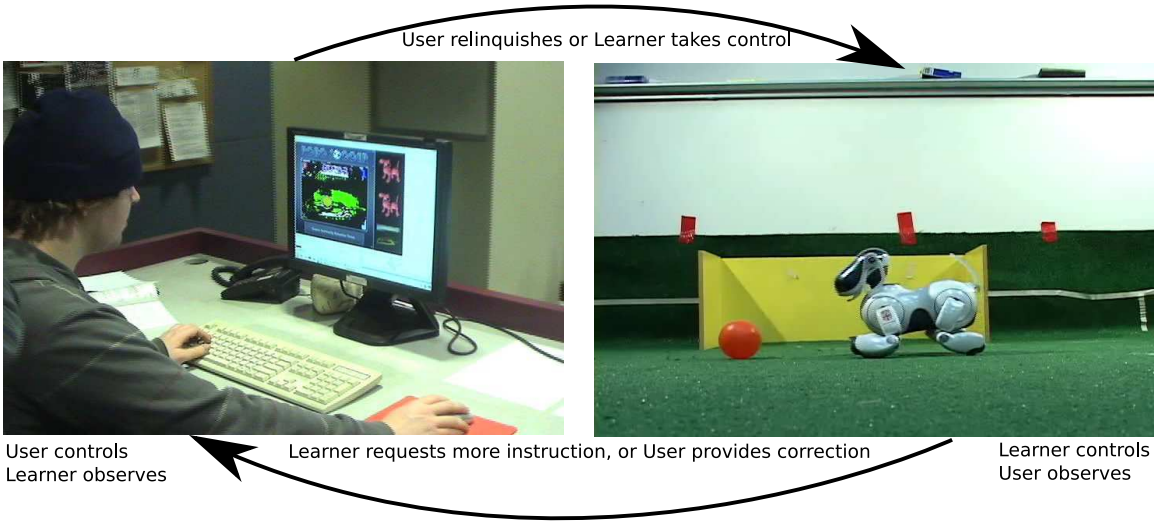


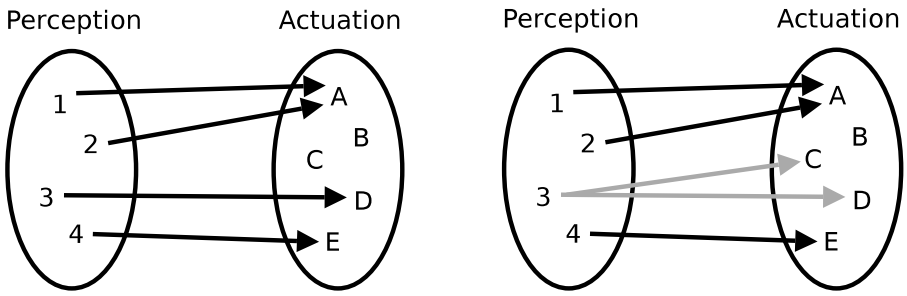
Figure 1.4: An illustration of mixed-initiative control. While control of the physical robot is shared between the user and learning system, only one has command at a given time. Both parties have the ability to *take* or *give* control to the other, as they deem necessary. If both attempt to take control at the same time, an arbitratory process decides who gets it. Likewise, if neither wants control, a default controller may take over.

the policy correctly, for a particular state, or overall. Such confidence measures can be used to prompt the user for more demonstration, in a form of active learning [43].

This dissertation provides a framework for robot tutelage that incorporates **Mixed-Initiative control** (MIC) for performing aspects of both sliding autonomy and active learning [3]. Illustrated in Figure 1.4, MIC enables both the user and the learned autonomy to take and give control of the physical robot based on their confidence in their policy. When the autonomous system's confidence is low, it gives control to the user (active learning). Conversely, the user can take control away from the autonomous system (sliding autonomy) to provide corrective demonstration. Likewise, the autonomous system can take control, and the user can give control when desired. If both the user and the autonomy want control of the robot, arbitration between the control signals is necessary. Arbitration can be as simple as a "higher wins" multiplexer, or as complicated as control fusion [97].

1.3.4 Multimap Policies

The decision making control policies themselves can take many forms. Viewed as a mapping $\pi(\hat{\mathbf{s}}) \rightarrow \mathbf{a}$ from perceived state $\hat{\mathbf{s}}$ to action $\mathbf{a}$, we concern ourselves with the two possibilities shown in Figure 1.5. In a univalued mapping (unimap), each perception leads to a single actuation that should be performed, although multiple perceptions can lead to the same actuation. A multivalued mapping (multimap), in contrast, allows for there to be multiple correct actuations associated with a single perception. Again, multiple perceptions can map to the same (sets of) actuations. Additionally, in both situations, there may be actuations that are not associated with any perception.



(a) Unimap (b) Multimap

Figure 1.5: The two forms of mappings from precepts {1 2 3 4} to actions {A B C D E} that we are concerned with for control policies. The only difference in the diagrams is the line from 3 to C in the multimap case, resulting in the highlighted multimap scenario. Correspondingly, multimaps allow multiple actions to be mapped to by a single perception, while unimaps allow for only one. In both, some actions {B} may not be mapped to at all.

Actually learning the mapping from collected data $D = \{\hat{\mathbf{s}}_i, \mathbf{a}_i\}, i \in 1 : N$ is an example of regression, or learning the relationship (π) between independent ($\hat{\mathbf{s}}$) and dependent ($\mathbf{a}$) variables. However, as discussed above, we do not assume a known parametric model for π, nor do we assume a linear relationship between $\hat{\mathbf{s}}$ and $\mathbf{a}$. In this case, both parametric and linear regression techniques are inapplicable, and we instead consider nonlinear, nonparametric approaches. A common approach is to model the observed data as corrupted by noise, perhaps Gaussian, as in

$$\mathbf{a} = \pi(\hat{\mathbf{s}}) + \mathcal{N}(0, \sigma^2)$$

Other noise models are possible, and may be appropriate for different settings. Putting this model in distributional terms, we can define the likelihood of an action being associated with a given perception as

$$\mathbf{a}^* \sim P(\mathbf{a}|\hat{\mathbf{s}}) = \mathcal{N}(\pi(\hat{\mathbf{s}}), \sigma^2) \tag{1.1}$$

Performed actions, $\mathbf{a}^*$, are drawn from this distribution by the demonstrator or learned controller.

To estimate π with this model, a Gaussian distribution is fit to the observed actions for a given perception. Computationally, they are averaged together to estimate the mean of the correct policy output distribution; we ignore here the calculation of σ^2. This approach to policy learning, as most other standard regression techniques, makes the assumption that there is *one* correct output for a given input. In other words, it performs **unimap regression**, and uncovers a unimap policy. To see this assumption, consider a multimap policy under which $\pi(\hat{\mathbf{s}}') \rightarrow [\mathbf{a}_1 \text{ or } \mathbf{a}_2]$. The above model, when queried with $\hat{\mathbf{s}}'$ will return from $\mathcal{N}(\frac{\mathbf{a}_1+\mathbf{a}+2}{2}, \sigma^2)$, assuming that both observed outputs are noise-corrupted versions of the true target.

The unimap assumption is equivalent to assuming that the policy is *Markovian*, or that the current state estimate contains all the information necessary to determine the correct action to perform. However, in many control policies the Markovian assumption is false, due to *hidden state*. By hidden state we mean state information about the world that is necessary for choosing the

appropriate action, but is not present in $\hat{\mathbf{s}}$. The hidden state itself can be physical, and not detectable by the robot's sensors, or mental, that is, latent in the demonstrator. Due to the resulting lack of knowledge, two perceived states may appear the same, but correspond to different true states. If the true states require different actions, the resulting mapping is a multimap.

In a multimap, the underlying distribution of actions given a perception is multimodal. If there is no noise, there is an impulse at each possible output. Compare this with the unimap case, which has a single impulse at the correct output. When corrupted by Gaussian noise, the unimap's single impulse becomes a Gaussian distribution, while the multimap scenario results in instead a mixture of Gaussians, one for each possible output. Note that the *noise* model has not changed, it is still a single Gaussian, which becomes multimodal when convolved with a multimap policy. The resulting model is thus:

$$\mathbf{a}^* \sim P(\mathbf{a}|\hat{\mathbf{s}}) = \frac{1}{Z}\sum_{k=1}^{K}\mathcal{N}(\pi_k(\hat{\mathbf{s}}), \sigma^2) \tag{1.2}$$

where $\pi_k(\hat{\mathbf{s}})$ are the K possible actions for this perception and Z is an appropriate scaling factor. For DPA, we must then fit a multimodal distribution to the observed data, and one issue that must be addressed is how to determine K.

The above described multimap scenario is exactly that of **perceptual aliasing** [152], where perceptually similar states require different actions. One approach to dealing with multimaps is to redefine the state space so that the Markov property holds, and thus unimap regression can be applied. This redefinition can often be accomplished by considering a history of previous perceptions in addition to the current one [16]. However, there is then the question of how much history to consider. In addition, each additional history step increases the dimensionality of the perception space, which can then lead to both data sparsity and data storage problems.

Alternatively, we can operate with the multimap itself, and estimate K directly. For example, in RL, Partially Observable Markov Decision Processes (POMDPs) deal with hidden state explicitly by modeling the connection between true state and perceived state via an observation mapping [148]. The agent then uses observations to update a belief of the true state, and learns a policy from beliefs to actions. To determine K, algorithms such as U-Tree [88] discover perceptually aliased states and expand the belief space.

A different approach is to break the policy itself into subpolicies or subtasks, each of which is Markovian. Again, unimap regression can then be applied at the subtask level, and the subtasks ordered or combined to accomplish the overall task. In a distributional view, each of the subtasks would be one of the π_k. Estimating K can be performed by any technique that performs model selection in a multimodal distribution [65].

In addition to hierarchical tasks, multimap policies are also related to the issue of feature selection. Specifically, the existence of a multimap indicates that necessary features are missing. Thus, their presence could be used to guide feature selection systems. If a multimap occurs, new features must be added to render the underlying policy un-aliased. Conversely, if the removal of a feature does not result in a multimap policy, then the feature is not needed for control.

Returning to our approach, recall that we assumed that the perceptual process of the robot provides enough information to decide the desired task. In fact, our teleoperative interface seeks to ensure this assumption. The existence of multimap policies then would indicate that this is not the case as by definition necessary information is missing. This information must then reside in the user's mind, possibly due to advanced perceptual processing, the use of memory, or additional task-specific knowledge. It is the purpose of multimap regression to uncover not necessarily what the information is, but how it affects the observed outputs. That is, we do not seek to learn a statement such as "The demonstrator has a latent variable named V that takes on the values $v_1 ... v_n$," but rather one such as "The demonstrator has a latent variable, and for some of its values we observe the action $\mathbf{a}_1$ and for others we observe $\mathbf{a}_2$ associated with the perception $\hat{\mathbf{s}}$".

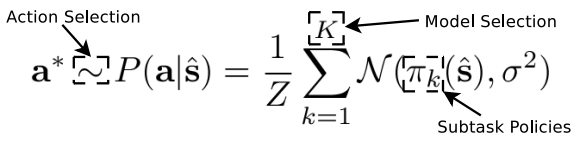
Finite State Machines

We chose to view the policy as composed of subtasks and model it as a Finite State Machine (FSM), where the overall task is composed of one or more subtasks, each of which can be composed of more subtasks, etc. Without loss of generality, we consider here only FSMs whose subtasks are Markovian, as any hierarchical FSM can be "flattened" to a single level. Learning a full FSM from demonstation can be broken into three steps, as depicted in Figure 1.6a:

1. Learn the number of subtasks. (model selection)
2. Learn each subtask's policy. (policy learning)
3. Learn to transition between subtasks. (transition learning or action selection)

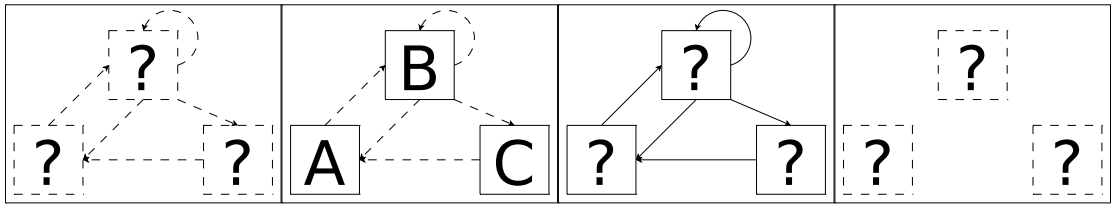
An optional additional step is to improve the policy past that of the demonstrator, which can be performed at both the subtask and total policy level.

Each of these steps corresponds to a portion of using the model in Equation 1.2, as annotated below. Step 1 is the estimation of K, Step 2 is the learning of the individual π_k, and Step 3 indicates which of the K possible actions should be returned. As written, this equation simply samples stochastically from the full multimodal distribution.



$$\mathbf{a}^* \sim P(\mathbf{a}|\hat{\mathbf{s}}) = \frac{1}{Z} \sum_{k=1}^{K} \mathcal{N}(\pi_k(\hat{\mathbf{s}}), \sigma^2)$$

Techniques already exist for performing some of these steps, given certain portions of the full FSM. For example, our discussion above on unimap regression highlights one approach to Step 2 [58]. Similarly, other techniques can be used to learn the individual subtask policies [95], but they



(a) Full FSM learning (b) Transition learning (c) Policy learning (d) Multimap Regression

Figure 1.6: (a) The three steps in learning an FSM: Learn the subtasks (boxes), their policies (letters), and the transitions between them (lines). Dashed lines and question marks indicate unknown quantities. Previous work has learned transitions given policies (b), or policies given transitions (c). We propose to perform both model selection and policy learning with multimap regression(d).

all require that the data already be separated out by expert. Alternatively, given the subtasks, the transitions between them can be learned by establishing pre and post conditions for the execution of each one [96] (Figure 1.6b). Or, by assuming a known task decomposition it is possible to allocate data and learn the subtask policies themselves [130] (Figure 1.6c).

Approaches to Step 1 are rarer, and such an approach is one of the primary contributions of this dissertation. Specifically, we approach both Steps 1 and 2 directly (Figure 1.6d), by treating the policy as a multimap and using an infinite mixture of experts to automatically determine the number of subtasks, assign data to them, and learn their policies. Techniques for Step 3 could then use these learned subtasks and infer the required transitions for complete FSM learning.

1.4 Dissertation Overview

We claim that a current significant challenge on the path to autonomous, multi-purpose robots is that of HRPT, transforming control policies latent in users' minds into forms that can be computed by robots. Traditionally, HRPT for autonomous policies is done by explicitly coding the policy, but robot learning, and LfD in particular, may represent an approach that enables more users to develop satisfactory control policies in less time. A long term goal of this work is to enable non-programmers to transfer policies at levels comparable with those transferred via coding. For example, imagine a user purchasing a team of robots, demonstrating how to play soccer in one form or another, and having the robots be able to play competitively against a hand-coded team using the same strategy.

Our first hypothesis is thus:

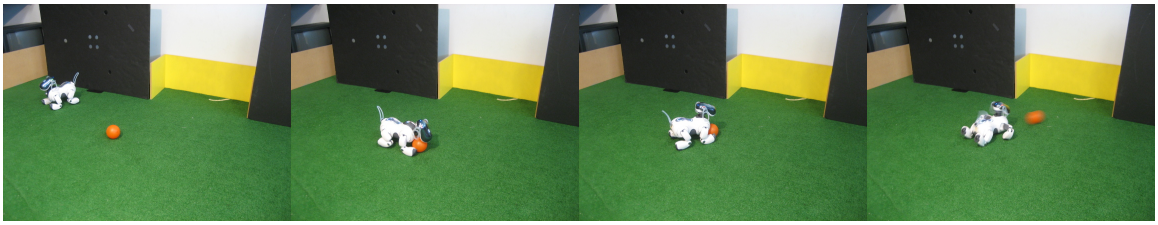
> **Hypothesis 1:** Assuming a platform capable of performing a particular task, robot tutelage using standard regression is a viable method for HRPT for that task, and can result in policies on par with their coded counterparts in terms of action error and task-specific metrics.

(a) Human demonstration (b) Learning (c) Autonomous execution (d) Further tutelage

Figure 1.7: Use of Dogged Learning. The user first demonstrates the behavior (a) while real-time learning approximates the underlying policy (b). Upon observing autonomous execution of the estimated controller along with feedback from the learning system (c), the user can chose to provide additional tutelage (d).

To test this hypothesis, we have developed the **Dogged Learning** (DL) framework, the use of which is depicted in Figure 1.7. Robot sensor-action data is gathered as a human teleoperatively controls it to perform a desired task. Real-time learning updates the robot's learned autonomy as the demonstration takes place and reports a confidence rating. The user can, at any time, stop demonstrating and the robot's autonomy will take over and perform the task as learnt. Further demonstration can be given as desired by the user, and even asked for by the robot.

Using standard sparse incremental regression techniques (Locally Weighted Projection Regression [149] and Sparse Online Gaussian Processes [42]), we train a Sony AIBO on a variety of soccer related tasks, including the goal scoring and goalie behaviors shown in Figures 1.8 and 1.9. The goalie behavior is successfully approximated, as are portions of the goal scorer. However, the complete scoring behavior is not learned, due to issues of perceptual aliasing, but an alternate formulation that avoids perceptual aliasing *is*. From these experiments we conclude:



(a) Seek (b) Trap (c) Aim (d) Kick

Figure 1.8: A finite-state-machine control policy for robot soccer goal scoring, learned herein. A multimap scenario occurs in (c), when the robot cannot detect the ball's location. It is then unsure if it should seek the ball (a), or kick to score (d).

(a) Search (b) Lock (c) Block (d) Clear

Figure 1.9: The goalie behavior. Not a multimap, thus it is learnable using standard regression.

Conclusion 1: For unimap controllers without perceptual aliasing, standard regression can infer control policies from interactive learning from demonstration that perform comparably to their demonstrator. However, the same techniques are unable to learn multimap policies.

To more generally address the issue of hidden state in demonstrated control policies, we use multimap regression. Casting policies as FSMs, our approach addresses both the issue of model selection and policy learning simultaneously. That is, we assume that the data is generated by a collection of lower-level subtasks, each of which is Markovian, that are switched between to perform the overall task. Looking again at Figure 1.8, this policy cannot be learnt with unimap regression, as knowledge of which subtask to perform is not perceived by the robot. We aim to estimate both the number of these subtasks and their individual control policies from unsegmented data, and thus hypothesize:

Hypothesis 2: From unsegmented demonstrated multimap data, an infinite mixture of experts approach can automatically infer a number of subtasks, and their individual policies, that can then be used to reperform the demonstrated task at a level unobtainable with standard unimap regression.

We developed the ROGER (Realtime Overlapping Gaussian Expert Regression) model, which consists of an infinite mixture of experts, where each expert is a hypothesized subtask. Inputs for each expert are assumed to come from a Gaussian in input space, and outputs are generated from sparse Gaussian process regressors. The number of experts that contribute to the model (K in the above notation) is determined by the Chinese Restaurant Process, an incremental procedure for producing sequences of numbers, where common ones become more likely, but there is always the possibility of producing a new number (or expert) [140].

When applying ROGER to the multimap goal-scorer of above, we discover a number of experts

and their policies that when switched between appropriately have been shown to perform the overall task. In our current system the learned controller is not as good as the demonstrator, but these residual errors may be due to simplifying assumptions made in both our model and implementation. Future development may improve results, and we thus conclude:

Conclusion 2: Multimap data from an FSM robot controller can be automatically partitioned into subtasks using incremental multimap regression. The resulting subtasks can be used to replicate the controller at a level beyond that of one learned with standard regression.

As an example of the limitations of the current system, we only currently consider Gaussian distributions in input space, and require that all experts share the same parameters. Further, we point out that ROGER only addresses two of the three steps to learning complete FSMs. In our experiments we used a coded switcher to select the appropriate expert for execution when performing the task. To improve its usefulness, ROGER will need to be combined with an approach to learning the transitions between experts themselves.

1.4.1 Contributions

In pursuing this research and answering these hypothesis, we make two major contributions to the field of robot learning:

1. Dogged Learning (DL): An architecture for interactive, mixed-initiative robot lifelong learning from demonstration for unknown tasks.
2. Realtime Overlapping Gaussian Expert Regression (ROGER): An incremental multimap regression algorithm suitable for robot tutelage.

The Dogged Learning architecture incorporates ideas of interactive learning, mixed initiative control, feedback, transparency and confidence for RLfD. Further, it has been designed to be agnostic, allowing different possibilities to be instantiated for the platform, demonstration interface, learning algorithm, and even arbitration scheme. The complete system and our developed modules are available for general academic use.

Using DL, we can perform direct comparisons between different possibilities by switching one module and holding the others constant. For example, we have been able to directly compare different learning algorithms for use on the same platform and task. While we only tested statistical regression techniques, the methodology itself is abstract and can be used with other forms of learning.

ROGER has been developed to address a need for learning multimaps. From our initial experiments, we concluded that perceptual aliasing causes multimaps, and the resulting control policies cannot be learned with standard unimap regression. Available techniques for learning tasks composed of subtasks rely on some prior knowledge of the task structure, such as subtask policies or transition locations. By learning multimaps directly, we seek to enable the learning of FSM-style controllers without the need for this information. ROGER is designed to be used in the DL framework, and thus is incremental and sparse.

Again, our long-term goal is to enable non-programmers to instantiate robot control policies at the same level of expertise as programmers. While the work presented here is a step on that path, this goal has still not been attained as our approach is not a method for complete FSM learning, as in Figure 1.6a. However, as we can now provide the subtasks and their policies, it is possible that techniques for doing transition learning, as in Figure 1.6b, can be combined with our approach to achieve full FSM learning. Additionally, our current work assumes that the demonstrator is a source of correct (even optimal) data, from which our system generalizes to new situations. However, demonstrators, and human demonstrators in particular, are not perfect, and often make errors, or suboptimal decisions. Extending our methodology to incorporate both positive and negative reinforcement may allow learners to leverage higher goal-directed information, and learn to outperform the demonstrator themself, or learn policies not performable by the demonstrator at all.

1.4.2 Outline

This chapter introduced the dissertation and situated the work with respect to the greater robotics community. Chapter 2 expands on some of the related work in Robotics, Machine Learning and Human-Robot Interaction. Chapter 3 describes our first contribution, Dogged Learning, and provides implementation details. Chapter 4 describes our second contribution, ROGER, and reviews related approaches to regression. We present our experiments in Chapter 5, demonstrating that tutelage and regression can be used to teach unknown tasks to robots without explicit coding. However, certain FSM control policies are not learnable with unimap regression, so we use multimap regression to infer a number of subtasks and their policies from unsegmented data, which can then be leveraged to perform the original task. In Chapter 6 we discuss the advantages and limitations of our current system, lay out directions for future work, and conclude.

Chapter 2

Background

> *Computers today exceed human intelligence in a broad variety of intelligent yet narrow domains such as playing chess, diagnosing certain medical conditions, buying and selling stocks, and guiding cruise missiles. Yet human intelligence overall remains far more supple and flexible.*
>
> Ray Kurzweil, The Age of Spiritual Machines, 1999, page 2

We argue that one method for achieving greater flexibility in what tasks robots perform, how they do them, and more generally how control policies are instantiated, is to improve robot learning from demonstration. In our pursuit of better RLfD techniques, we heavily overlap with research in the fields of machine learning and human-robot interaction, which address, respectively, how a control policy is inferred from data, and how the data itself is gathered from humans. In this chapter we survey related previous work from these two areas and also from the area of robotics, specifically different techniques for representing and instantiating control policies.

2.1 Robot Platform

A robot's interaction with the world can be framed as control loop, as in Figure 2.1, where information flows from the environment, through sensors and perceptual processing, to the robot's decision making capabilities. Once an action to perform has be determined (in some fashion), it is transmitted back into the environment, via actuational processes and the effectors themselves. Within this loop, our primary concern is with decision making, how information extracted from the physical world is mapped by the robot into actions. We call this mapping the robot's control policy, π, but before we discuss the various forms and abstractions that can be used to define policies, it may be useful to examine the possible computational architectures in which they may be embedded. To some extent, these choices are less important than those of the robot's embodiment, discussed in Section 1.1.1 and displayed on the left of Figure 2.1, as suitably designed interfaces should enable any policy to operate the embodiment through any architecture. However, in practice, different architectures lend

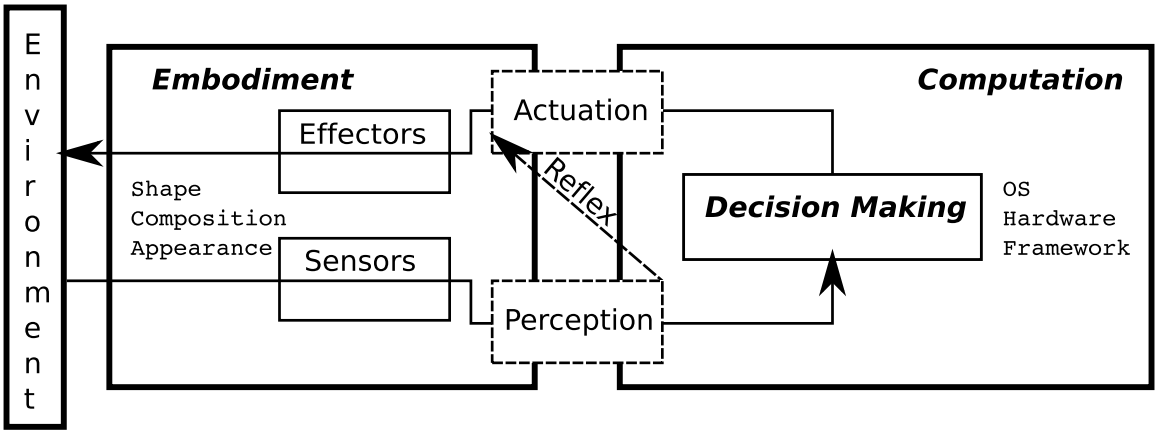


Figure 2.1: The basic robot control loop. Sensors and effectors, part of the robot's physical embodiment, interact directly with the environment. Decision making, embedded in a computational architecture, performs task-specific control. Perception and actuation, which may be either computational or embodied, transform data to and from the space of decision making, while reflexes bypass decision making all together. If perception and actuation do not exist, decision making occurs directly in the raw sensor and effector spaces.

themselves more easily to some policies than others, and the choices for embodiment, architecture, and decision making can all be interrelated.

We focus now on the computational portion of the robot, on the right of Figure 2.1. At the lowest level of the architecture, where it interfaces with the embodiment, is the actual computing hardware. Limiting ourselves only to modern electronic, digital computers, there are multiple options that have been used for robot "brains," such as standard consumer machines, custom-built boxes, or single-board computers. Each of these choices has associated with it additional options, such as the address-size (32 or 64, for example), memory size, ports, etc. If no consumer system is appropriate, perhaps due to required interfaces or desired capabilities being lacking, custom-made systems, designed from the chips up, may be used as well.

On top of the hardware runs the robot's operating system (OS). By this we mean not only the OS that runs the hardware, but also the "environment" in which the control program runs, if it is separate. Again, options abound, and choosing one can limit the capabilities of the robot, or the pool of developers. Standard OSes may be appropriate as is, although for robotics realtime or custom variants such as QNX[1] or Robobuntu [84] may be necessary. There have also been some extensions developed specifically to ease the creation of robot control policies that can then be used on multiple robots, such as Microsoft Robotics Developers Studio[2] and ROS [109]. The latter is an open source effort, designed to enable researchers to develop reusable modules. Alternatively, some robots, such as our Sony AIBO, have proprietary systems[3], which require special development tools.

[1] http://qnx.com

[2] http://msdn.microsoft.com/en-us/robotics/default.aspx

[3] The AIBO's is OPEN-R: http://www.aibo.com/openr

The data flow through the lower levels can put limits on the framework that is used to define control policies. For example, some robot systems may be push-driven, where changes in sensors are sent directly to the control system. The robot controller would then have to have a set of appropriate interrupt handlers, and possibly a main computation loop that incorporates information from multiple readings. An alternative is to have a pull-based architecture, where sensor readings are not generated until the control system requests it. The control system then has to explicitly poll the sensors to get inputs. The output side of the equation can be similarly split, with effectors requiring updates and interrupting to obtain them, or only reacting to sent commands. Hybrid approaches can blend these two extremes, where perhaps some portions of the robot are interrupt driven, and others not.

For this work, we have designed Dogged Learning to abstract away the robot's embodiment and computational architecture, and focus on the control policy itself. However, it is within the chosen robot platform that the control policy must exist, building on top of all the previous design decisions. The plethora of choices that have to be made up to this point can contribute to the difficulty in replicating results in robotics research, and reusing previously developed systems. Particularly, in our area of research, individual learning systems are often tied closely to their robot's physical embodiment, computational architecture, and possibly the specific tasks it performs as well. It then becomes troublesome to reproduce results that were obtained on different systems, or even transfer approaches to new robots. Careful use of abstraction and the adoption of standards could alleviate this problem somewhat.

2.1.1 Perception and Actuation

In addition to the raw sensors and actuators of the robot, our control loop in Figure 2.1 allows for perception and actuation processes that are independent of decision making. These processes are responsible for transforming data from the low-level representation of the underlying hardware to a higher-level representation suitable for decision making. Examples include vision systems that turn raw camera pixels into objects or faces, or Proportional-Derivative-Integral controllers that turn desired positions into motor torques. Note, that these processes can be instantiated either as part of the embodiment (in hardware) or computation (software).

Theoretically, it is possible to make policy decisions based only on the raw data, without further processing, as all the necessary information is, by definition, present. Practically, however, perception and actuation processes are used to simplify controller development. Bear in mind that using these processes can unintentionally bias decision making, by making some control policies easier to instantiate than others. Perception acts as a feature selection system, removing some data from consideration while highlighting others when making command decisions. Likewise, actuation can be seen as providing motor primitives, making some actions easy to perform while forbidding others.

While any connection between perception and actuation is technically a form of behavior, or decision making, we distinguish between those that are built into the robot, and those under the control of the (learned) autonomy. Those built in can be seen as connecting perception directly to

actuation, bypassing the decision making component of the system entirely. These couplings are therefore similar to reflexes, in that the control policy cannot effect them, although it can leverage them in its activities. Inverted pendulum robots, for example, often have built in balancing reflexes, which navigation utilizes by shifting the center of mass of the robot, instead of directly driving the wheels [78]. The reflexes themselves can be implemented computationally, as separate running processes or subroutines, or in the embodiment itself, with circuits and physical connections.

2.2 Decision Making

Conceptually, the act of controlling a robot can be divided into three stages: Sense, when new information from the environment is detected; Think, when computation is performed; and Act, when signals are sent to the actuators. The actual order and manner in which these stages take place defines the *form* of the robot's control policy, or the robot control architecture that is used to implement it. We distinguish this from the *model* of the control policy, which is how the decision making process itself is conceived. While we consider the two aspects in isolation, the choice of one can effect the choice of the other.

2.2.1 Policy Form

We discuss four main forms of architectures for robot control policies, as shown in Figure 2.2. When choosing one to use, a major concern is the amount of time between when something is sensed and appropriate action is taken. Generally, as more time is spent processing, more in-depth ramifications of possible actions can be considered. However, increased delay means that the environment may have changed in the meantime, rendering the processing moot.

Deliberative Controllers

Policies that search over possible outcomes are termed *deliberative* policies, referring to the fact that the robot considers many possible actions and deliberates on which is best. Shown in Figure 2.2a, policies of this form collect data from the world, process it all, and then generate actions to perform. These approaches are typified by long computational times, and thus may be inappropriate for highly dynamic situations. Because decisions are made by considering all of the possibilities and using all available data, deliberative systems are sometimes said to be examples of "top-down" control.

Planning is one form of deliberative control, where the robot precomputes an entire sequence of actuations to achieve its goal [77]. To do so, a robot may build a precise model of the world, in the form of a map [142], which it can then use for both for navigation and localization, as in SLAM (Simultaneous Localization and Mapping) [120]. The main advantage of deliberative policies is that they can consider long term effects of actions, and formulate control that will optimize desired features such as travel time or energy use. That is, they can often "think their way out of" difficult situations. Consider navigation, where loops and hazards in the environment may make getting to

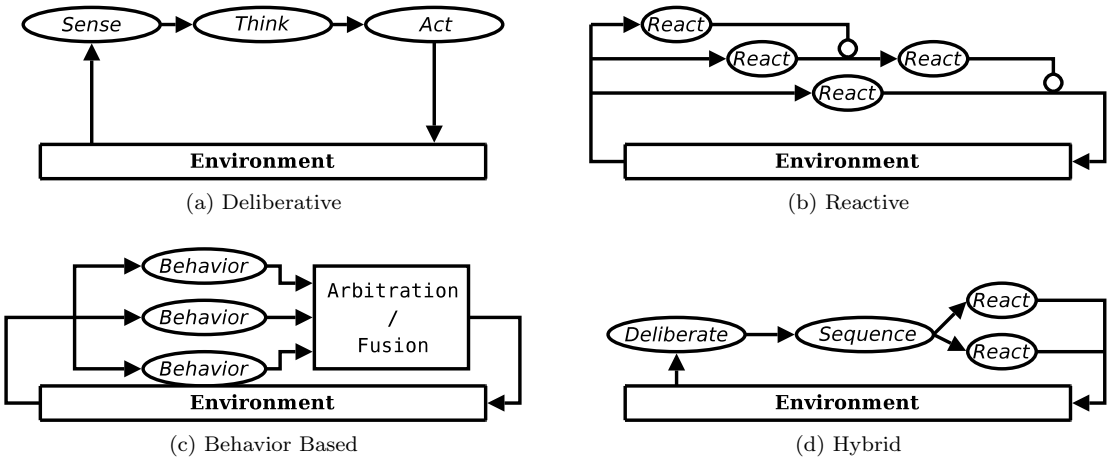


Figure 2.2: Various forms that a robot control policy can take. Deliberative policies (a) can determine global strategies, but may not be able to deal with highly dynamic situations. Reactive controllers (b) make the opposite tradeoffs, responding rapidly but with minimal processing. Shown is a subsumption controller, where higher-level reactions subsume on lower level ones. In a behavior based architecture (c), the final policy emerges from the combination of multiple behaviors. Hybrid controllers (d) use deliberation to select reaction.

a goal location difficult. A deliberative system is able to avoid these trouble spots by planning a global path. Being able to determine it, however, requires accurate models (both of the world and the noise in it), and sufficient processing time.

Reactive Controllers

At the other end of the computation time spectrum are *reactive* controllers, which attempt to minimize the "think" portion of the cycle. In this dissertation, we seek to learn reactive policies, where the estimated state of the world is translated directly into robot action, without planning or deliberation. Such policies can be viewed as direct couplings between perceptual variables and actions, where what is immediately observed directly and completely determines the action taken [74]. As computation time is minimized, reactive controllers are more suitable for highly dynamic situations, when the environment changes rapidly. However, as they do not consider the global situation, they can get stuck in sub-optimal behavior. In the navigation task above, a reactive controller, returning to the same location via a loop in the environment, will perform the same action as it did before, and may get stuck infinitely looping.

Reactive controllers are very similar to the reflexes discussed above, as both directly connect sensing to acting. We therefore again distinguish between capabilities that are built into the robot, and those that are being actively developed for performing some task. The line between the two is ambiguous, and can be drawn in different places by different researchers, or even by the same researcher in different cases. For instance, our robots have LEDs that indicate the status of various parts of the robot (battery level, wireless reception, etc). These behaviors were not built into the

robot, so we had to develop them ourselves. However, as they are not relevant to the tasks we are concerned with learning and are not accessible to the controllers, we consider them reflexes. If, instead, they were part of the behavior that we were trying to teach, and the control system had access to the LEDs, we would consider them part of decision making.

As one reactive controller represents a direct coupling from preception to actuation, a collection of reactive controllers represents a set of possible reflexes. To choose between them, they could be arranged hierarchically, as in Figure 2.2b. The result is a subsumption architecture, where higher-level reactions subsume on lower-level ones, modifying their inputs to generate desired outputs [26]. As the overall policy is then built out of a collection of individual reactors, each of which relies on those below it, subsumption architectures are often said to be designed "bottom-up."

Behavior Based Controllers

Another way of looking at a collection of individual controllers is to treat each as an individual *behavior*. Acting at the same time, the overall policy itself then emerges from their interaction [11]. Again, a method for choosing the actual action taken from among the proposals from all behaviors must be included, as in Figure 2.2c. Two possibilities are *arbitration*, where one behavior is given control, and *fusion*, where the multiple outputs are merged to create a new one. Note, that the subsumption architecture described above is one particular choice of behavior output combination.

Because the overall robot behavior is broken into multiple, parallelly active, components, behavior based forms of control policies are well suited to distributed systems. Each behavior could be in a separate thread of computation, possibly on a different processor, and would receive inputs from the sensors it requires, and send outputs to the combiner. The entire robot controller would then result from the asynchronous communication between components.

Hybrid Controllers

In a behavior based architecture, it is possible for the various behaviors to operate on different timescales. In the extreme, one behavior could be a deliberative planner, while another in the same robot can be reactive reflex. The result would then be a *hybrid* controller, which attempts to combine deliberation and reaction to get the advantages of both without their disadvantages. Another type of hybrid controller is a *Three-Layer Architecture* (3LA) [51] where a sequencing component uses the plan from deliberation to chose reactive controllers to execute, as shown in Figure 2.2d.

The resulting controller can be the best of both worlds. The active reactive component handles the rapid dynamics of the environment, while the deliberative component plans the overall behavior. Such a setup has been used in autonomous vehicle navigation [121], where the reactive controller handles low-level issues like staying on the road and avoiding unforeseen obstacles, while the planner sets the overall trajectory to reach the goal location. Separately, the two components may not be able to accomplish the task efficiently, or at all. For example, the reactive controller alone might get stuck in cul-de-sacs and be unable to navigate around them, while the deliberative component would be unable to adjust to a deer darting across the road.

2.2.2 Policy Model

Separate from the instantiation of the controller is the computational model used to describe it. As an example of the distinction between the two, consider a controller for a mobile manipulation task. The same controller can be implemented in multiple ways, in, for instance, a subsumption, three-layer, or behavior based fashion. If the world is sufficiently static, a deliberative architecture could be used as well. However, all the different possible implementations would behave the same and can furthermore all be described, abstractly, using the same model, such as a decision tree.

Models are then in some sense equivalent, and the choice of the particular one used for a given controller can be made to ease analysis or implementation, or highlight certain aspects of the control policy. We will examine two models here: Markov chains which directly map states of the world to actions, and finite state machines, which break the overall task into subtasks. In our discussion of how the issue of multimap control policies manifests itself in each, we will take $\hat{\mathcal{S}}$ to be the set of states that our robot can detect, and $\mathcal{A}$ to be the actions it can perform. The evolution of the world is given by $\mathcal{P} = P(\hat{s}_{t+1}|a_t, \hat{s}_t, ..., a_0, \hat{s}_0)$, the probability of the robot observing $\hat{s}_{t+1}$ given its complete history of perceptions and actions. The corresponding robot control policy is then $\pi = P(a_{t+1}|\hat{s}_{t+1}, a_t, \hat{s}_t, ..., a_0, \hat{s}_0)$, the distribution over actions to perform. For generality we have described the probabilistic case where full history matters, but other models, such as one with a completely deterministic $\mathcal{P}$ and π, could be used as well.

Markov Chain

A Markov Chain (MC) simplifies our above model by making the Markov assumption, that all necessary information about the world is contained in the current state. This assumption is equivalent to saying that history is not relevant, or that $\mathcal{P} = P(\hat{s}_{t+1}|a_t, \hat{s}_t)$ and $\pi = P(a_t|\hat{s}_t)$. A policy then only considers the current state when choosing an action, and for discrete state and action spaces can be stored as a table, of size $|\hat{\mathcal{S}}| \times |\mathcal{A}|$, assuming that enough memory is available. If memory is insufficient to store π, or if states and/or actions are continuous, other representations such as approximate functions may be necessary [133].

Often, Markov chain controllers are the result of learning in a Markov Decision Process (MDP), where in addition to the states, actions, and transition function, there is also a reward mapping that associates scalar rewards with every state of the world, or state-action pair. Learning the final control policy involves finding the policy that maximizes reward, and is thus an example of reward-based learning as will be discussed in Section 2.3.1. In keeping with the literature, we will discuss variations in MC controllers in terms of the MDP variants from which they derive.

The basic MDP and MC models as described deals with noise in the performance of an action by having a distribution over the next state of the world. That is, if the robot executes a walk forward action, the chance that it might actually move sideways, or not at all, is reflected in P. Likewise, noise in the perception that leads it to *decide* to walk is handled by having a probabilistic policy. For a given state, the possibility that the robot is actually in another state and should perform a different action is captured in π. This distribution also deals with situations where, in a given state,

the action that should be performed is uncertain. These cases are examples of perceptual aliasing.

Another approach to uncertainty in sensors, and one that directly considers that states may be perceptually aliased, is that of Partially Observable MDPs (POMDPs) [71]. This abstraction considers that there may be differences between what the robot perceives, $\hat{s}_t$, and the true state of the world, s_t. This difference is modeled with an observation function that maps perceptions to distributions (or beliefs) over true states, $O(\hat{s}) \rightarrow b = P(s)$. Decisions are then made with respect to b, instead of $\hat{s}$. To do so, the transition function $\mathcal{P}$ must be over true states as well, so the system can propagate its belief forward. When two true states require different actions, but are both perceived the same, the observed mapping between perception and action would be a multimap.

Consider a robot in a room with a box, where a human enters and places an object in the box, and once the human leaves the robot needs to perform some action depending on the box's contents. In the MDP model, the robot cannot perceive the contents of the box, and a MC policy from perception to actuation would have to have the same distribution no matter what the contents. Even though the robot watched the box being filled, once the box was closed the robot would not be able to determine an appropriate action. Using a POMDP, however, the robot could maintain a belief about what was in the box, and act accordingly. Another way of saying this is that the robot explicitly tracks the hidden state, corresponding to what is in the box. To do so, it needs to know that this information exists, and that it is relevant to the task. A similar effect could be achieved by expanding the robot's perception space to include previous observations, without knowing which parts will be important. Instead of making the first order Markov assumption, this approach makes an N-order Markov assumption, where N is the amount of history that should be considered. An issue that then must be addressed is choosing N.

Another technique that can be used to deal with multimaps is to allow actions to be extended in time [46]. One approach, termed semi-Markov, provides a set of *options* ($\mathcal{O}$), in addition to the primitive actions ($\mathcal{A}$) [135]. Originally designed to allow for the simplification of high-level control, each option has associated with it its own policy, which takes over control of the robot when that option is active. For example, a "go to red" option would take control of the robot once activated, and guide it towards red objects, while a "run from red" option would do the opposite. Both options could be valid for the same perceptions, but give different outputs. Thus, the control policy for a given state, s_t, may be different depending on if an option was selected in the past and is still active.

Again, consider our robot and the box. With a semiMC policy, the robot would activate different options when it saw the box being filled. These options would be time-extended, and continue to generate appropriate actions for the robot until they terminated. Thus, when the human left the room, a content-appropriate action would be performed.

While MCs are closely related to reactive controllers, in that they intrinsically frame the robot control problem as a direct connection between what the robot perceives and how it acts, semiMCs can be seen as a move to a hierarchical model. Particularly, the overall policy is now broken into subsections, or options, which may be mutually applicable in some states. The complete controller may then lend itself more to being implemented in a behavior based fashion, where each behavior

is an option and the combiner chooses which should be active.

Finite State Machines

Another hierarchical way of thinking about robot controllers is to model them as Finite State Machines (FSMs). An FSM can be viewed as a set of K subtasks, with a separate policy for each one. On top of these policies is a transition matrix that specifies when the active subtask changes. When a particular subtask (k) is active, its policy (π_k) determines the robot's activity, until the system transitions to a new subtask [106].

Consider our robot soccer goal-scorer in Figure 1.8, it is an FSM with 4 subtasks: Seek, Trap, Aim, and Kick. The seek subtask, for example, has a policy that drives the robot to locate and approach the ball. Once the robot is close enough to the ball, the system transitions into the trap subtask. Additionally, multiple subtasks' policies may have different actions associated with the same perception. For the two mentioned subtasks, this situation occurs when the ball is in the area in which it can be trapped. The seek controller, if active, would attempt to control the robot to position it better with respect to the ball, while the trap controller would move the head to initiate the trap motion. This situation is then a multimap in the overall policy.

In this respect, FSMs at this level are similar to semiMCs. The FSM's subtasks are analogous to the semiMC's options, and the top-level MC would define the appropriate transition matrix, determining when each subtask should be active. Both abstractions similarly deal with perceptual aliasing in the present by having different policies, chosen in the past, controlling the robot. However, both approaches raise questions about how the subpolicies are derived. That is, how are the semiMC's options or the FSM's subtasks chosen? Additionally, how many are there? A difference between the two models is that in a semiMC, the options are thought of as being primitives, on which the control policy, the MC, is built. In an FSM, the subtasks are part of the policy itself. These two possible interpretations parallel to the two ways in which direct perceptual-actuation couplings can be viewed: as part of the architecture (semiMC options), or part of the policy (FSM subtasks).

Taking the parallel between FSMs and MCs further, if each of the underlying world states has its own subtask ($K = |\mathcal{S}|$), then the FSM becomes an MC. Essentially, each subtask's policy's output is the output from the associated MC's policy ($\pi_s = \pi(s)$). The FSM transition matrix would then be equivalent to the transition matrix of the underlying state space, $\mathcal{P}$.

2.2.3 Mapping

The approaches discussed so far represent various methods of *instantiating* and *designing* a robot control policy, and it is often the case that the same policy can be described and developed in multiple ways. At a higher level, when viewing the control policy from the outside, the exact specifics of how the computation gets done are somewhat irrelevant. Put bluntly, when observing only the behavior of the robot, the techniques used to control it may be indistinguishable. No matter what style of processing goes into determining what is done, all that is observed is that in situation $\hat{s}$, the robot performs action a.

Taking this view, we argue that at its most abstract, a robot's control policy is just a mapping, from perceptual inputs to actuation outputs. Applying this concept to HRPT, what we are seeking is then a method transferring this overall mapping, π, specifying what the robot should do in each state of the world that it can detect, from a human. The exact form and model used to describe the resulting policy on the robot may thus be different from the one used by the user.

2.3 Learning Control Policies

Rather than designing and instantiating a control policy by hand, we are instead interested in learning the policy from collected data. We here consider two possible methods for generating the data used. One, reward-based learning or learning from guidance, is to associate a reward, or cost, with each of the robot's decisions. Learing is then focused on determining a policy that maximizes reward or minimizes cost. In the following, we will only consider maximizing reward, as all of the approaches are analogously defined for minimizing cost. An alternate approach is learning from demonstration, where examples of the task policy itself are collected, perhaps as trajectories through the robot's perception and action space. In this case, learning attempts to estimate (either explicitly or implicitly) what the goal of the task is, and perform appropriately in novel situations.

Recall that both types of learning are learning a policy that maps perceptual features to actions. An issue that must be considered is then that of feature selection, or choosing which features to learn on [146, 62]. This issue is also related to that of hidden state, in that if the appropriate features are not selected, necessary state may be hidden from the robot. The combination of features and learning algorithm can significantly ease or complicate learning. In general, with sufficient perception and actuation processes, simple learning algorithms such as linear or nearest-neighbor regression may suffice [122]. However, the needed high-level features and motion primitives (such as object identities and appendage trajectories) may be too closely tied a particular behavior, preventing the robot from learning other tasks. In contrast, lower-level features and actions such color histograms and generic grasps may allow for more varied tasks to be learnt, but often have more complicated (e.g. nonlinear) relationships to the control outputs.

2.3.1 Reward Based Learning

In traditional reinforcement learning, data takes the form of scalar rewards coupled to each state of the robot, or perhaps state-action pairs. During training, the robot experiences different states and receives the associated reward. The goal of learning is then to develop a policy that maximizes the total (discounted) reward [45]. Given a reward function ($\mathcal{R}(\hat{\mathbf{s}}) \rightarrow r$), we can state the value of a policy π as

$$V^{\pi} = E\left[\sum_{i}^{N} \gamma^{i} \mathcal{R}(\hat{\mathbf{s}}_n)\right]$$

where γ is a discount factor and the N $\hat{\mathbf{s}}$ are obtained by following π. The π with the highest value is the optimal policy, and can be used to control the robot.

One of the arguments for reward-based learning is that the reward function may be a more compact representation of the robot's policy than the actual mapping itself. As such, it may be easier to specify for a human user. While this fact may be true for states that are clear goals (like checkmate in a game of chess), other states (particular board positions) are not as easily assigned scalar values. Doing so is related to the credit assignment problem [21], where it is not clear which decisions in the past are directly responsible for the current situation.

Assuming that a reward function is given, one approach to learning is *value iteration*, where the value function is maximized iteratively, via the Bellman update equations (or temporal difference learning). Techniques for value iteration exist in both discrete [125] and continuous [107] domains. If the full value function cannot be stored (as, for example, a discrete table), approaches exist to approximate it using, for example, proto-functions [83] or a Radial Basis network [134]. An alternative model is to consider the Q-function, which maps states and actions to expected reward ($\mathcal{Q}(S \times A) \rightarrow r$), instead of considering the policy as a whole. The optimal policy can then be derived by choosing the action with the highest expected reward at each state. The Q function itself can be learned in continuous spaces as well [90].

Rather than learning one of these secondary functions [133], the policy itself can be learnt directly. Generally termed *policy iteration* techniques, these approaches formulate a complete policy, test it, and then change the policies parameters to improve behavior, as determined by accumulated reward. One technique is to use genetic algorithms [139], where simulated evolutionary processes "mate" and "mutate" good policies in the hopes of achieving better ones. If the policy's value function is derivable, gradient ascent is another choice [130].

While function approximation techniques allow RL to be applied to the continuous domain, as the state and action spaces of the robot continue to grow, these approaches themselves become intractable. Hierarchical RL represents an attempt to deal with these growing spaces by decomposing the overall task into smaller, more easily learnable ones [15]. Doing so directly parallels the shift from MDPs to semiMDPs, where a top-level MDP can have as actions complete policies in lower-level ones. Often, the decomposition of the overall task into subtasks is given, and the same learning techniques are applied at all levels [101].

An issue that arises in RL is that of exploration versus exploitation [85]. That is, once an agent has a good policy, it must decide whether to *exploit* this policy for good performance, or *explore* alternate actions in the hopes of finding an even better policy. Basic ϵ-greedy approaches take a fixed percentage of random actions, to always explore other possibilities [56]. However, this choice can lead to degraded overall performance, since the final learned policy needs to account for the possible catastrophic effects of these random actions. Alternatively, *simulated annealing* decreases the percentage of exploratory actions over time, but ends at a policy that has none. Additionally, as the state and action spaces grow, the probability of finding high-reward areas by random exploration decreases. This problem is one of the aspects of the *curse of dimensionality*.

2.3.2 Demonstration Based Learning

In contrast to learning from collected rewards, control policies can instead be learnt from demonstrations of the control policy itself. In this approach, data takes the form of matched $(\hat{\mathbf{s}}, \mathbf{a})$ tuples, with the added assumption that $\mathbf{a}$ represents an appropriate action to perform when $\hat{\mathbf{s}}$ is perceived. This assumption serves to bootstrap the exploration problem discussed above, as the robot is directly provided with examples from desired (high-reward) regions of the state-action space.

Inverse Reinforcement Learning (IRL) techniques perform learning from demonstration by first transforming the problem into a reward-based learning one. They use the data to infer an underlying reward function that is latently guiding the demonstrator, and proceed as above [111]. One benefit of this approach is that differences between users can be framed as differences in how they reward the various features of the space. By learning a user-specific reward function, different demonstration styles can be mimicked. However, estimating the reward function itself is a regression problem, but the actual target values are hidden. To constrain the search, $\mathcal{R}$ is often taken to be linear with respect to the state, leading to issues of feature selection. Further, the state variables themselves define a control basis, where each variable has a behavior that attempts to maximize it, thus further limiting the space of performable actions. Alternate, nonparametric, techniques may be able to learn more general reward functions, although we know of no work in this area.

Rather than first inferring a reward function and then attempting to maximize reward, we can approximate the policy, as a mapping, directly. That is, we can view the policy as $\pi(\hat{\mathbf{s}}) \rightarrow \mathbf{a}$ and attempt to derive an approximation $\hat{\pi}$ straight from the data. Such approaches are termed Direct Policy Approximation (DPA) or Direct Policy Learing (DPL) [118].

Similar to RL, if the form of the policy is known, parametric regression can be used to fit the parameters [13]. In this case the goal is to minimize the difference between what is predicted by the approximator and what is commanded by the demonstrator, instead of maximizing an external reward function. In this respect, LfD approaches can be seen as having an intrinsic reward function that rewards behaving like the demonstrator.

If the form of the policy is not known, nonparametric algorithms can expand the parameter space as needed [66]. However, as above, as the state and action spaces grow and the policies become more intricate, a hierarchical decomposition of the task can prove useful, especially if there is hidden state. This switch is akin to changing from a single reactive controller to a behavior based one, where the subtasks are portions of the overall task or lower-level behaviors. For instance, the lower level policies can be basic navigation and manipulation capabilities, and a higher-level policy is responsible for modulating or sequencing them. By "gating" data to the different policies, each can be learned in isolation, as is done with neural networks in [138]. Similar to hierarchical reinforcement learning, the decomposition is taken as known.

This approach can also be viewed as an FSM, where the lower-level policies are the subtasks. If they are given, learning a complete policy directly from demonstration can be done by learning the appropriate sequence of subtasks to perform. To do so, each subtask can monitor its activity during demonstration and learn pre- and post-conditions for transitioning to others [96]. Alternatively, it

may be necessary to fuse the policies from multiple subtasks, again according to their applicability [97]. By further tracking the applicability of each subtask during autonomous execution, [99] learns a rough topological map of the environment, similar in spirit to [59]. These maps can then be used to address hidden state by incorporating a planner, moving more towards a 3LA.

We can also view a hierarchical decomposition of a task into subtasks as a mixture of experts model [68], where each subtask is an expert in its domain, and the full policy is formed by mixing, or gating, their outputs. If the domains of the experts overlap, a multimap control policy results, as hidden state is required to choose the appropriate expert. The approaches described above take the experts as known, and learn the mixing parameters. If, however, the subtask decomposition is *not* known, it must be estimated from the data. This estimation gets at the heart of multimap regression, where the presence of a multimap scenario must be differentiated from noise. Rather than taking the number of subtasks as fixed, and learning their domains of expertise, we can instead consider a possibly infinite number of components and learn the number of active ones from data [113]. In Markov Chain terms, this approach is an infinite hidden Markov model [18]. We combine this approach with individual subtask policy learning to address both model selection and policy learning at the same time [112], given only demonstration data.

2.3.3 Regression

Our specific approach to learning from demonstration with direct policy approximation is a form of regression. Regression, generally, is the estimation of a mapping $f(x) = y$ from known input-output pairs $\{x_i, y_i\}, i \in 1 : N$, where x is termed the independent, and y the dependent variable. For us, x is the state of the world, and y is the robot's action.

Usually, x and y are continuous variables, as classification techniques may be more appropriate if they are discrete. In regression, techniques can be divided into two groups: Parametric approaches, which take as given the form of the target mapping, and nonparametric ones, which do not.

A common parametric technique is to fit a known polynomial form to the data, such that $y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + ... + a_d x^d$ up to the order of the desired polynomial, d. If $d = 1$, linear regression is achieved. The fit can often be performed by *Least Squares* regression, or minimizing the squared error between predicted and known outputs. This method is equivalent to assuming that the observed data comes from the model

$$y_i = f(x) + \mathcal{N}(0, \sigma^2)$$

where observed outputs are distributed in a Gaussian fashion around the true target.

Nonparametric approaches, on the other hand, do not take as known the form of the mapping. That is not to say they do not have parameters, instead nonparametric methods can be thought of as those where the data itself are the parameters, or the parameterization grows with the data. One example would be if the degree of the polynomial model above were to grow with the data, such that $d = N$. Nonparametric models can thus grow in complexity with the data, but also run the risk of overfitting.

Given that our data is continuous and the mapping is of unknown form, we consider nonparametric regression. Of course, as the parameterization can grow with the data, an algorithm may require unlimited data storage. We must then focus on approaches that explicitly limit the growth of the parameterization, or *sparse* nonparametric regressors, that do not require all previous data to make a prediction. Even within this subset of techniques, there are many possible methods for learning $\hat{\pi}$ in an interactive, scalable, robust approach.

We initially considered Locally Weighted Projection Regression (LWPR) [149], which has been previously used for robot learning. LWPR is a local approximator, in that it fits the overall mapping by dividing the input space into multiple, overlapping regions called receptive fields (RF). Each RF performs a local projection and linear regression on the data in that region, and predictions from multiple RFs can be combined to form the total system's output. LWPR is sparse in that only the sufficient statistics for the RFs need to be kept, so that once a datapoint has been incorporated, it can be discarded. Incorporation of new data (inference) is incremental, through the use of partial least squares regression in each RF, and an incremental update to the area of the RFs themselves. LWPR has the added benefit of explicitly considering that there may be unnecessary dimensions in the data, and seeks to project them out.

Other possible nonparametric regression algorithms that have been used for learning robot control include K-Nearest Neighbors [128], Neural Nets [138], and Gaussian Mixture Regression [30]. Herein we will use the global approximator Sparse Online Gaussian Processes (SOGP) [41]. It is a form of kernel regression, where each datapoint has influence over an area of space near it, ensuring that data close by in input space have similar outputs.

Separate from the issue of parametric versus nonparametric regression is that of unimap versus multimap regression. All of the approaches discussed above are unimap regressors, where it is assumed that there is *one* correct output for a given input, which is observed in a noise-corrupted form. We can achieve multimap regression by considering multiple overlapping unimaps, which is again a mixture of experts model [68]. Also again, both parametric and nonparametric approaches to multimaps exist. That is, the number of possible outputs for a given input can be taken as known a priori, or also inferred from the data.

In order to be as general as possible, we apply nonparametric multimap regression, where the number of possible outputs, form of the individual unimaps, and the unimaps themselves are all estimated from the data. A drawback of being nonparametric in this way is that more data is required to perform the additional estimation. Further, as overfitting is a very serious concern, data is needed not only for training the model, but additional data is needed for testing. As we are learning from human demonstation, these data must come from the demonstrators.

2.4 Human-Based Data Collection

As our work involves the teaching of robots tasks by human demonstrators, it necessarily incorporates several aspects of Human-Robot Interaction (HRI). Specifically, from the field of HRI we draw

techniques to obtain the needed data from human controllers. In doing so, our human-robot interface must balance two competing goals. First, we wish the transferred policy to perform as well as possible. Secondly, we wish to make the interaction usable for the human, meaning that performing HRPT should minimize excessive training, cognitive load, or time. In general, however, the more time spent on HRPT, the better the resulting policy will perform. Balancing these two goals depends on finding a point where policy correctness and user ease are in equilibrium.

Both policy correctness and user ease can be calculated in multiple fashions, quantitatively and qualitatively. What we are truly interested in is the tradeoff between the two, sometimes called *Return on Investment*. ROI can be thought of as the ratio of policy correctness to user effort, with appropriate units. As an example, a robot controller that performed at 90% of optimal after 9 hours of user time would have a lower ROI than a controller that performed at 80% optimality after only 4 hours of user time, if the hours spent were equivalent. By switching to LfD from explicit coding, we seek to increase the ROI for non-programmers performing HRPT, although the exact policies instantiated may not be as good.

While we are interested in LfD, specifically from human teleoperative control, this same tradeoff must be considered in RL. For example, the reinforcement signals themselves may be ambiguous [155], and getting more specific ones may require a more involved interface, leading to increased user cognition. Specifically, humans may not give reinforcement only when good behavior is observed, but also when it is anticipated [141], leading to the two types having to be disambiguated. Further, collecting sufficient reinforcement data for learning may require large amounts of interaction [72].

2.4.1 Control Interface

When teleoperating a robot, a human user provides control signals that drive the robot to perform the intended task. At the simplest, the user may control every actuator individually, but the interface may also allow for higher-level control. For instance, the user may have access to provided or learned reflexes, as discussed in Section 2.2. Alternatively, the subtasks or action options as in Section 2.2.1 may be known, and the user needs only select one to be active, or indicate parameters such as waypoints for them. Or, given sufficiently high-level autonomous subtasks, the user may only have to indicate transitions between them [40, 63]. In this limited case, a user's attention could be split between multiple robots [4], but it is still the human's responsibility to set each robot's behavior in a manner consistent with the task at hand.

No matter what the level of control, the user must have access to portions of the robot's state, or sensory experience, as part of the interface. This information is necessary to enable the user to make any decisions at all. However, the way this information is presented is very important, as it impacts the user's ability to control the robot [91]. In fact, presenting the same information to the user in different ways can often effect task performance [98]. Consider two representations of speech data, one as an image of the waveform on the screen, and another as rendered through a speaker. While both contain the same information, untrained users may require more time to extract that information from the graphical representation. Poor design of displays and controls, which require

more cognitive effort on the part of the user to use, has been linked to severe errors in the past [100].

Complementary to the display of robot perception is the extraction of control signals from the user via some sort of control device. Graphical user interfaces (GUIs) are one such approach, where the user points and clicks, and possibly types, to give commands to the robot. Research in assistive and rehabilitative robotics has lead to a wide variety of interfaces that may be used for users of different skills and cognitive capabilities [145]. Future control interfaces may automatically adapt themselves for display on different devices, for different users' needs [50]. Alternatively, brain-machine interfaces, may enable users to control robots without any physical movement at all [147].

As all interaction between the robot and the human is mediated by the teleoperative interface, the two need not be co-located [104]. Indeed, the display of perception and gathering of actuation data may be distributed over multiple locations, and possible multiple users [54]. Using a distributed data gathering framework over the internet may be one way to simplify data collection for teaching robots from demonstration. That is, users who would not usually have access to a robot could generate data from afar, leading to larger and more varied data sets.

2.4.2 Transparency

While providing appropriate actions for extracted perception data, users can improve the quality of their teaching by leveraging information about the learning process itself. Called *transparency*, the availability of this knowledge can enable users to understand which portions of the task have been successfully learned, and which have not. Users can then focus their teaching efforts appropriately [141]. In the context of training a robot by demonstration, transparency can be as simple as just observing the robot's behavior, and targeting more demonstrations where it is incorrect.

A training scenario thus usually becomes a series of train-test phases. The expert first demonstrates the task or portion thereof, the robot performs learning, and then the robot uses the learned policy to act. The user then observes the robot's behavior, and when appropriate (after the task is finished or an error occurs), generates a new demonstration. This cycle is repeated as necessary.

Often, however, just observing the failure case of the robot is not enough, as the reasons for the failure are not clear. Multiple repeat demonstrations may then be necessary, until the user generates data that addresses the underlying error. In this situation, more specific feedback from the robot can help shape the demonstrations that the human provides [102]. One form this feedback can take is a measure of confidence, of how reliable the system believes its learned autonomy to be in the current situation. Using this information, the user can select future demonstrations specifically to target low confidence areas. Note, that confidence does not necessarily mirror poor task performance, as the robot could do the wrong action confidently, or the correct action with low confidence. Thus, confidence and task error are complementary information channels. Confidence can further be used to enable the learning system to be active, or ask for more demonstration from the user, when confidence is low.

We have so far discussed feedback from the robot to the user, with the aim of enabling the user to generate better demonstration data. There is also the opposite direction of information flow, where

the user gives feedback to the learning system to enable it to make better use of the data it has. One method is to enable the user to provide high-level feedback, or critiques, to the robot instead of more demonstrations [7]. That is, after observing robot behavior, the user indicates which portions of the task were performed well with the current policy, and which portions should be improved. Learning can then focus on the appropriate areas.

A similar technique allows users to modify the data after they generate it [8]. That is, the learned system is given *advice*, such as "go faster here," and the underlying data from which the policy is learned is changed to reflect that fact. This method is especially applicable in domains where the robot is potentially a better performer of the task than the human demonstrator. In this respect, feedback from humans to robots is similar to reinforcement learning.

One last way in which bidirectional feedback between human teachers and robot learners can be leveraged is in situations of *scaffolding* [116]. In scaffolding, a learner is initially trained on an easier version or portion of the goal task, and then incrementally introduced to the desired overall behavior, with a goal of decreasing the total learning time with respect to learning the complete behavior all at once. Feedback from the robot can help the user determine an appropriate scaffolding order, as portions that are hard to learn could be simplified and taught separately. Similarly, feedback to the robot could be used to indicate where previously learned, scaffolded subportions could be applied.

2.4.3 Tutelage

As introduced in Section 1.3.3, robot tutelage arises when learning is suitably fast and interleaved with evaluation in an interactive teaching paradigm [67]. Feedback in both directions can also be incorporated. The main draw of tutelage is that it may free the demonstrator from the need to produce multiple demonstrations of the entire task ahead of time, as only the areas of the policy that are learned poorly would need to be redemonstrated. A hierarchical approach is also possible, where after the overall task is learned, individual low-level modifications are introduced [75].

Outside of robotics, tutelage has been used to adapt other learned systems to users in real time. For example, [132] describes an email program that will automatically generate folders and sort mail for a user. If a user is unhappy with the resulting organization, they provide counterexamples (demonstrations) of how they would prefer it, and the learned system is retrained. Because the system is retrained after each example, often only a few demonstrations are needed from the user.

More generally, programming by demonstration is a paradigm that has been used in attempts to simplify routine tasks for general computer users. By keeping a trace of a user's activities, repetitive tasks can be extracted and formed into macros. The version space algebra is one approach, that forms macros in the background as users behave normally, and offers to complete common tasks when they are detected [80]. If the completion is incorrect, the user can override it, and the learned macro is edited with the new information.

Mixed-Initiative Control

As our demonstrations come from teleoperation, we must deal with the potential for conflicting commands from the learned autonomy and demonstrator. Particularly, if both sources are generating control outputs, we must determine the actual actions performed by the physical robot. Our solution uses confidence measures to arbitrate between the two, enabling mixed-initiative control, where the controller with less confidence gives control to, or has control taken away by, the other.

This approach is not the only way in which a user and autonomous system can interact. Rather than giving control of the robot fully to one or the other, they could share it. Adjustable Autonomy [40] is one such approach, where the user can decide to take over partial control of the robot, making it less autonomous. They can then guide the robot to perform a particular aspect of the task, with which it may have had trouble, while the autonomy continues to run other portions of the overall behavior. Take as an example the robot task of walking in a circle while tracking a ball. Our MIC approach requires that both walking and tracking be demonstrated simultaneously. An adjustable approach might let autonomy control the legs while the demonstrator used the head, or vice versa.

With MIC, we are also concerned with the user suddenly being thrust into control of the robot, if the autonomy has low confidence. At issue is that the user may not be paying attention, or have their attention focused elsewhere, or on another task. They then may lack *situation awareness* [28] of the robot and its environment. By the time they acquire it and are able to provide appropriate control, the environment may have changed, so that instruction is no longer needed.

Along the same lines, by having control of the robot thrust upon them, whatever the user was attending to before that may suffer. Further, if this situation arises often, the system may be perceived as bothersome or overly interrupting [93]. Approaches that monitor a user's activities and determine appropriate times to request their attention may alleviate this issue, enabling robots to weigh the potential gain in policy performance against any annoyance that may befall the user.

2.5 Robot Learning

We are not the first to perform interactive robot learing from demonstration, nor are we the first to attempt to learn to play robot soccer. The Brainstormers [114], for example, used hierarchical reinforcement learning to learn a simulation Robocup team. Individual robot subtasks (which were given) were learned as MDPs, and the overall team policy was treated as a multi-agent MDP over these subtasks. A central value function, approximated by a neural net and representing the likelihood of scoring a goal in the current state, drives learning. Due to the RL framework, random policies are initially needed to explore the state and action space of the team to discover goals. Later work transitioned a learned policy for intercepting the ball from simulation onto a real robot [92].

More similar to our approach is [128], which uses memory-based learning of a continuous function to determine when a simulated robot should kick or pass. The policy itself is learned by storing memory of previous attempts, and using nearest-neighbor lookup to determine the action in novel states. The continuous function learned is the probability of scoring as a function of defender

location, the actions chosen are discrete, either pass or shoot.

We used a similar nearest-neighbor procedure in our 2006 DemonstraDogs team [81], but applied to real robots and learning continuous state to continuous action mappings. In that system we collected data representative of correct behavior in batch, and achieved sparsity (and thus realtime prediction) by selecting an appropriately sized random subset for storage. During play, a robot's current perception was matched with a KD-Tree to the nearest perception in memory, and the corresponding action performed. The DemonstraDogs team lost all of their games, in part because all the robots would walk off the field and end up penalized. Despite this behavior, at times the team showed a "glimmer" of intelligence, reacting the the ball.

2.5.1 Inverse Reinforcement Learning

Closer to our desired goal of intelligently learning unknown tasks from human user demonstration, recent work in Inverse Reinforcement Learning has successfully been used to learn to fly an aerobatic helicopter [1], drive in parking lots [2], and traverse uneven terrain with quadrupeds [75]. While the approach itself is thus proven to be applicable to multiple domains, a major limitation is that the features for each domain must be hand selected. In particular, as their system learns a reward function that is linear in a set of features, the features must therefore be appropriately high-level. Examples include a "measure of distance between a driving trajectory and the lane" in the parking lot task and "several features capturing the roughness and terrain slope at several different spatial scales around the robot's feet" in the quadruped task. IRL can also be performed with lower-level features, such as position and orientation in the helicopter task. Still, however, the control policy is linear in these features, leading to concerns about the space of possible policies covered by the resulting control basis.

The main difference between this work and our own is that we consider approximating the policy directly, instead of inferring an underlying reward function. By using RL, they leverage the claim that the reward function is often a more compact representation of the policy. Further, their use of RL allows them to learn to outperform the human user. However, their representation relies on high-level, task-specific features, of which they only consider linear combinations. We instead utilize nonlinear approximators, which may enable the same tasks to be learnt from lower-level features. Additionally, we explicitly address the issue of hidden state and perceptual aliasing without a known task decomposition, although work such as [87] and [70] may enable IRL to do the same by building the state space incrementally. Lastly, we desire to operate in an entirely interactive framework, while their system requires batch data collection.

2.5.2 Confidence Based Autonomy

The Confidence Based Autonomy (CBA) algorithm utilizes a notion of confidence to perform mixed-initiative robot learning from demonstration [36]. There they use a set of Gaussian Mixture Models (GMMs), one per discrete action, to divide the state space of the robot into regions where the different

actions are appropriate. The GMMs are updated incrementally, as data arrives, using the Akaike Information Criterion to discover the number of mixture components in each one. This criterion is a form of penalized log likelihood, where overfitting (by having more mixture components) is weighed against the gains in likelihood of the model. They have successfully shown multiple robots learning a collaborative task using this framework [35].

Confidence comes into play as the robot observes new data. If the current model confidently predicts the appropriate action, it is executed autonomously. Otherwise, the robot pauses and requests a new demonstration from the user. This behavior is similar to our default behavior, which intervenes when learner confidence is low to trigger active learning. CBA also allows for corrective demonstration, as we do, allowing the user to provide new data as desired. By utilizing different confidence thresholds in different areas of the state space, their technique can adapt to different distributions of data [34]. Similar to our work, they are interested in perceptual aliasing, and use confidence to detect areas of ambiguity, where multiple actions are possible, and the demonstrator is inconsistent. They then hypothesize a new option class that incorporates the multiple actions [33].

While this work is similar to ours in several respects (the use of confidence, tutelage and addressing perceptual aliasing), the major difference is that CBA operates in a classification domain. That is, a finite set of actions exists, and the perceptual space of the robot is divided into classification regions, where the label on a region indicates what action to perform. In contrast, we operate in a continuous action space, and thus use regression to approximate the mapping. Further, we approach ambiguity differently. CBA considers creating a new class that represents the union of two or more actions, and then selects an action uniformly from them at execution. We, by performing multimap regression, instead represent each action separately, and draw from the distribution over actions when needed. Lastly, our approaches to model selection differ. CBA divides the state space using a set of GMMs, where each action (the number of which is known a priori) has its own GMM with the number of components chosen with AIC. We also use a GMM to divide the state space, but consider only one with an infinite number of components.

2.5.3 Gaussian Mixture Regression

Also utilizing GMMs to perform incremental robot learning from demonstration is [29]. They, however, utilize Gaussian Mixture Regression (GMR), which is more similar to our approach then the classification-based CBA. They further explicitly consider dimensionality by projecting the data into a latent, lower-dimensional space (which includes time) using principal component analysis (PCA). In the reduced space they fit a GMM model to the data, using the Bayesian Information Criterion (BIC) to determine the number of components. Similar to the AIC, the BIC is a penalized log likelihood which discourages overfitting. As time is included in the state, executing an action corresponds to predicting the robot's pose at a given time in the action execution. To accomplish this, all demonstrations of a task are resampled to take the same amount of time. Using motion capture and kinesthetic teaching, this technique has been used to learn a series of free space movements.

For object-oriented movements, the method is extended to consider task as well as joint space

[30]. In addition to the robot's motors, features of the environment related to the task, such as the relative locations of particular objects, are included in the state space. By fitting a GMM to multiple demonstrations of the same task, the important features at each time in the task are detected by considering the variance of the demonstrator at that time.

This work departs from those above, and ours, in explicitly considering time. While we learn a reactive policy, mapping from the current state to a desired action, they learn a time-extended motion, which is a mapping from time in the task to a desired location in joint/task space. To do so they need to map all demonstrations of a task into the same length of time. Their approach is incremental in the sense that new demonstrations of the entire task can be incorporated to refine task performance. However, our approach is incremental in that the task itself can be demonstrated incrementally, or a portion at a time. For multi-state tasks, this ability may be advantageous.

2.6 Summary

An autonomous robot requires many parts. The physical embodiment, sensors, effectors, computational architecture and perceptual and actuation processes are all parts of the substrate on which the control policy operates. We take all of them as given and focus on the process whereby a human instantiates the robot's behavior. The control policy itself can be formulated and implemented in multiple fashions, but at its most abstract we view it as a mapping from robot perception to action. Using learning, we estimate policy mappings from demonstration data collected interactively through tutelage. We further employ mixed initiative control to share command of the physical robot between the learner and teleoperative demonstrator in support of the tutelage framework.

Chapter 3

Dogged Learning

> *Robot systems are now installed, debugged, and updated by trained specialists, who measure and prepare the workspace and tailor job- and site-specific control programs. Few jobs are large and static enough to warrant such time-consuming and expensive preparations. If mobile robots for delivery, cleaning, and inspection could be unpacked anywhere and simply trained by leading them once through their task, they would find thousands of times as many buyers.*
>
> Hans Moravec, Robot, 1999, page 92

Towards the goal of enabling non-specialists to more easily instantiate autonomous robot control policies, we present here our architecture for interactive robot lifelong learning of unknown tasks from demonstration with mixed initiative control [57]. Shown in overview in Figure 3.1 and algorithmically in Algorithm 3.1, our architecture is designed to work with different demonstrators, platforms (robots), learners and arbitration schemes. This flexibility has allowed us to directly compare the performace of different learning algorithms, and rapidly apply them to new robots. Further, as the approach allows for the interleaving of autonomous activity with training, it is appropriate to be

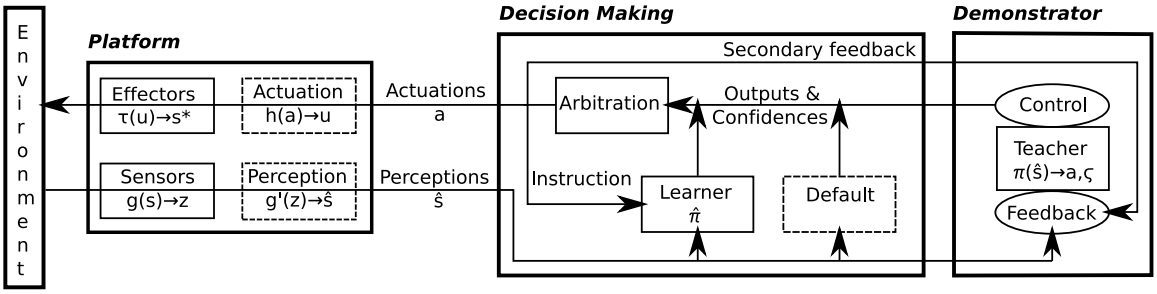


Figure 3.1: The Dogged Learning architecture. The platform interacts directly with the environment and extracts perceptions ($\hat{\mathbf{s}}$), an estimate of the true state of the world. Decision making generates an actuation ($\mathbf{a}$) by arbitrating between the demonstrator's policy (π), the learned approximation thereof (π), or a default controller, based on their confidences (ς).

Algorithm 3.1 The Dogged Learning Procedure

loop
  Receive perception ($\hat{\mathbf{s}}$) from platform
  Display perception to user {Feedback}
  Receive estimated action ($\mathbf{a}_L$) and confidence (ς_L) from learner ($\hat{\pi}$)
  Get commanded action ($\mathbf{a}_D$) and confidence (ς_D)from demonstrator (π) {Control}
  if $\mathbf{a}_D$ is NULL **then** {No input from demonstrator}
    $\varsigma_D \leftarrow 0$
  Arbitrate using confidences (ς_L, ς_D) and select action ($\mathbf{a}$) and confidence (ς)
  if $\varsigma < \tau$, the default limit **then**
    Get $\mathbf{a}$ from default controller
    $\varsigma \leftarrow \tau$
  if $\mathbf{a} \neq \mathbf{a}_L$ **then**
    Update learner with $(\hat{\mathbf{s}}, \mathbf{a})$
  Send $\mathbf{a}$ to platform
  Display results of arbitration to user {Secondary Feedback}

run over the entire lifetime of the robot, to enable *lifelong learning* [143]. As after a task has been learned satisfactorily, the user simply stops teaching and the learned autonomy takes over, there is always the potential for the user to resume teaching and modify the robot's behavior in new ways.

Conceptually, DL defines the way in which the tutelage process arises from the interaction between four entities: the environment, the platform, the autonomous decision making and the demonstrator, paralleling the robot control loop in Figure 2.1. The demonstrator is new, and serves as the source of the latent control policy that trains the robot's autonomy. Information flows from the environment through the platform, where it is processed and then presented to decision making. Decision making generates a response, possibly calling upon the demonstrator to do so, which is passed back through the platform, processed, and emitted into the environment, where it causes changes that continue the cycle.

The architecture itself is not tied to any particular system, and can be implemented in many different ways. As we focus on real, physical robots, our environment of choice is the real world, and thus no computational effort is needed to instantiate it. We now discuss the other constituent entities in more detail, and then describe our experiences with various implementations.

3.1 Platform

The platform represents a robot, or more generally an agent, embedded in the environment. We abstract away the low-level details of platforms, such as physical structure and computational architecture, and take as given fixed preception and actuation capabilities. That is, we assume the platform is able to extract an estimate of the world's state from its sensors and turn desired actions into effector activity. Specifically, given a world state $\mathbf{s}$ and sensors that extract information $\mathbf{z} = g(\mathbf{s})$ from it, the platform is responsible for producing $\hat{\mathbf{s}} = g'(\mathbf{z})$ containing, ideally, all the information necessary for the task at hand. Note that sensing and perception are not assumed to be inverse

actions, so that $\hat{\mathbf{s}} = g'(g(\mathbf{s})) \neq \mathbf{s}$. It is this difference between what the true state of the world is and what is extracted from the robot's sensors, perhaps due to noise or insufficient sensing, that gives rise to the form of perceptual aliasing with which we are concerned.

After a proper action response ($\mathbf{a}$) has been selected by decision making, the platform is then responsible for turning it into control signals $\mathbf{u} = h(\mathbf{a})$. The actuators themselves then cause the environment to change to $\mathbf{s}^* = \tau(\mathbf{u})$. Like with sensing, noise can result in a difference between the action commanded and the resultant change. Further, physical limitations of the platform can render portions of the environment unchangeable. For instance, a short robot may not be able to reach a high shelf. In Figure 3.1, perception and actuation are shown dashed, to indicate that they are not required. Indeed, the perceptual and actuation mappings may be identities, and learning can take place directly on sensor data and actuator control signals, as discussed in Section 2.2.

The dimensionality of $\hat{\mathbf{s}}$ and $\mathbf{a}$ are platform dependent, and can in principle vary over time. For example, a vision system could extract the coordinates of all visible faces, so the resulting perception vector would have to be different lengths depending on how many faces were detected. We simplify somewhat and assume fixed dimensionality for a given platform, although future development may allow for variable sized-vectors. Current systems can approximate variable lengths by assuming a maximum number and returning a null value as necessary. However, care must be taken when defining the null value, so that it cannot be confused with a valid one.

Further, we assume that dimensions are unordered yet non-exchangeable. Unordered refers to the property that there is no information in the order in which the perceptions and actuations are stored; the dimensions can be randomly permuted to no ill effect. Alternatively, the ordering could indicate information such as relative importance. Non-exchangability means that no two dimensions are equivalent, and thus dimensions cannot be swapped. Practically, these assumptions mean that we take the order of the dimensions to be fixed, but do not use the ordering when making decisions.

For generality, we allow both discrete[1] and continuous values for perception and actuation dimensions. However, to simplify learning somewhat, we require scaling parameters, representative of the maximum and minimum values for each dimension. As all dimensions are tied to some physical process with finite limits (in the robot itself), these values are often easy to obtain.

It is important that in the DL architecture, platforms are defined not by their physical characteristics, but by their perception and actuation spaces. This fact means that the same physical robot with different perceptual or actuation capabilities is treated as a different platform. On the other hand, if two different robots have identical spaces, they are in effect the same. We could, for instance, provide a legged humanoid and a wheeled trash-can robot with identical object detection and location control systems. Then, as their perception and actuation spaces match, policies learnt on one could be directly used on the other.

[1] Either multi-valued or binary one-of-n encoding.

3.2 Demonstrator

The platform subsystem of Dogged Learning encapsulates all interaction with the actual robot, and through it the environment. Similarly, the demonstrator subsystem is responsible for interacting with the expert, or latent control policy. Recall that we aim for our robots to learn from human demonstration of a desired task, but choose to not address the issue of mapping observed human poses and behaviors onto a robot's capabilities. Thus, instead of allowing a human to free-form demonstrate in the world, we require them to physically guide the robot though the task. Users usually perform this guidance using teleoperation, although we have also experimented with kinesthetically guiding the robot as well.

Likewise, we seek to avoid issues of perspective matching by only providing the user with the state information extracted by the robot's perception routines. By requiring that the human be able to perform the task in this fashion we ensure that the task is, in principle, decidable using only the information present in the robot's perception and performable using only the robot's actuation. However, we as of yet know of no way to limit the user's use of higher cognitive functions, such as memory. The use of these abilities gives rise to hidden state, information not available to the robot. The presentation of the perceptions and extraction of actuations to and from the user are dependent both upon the platform and the interaction devices used. A platform developer must therefore, in addition to developing the platform module described above, also develop a user interface. We separate the UI into two components, that of Feedback (presenting the perception data to the user) and that of Control (getting actuation data from the user).

3.2.1 Feedback

The primary purpose of feedback is to limit the user's knowledge of the robot's state to that known to the robot itself. Even then, displaying the information in a manner that the user can rapidly interpret and respond to is a non-trivial problem. As described in Section 2.4.1, the same information can be presented in multiple fashions, effecting the ease with which the user determines their response.

Generally, all of our platforms use a computer screen for feedback, although we have experimented with other modalities such as audio and tactile vibration. As we are not UI designers, we attempt to follow best practices [86], but our resulting displays are by no means the best possible. While the resulting perception displays may be robot dependent, often similar techniques, such as general layout of the display and certain imagery, can be used for multiple platforms. Further, as the data is represented abstractly, there is the possibility that the same display can be used for all platforms, assuming no platform-specific information is in the display. An example of a general feedback display is a text stream, where all perceptions are displayed as space-deliminated floating point numbers. While general, this particular display is often hard for humans to utilize for robot control.

Secondary Feedback

To enable users to make use of information from the decision making component itself, we allow for a secondary feedback channel. One item of interest may be the result of arbitration, described in Section 3.3.3, which indicates which proposed actuation, or combination thereof, is actually being sent to the platform. Additional information from the learner, such as its confidence, could also be used to guide demonstration.

While algorithm-specific information could be passed, we instead use the secondary feedback channel to convey the general notion of *confidence* in the learned policy. Such a measure, indicating roughly how well the learner has learned the correct action for the current perception, is available for many learning algorithms. The demonstrator can then focus demonstration on areas of low confidence, and allow autonomous activity when it is high.

As with the primary feedback, multiple modalities are possible for providing this secondary information to the user. As it is often used to evaluate robot performance of the task, we have chosen to present it on the robot itself, so the user can see it while observing task performance. For users that are distally located, it may be more appropriate to display this information on the screen along with the perception data.

3.2.2 Control

The opposite of displaying perception to the user is getting task-appropriate actuation from them. Again, a variety of interfaces and modalities can be used. Providing multiple control interfaces may give potential users options, allowing them to chose the one they are most comfortable with. As with feedback, the platform developer must define the mapping from control interface to robot actuation.

A major concern when designing the control interface is how to map a user's actions onto a potentially higher degree-of-freedom actuation space. Similar to how different perceptual representations can make information easier or harder to access for a user, different actuation mappings can make particular actions easier or harder to perform. For example, using a 2 degree of freedom mouse to control the 3 dimensional location of the robot's head necessarily leaves one degree under-controlled. One option would be to use an adaptive teleoperation interface, where the level of control changes with the task [126].

Another possibility is kinesthetic control, where the robot itself is manipulated. By doing so, each degree of freedom of the robot is directly accessible to the user. However, as the user now interacts with the robot itself, it is often difficult to limit their perception to the displayed feedback. Alternatively, we could use a duplicate robot, allowing the user to manipulate it to control the robot engaged in the actual task. In both situations, the coordination necessary to provide correct control for all of the joints in real time is often lacking. There is, however, the possibility of having multiple users, each controlling a portion of the robot, and collectively providing demonstration data.

Alternate Demonstrators

As both perceptual feedback and actuation control interfaces may limit the user's ability to demonstrate tasks, their proper design and implementation may be research areas in their own rights. Further, their use is only for human demonstrators, who require alternate methods of accessing the data. For demonstrators that are more computational in nature, it may be possible to utilize the perception and actuation data structures directly.

Conceptually, such a demonstrator could be a learning system that has already learned the desired task, and is now teaching it to another system. An alternative would be a hand-coded controller (HCC), written by a programmer. However, the use of such a controller raises a question: "If we have code that drives the robot to perform the task, why not just use that, instead of teaching?"

The first reason to do so, and the reason we use them here, is for testing and development. Coded controllers, like simulators, allow for more control over unintended noise, and perhaps for faster-than-real-time data generation. An HCC, hooked up to a simulated robot through the DL architecture could be used to test the scalability of the system to learning from data that would take months to collect from human demonstrators. Further, HCCs can be designed to provide consistent actions for given perceptions, which many humans cannot do. Additionally, for tasks that require hidden state, a HCC can be written that explicitly provides this information, which may not be immediately accessible to a human demonstrator.

A second reason may be to enable adaptation of the policy itself. Writing an HCC that handles all possible situations may be difficult, as all rare edge cases must be considered. An alternative would be to write a simple controller that handles the most common cases, and then use that controller to train an adaptable learned system. Support for the edge cases could then be added via tutelage, as they arise. Similarly, if learning is able to improve the policy beyond that of the demonstrator, easily-written, but suboptimal, controllers could be used to bootstrap the process.

In both cases, the tutelage framework can be utilized with HCCs by allowing human users to toggle them on and off. That is, while the user does not control the content of the instruction, they control its presence, shutting it off to evaluate the learned autonomy. Such a combined Human-Gated Controller (HGC) keeps the human in the loop, and may also provide a method of probing the utility of the interactive teaching style itself.

3.3 Decision Making

The decision making component is the last part of the dogged learning architecture. It mediates the interaction between the platform and the demonstrator, and encapsulates the actual learning. If the demonstrator instead connected directly to the platform, or if learning did not take place, we would have a regular teleoperated robot.

As it is a mediator, the decision making component itself only requires knowledge of the dimensionality of the perception and actuation spaces of the platform. That is, no other knowledge as to which platform is being used, what the perceptions and actuations correspond to, or if the

platform is real or virtual, is needed. Decision making only views abstract perceptions coming in, and abstract actuations going out. It is for the demonstrator (and eventually the learner) to imbue them with meaning by showing the correct way in which they should relate to each other.

Abstractly, decision making itself can be viewed, as is the demonstrator, as taking in platform-generated perceptions and outputting actions appropriate for the task being performed. The difference is that from the decision making component we can obtain actions that are derived from the demonstrator, the learned autonomy, a default controller, or some combination of the three. We have already described the demonstrator in Section 3.2, now we discuss the other two sources of actuation outputs and the connections between them.

3.3.1 Learner

The learning portion of decision making is responsible for actually utilizing the demonstrator's generated preception-actuation pairs to form an approximation of their latent control policy. In addition, it must use that approximation to control the robot when the demonstrator is not demonstrating. The learner must then operate in two modes, which we call inference and prediction.

In inference mode, the inputs to the learner are the same $\hat{\mathbf{s}}$ that were displayed to the demonstrator, and the $\mathbf{a}$ that the demonstrator returned. Internally, the learner updates its approximation of the policy, and generates no output. We have focused on incremental, sparse learning, where the new policy (π_t) is derived from the previous one (π_{t-1}) and the data ($\hat{\mathbf{s}}, \mathbf{a}$) is discarded after the update. However, there is nothing in the concept of the learner that precludes other approaches. That is, the learner could be a batch learner, store all of the data it is shown, and recompute the policy from scratch each time.

The main reason to favor sparse incremental approaches over batch ones is that they are often faster and interruptible, and due to the tutelage paradigm, the learner is expected to be able to take control of the platform at any time. Control involves performing prediction, where the learner is given only $\hat{\mathbf{s}}$, and must generate an appropriate action. In the DL procedure in Algorithm 3.1, we interweave both prediction and inference. Given a perception, first prediction is called, to generate a possible output while we wait for the demonstrator to react. If the demonstrator provides a suitable action, it is used instead and the learner performs inference to update its policy.

Similar in spirit to our platform abstraction, the learning component of the DL architecture thus abstracts over the internals of various possible learning algorithms. Sparse, incremental, batch, parametric, nonparametric, Bayesian, or heuristic, all approaches can be used, as long as they implement inference and prediction. In this respect, DL can be used to perform empirical comparisons between different learning algorithms, similar to [32].

3.3.2 Default Controller

One of the optional components of the DL architecture is a default controller in the decision making subsystem. The idea is that it is a controller that is always active and will control the platform

in situations when both the learner and demonstrator are unwilling or unable. One such situation arises right after the system is started, before the demonstrator has demonstrated anything, and the learner is untrained. Another possibility is that the learner has learned poorly and is attempting to control the robot in an unsafe manner. The default controller thus can be seen as instantiating self-protective behavior, as it prevents an untrained (or ill-trained) learner from damaging the platform.

The default controller can also be thought of as providing prior information about desired robot behavior. That is, in the absence of demonstration to the contrary, the default controller provides an appropriate action for every perception. If the learner is allowed to learn from the default controller, data generated in this fashion will bias the learned autonomy.

The default behavior itself can range from being a simple constant controller, that always does the same thing, to a fully developed control policy. Further, it may be platform specific, as different values may be appropriate for different platforms. The main difference between the default controller and general HCCs is that the default controller contains no task-specific information. It, instead, provides control signals for when there is no task to perform. Possible default controllers include random exploration (if reinforcement learning is possible), continuing to do the same thing, or stopping all activity and awaiting instruction. We chose to do the last one, and use a default controller that always returns zeros for all actuation values, for all platforms.

3.3.3 Arbitration

Since the demonstrator, default controller, and learner may all be attempting to control the same physical platform, some form of determining the actual actuation ($\mathbf{a}$) used is needed. In our work, we arbitrate between the possibilities using the concept of confidence. That is, not only do the controllers produce outputs for query inputs, but we require them to have an associated measure of how sure they are that this is the correct action to perform. Confidence is a general notion, and there are various ways in which it can be derived. For a statistical learning algorithm, it could be a measure of variance in the predicted output. Alternatively, a measure of how well the state-action space is explored around the area of interest may be used. For a human, a more intuitive definition suffices, and we ask for a number between 0% (unsure) and 100% (completely sure). For comparison between the three controllers, we map all confidences to the human scale.

Actual arbitration itself can take many forms as well. We could, for example, normalize the confidences to sum to 1 and draw a controller probabilistically from the resulting distribution. Instead, we use a winner-take-all strategy, where the most confident controller gains control of the platform. In a general sense, this scheme allows for a confident learner to take control away from an less confident demonstrator, or a highly confident default controller (perhaps because of impending robot damage) to usurp them both.

We further simplify arbitration by using fixed confidences for the demonstrator and default controller. Specifically, the demonstrator is taken to be 100% confident if an action is generated, and 0% confident if not. By doing so we ease control somewhat, as the user does not need to provide the confidence directly. For the default controller, we use a constant τ, in our experiments $\tau = 10\%$.

Confidence	Demonstrator Present (100%)	Demonstrator Absent (0%)
Learner $> \tau$	Demonstrator controls	Learner controls
Learner $< \tau$	Demonstrator controls	Default / Demonstration requested

Table 3.1: Our arbitration matrix. The user, if providing control output, has command.

While this setting may allow a confident learned autonomy, or irresponsible demonstrator, to control the robot to cause itself harm, we have not found this to be the case.

Our resulting arbitration matrix is shown in Table 3.1. The results of arbitration, the selected action to be performed, is then passed on to the platform for execution and the learner for instruction. Note that in the absence of a demonstrator, the learner will receive signals from, and learn to mimic, the default controller, leading to the bias discussed earlier. To avoid feedback, where the learner's confidence grows without bound, we prevent the learning subsystem from learning from itself. If the arbitrator chooses the learner's output for enaction, the policy update does not take place.

3.4 Analysis

We have not rigorously analyzed Dogged Learning, as formal user studies are outside the scope of this dissertation. However, during its development our DL system has interacted with various users in multiple venues, such as scholastic projects, conference and symposia demonstrations, and research collaborations. Further, we have carried out several "proof-of-concept" experiments that, while not extensive, point out some fashions in which DL may be employed. We present here anecdotal evidence from these experiences, noting that platform developers, UI designers, machine learning researchers, human demonstrators, and other users interact with the system in different fashions.

During the work associated with this dissertation, we have implemented the DL architecture several times, in various fashions, the evolution of which is summarized in Table 3.2. Initially, our system was tied to one robot and particular tasks. Over time, as the research developed, the code was refactored to enable the easy inclusion of new platforms and controllers. More recently, we have added support for distributed interaction, where the demonstrator and learner need not be in the same physical location.

Currently, the development of our system focuses on making it easier to apply it to new robots and learning algorithms, so that it can be used to perform comparative studies. To that end we are moving towards a framework where each portion of the architecture (platform, control, feedback,

#	Name	Description
1	Prototype	A set of distinct programs that communicated over pipes in synchrony
2	Dogged Learning	A monolithic, multithreaded program
3	RGame	A server/client model for collecting data over the internet
4	Unnamed	A distributed system with cross-platform asynchronous communication

Table 3.2: Dogged Learning Instantiations

(a) AIBO (b) SMURV (c) Harvey

Figure 3.2: Instantiated robot platforms, the Sony AIBO, Brown SMURV and Harvard Harvey.
AIBO image copyright Sony Corporation.

learning) can be developed and run separately. One concern we have is with maintaining synchrony of the robot's perception and human's demonstration. In order to learn, we must correctly associate a commanded actuation with the perception that caused the human to perform it. However, if the system is fragmented and running over a network, lags make cause the various parts to come out of synch. We have addressed this problem somewhat in our current server/client system, which collects perceptions and actuations at a central point for synchrony.

3.4.1 Platforms

During development and testing of the DL architecture, we have implemented a variety of platforms, both robotic and virtual. Three of the physical robot platforms are shown in Figure 3.2, and their corresponding perception and actuation spaces in Table 3.3. Originally, our prototype DL system was tied to the AIBO, but since Version 2 support for multiple platforms has been possible. Currently, implementing a new platform consists of writing two C++ classes, one that handles communication with the robot, extracting perception from the sensors and translating actuation into appropriate actuator commands, and the other that interacts with the user, defining how perception is displayed and actuation obtained. We are continuing to streamline this process, and present here our main platform, along with our experiences in implementing others.

Platform	Perception	Actuation
AIBO	24D: 6 Colors × 3D (image X Y and size) + 4 head motors + 2 tail motors	10D: Walk Parameters (X Y and α) + 4 head motors + 2 tail motors + kick/block (discrete)
SMURV	26D: 6 colors × 4 bounding box corners + left and right bumpers	2D: Drive and Rotate
Harvey	6D: 3 colors × 2D (X and Y in the plane)	3D: Drive and Rotate, PickUp

Table 3.3: Platform perceptual and action spaces

AIBO

Our primary robot is the AIBO, pictured in Figure 3.2a. Developed by Sony and sold commercially for many years, the AIBO was used in the Robocup competitions from 1999 to 2008. To assist in this usage, Sony released an SDK, allowing programmers access to many low-level aspects of the robot. The robot itself has 18 degrees of freedom, a color camera, binaural microphones, touch sensors on the feet, chin, back and head, and 3 IR distance sensors. In addition, several LEDs on the head, back and face as well as a speaker are available for giving feedback to the user. Program code and data are stored on a removable "SmartDisc," with up to 256MB of space.

The low level proprietary operating system of the AIBO operates on a 32ms[2] frame rate, roughly 30Hz, making it well suited for human-interactive tasks. We use the onboard CPU to process raw sensor values and extract state estimates which are sent to a desktop processor over wireless ethernet (UDP). In our implementation state estimation, or perception, consists of color segmentation of the camera image to identify six colored blobs: black, orange, green, yellow, blue, and white. Segmentation and blobbing is performed by a custom vision library and the blob locations (in camera coordinates) and sizes, along with the motor angles of the head and tail make up the 24-D continuous perception space of this platform.

Similarly, the on board processor handles actuation, in response to commands received over wireless. The actuation space for this platform is 10-D, and consists of 3 walking directions (left/right, forward/back, turn) and the same 6 motors as in the state estimate. In addition, we use a discrete action indicator that triggers a pre-recorded kick or blocking motion. All control of the legs, for both walking gait and pre-recorded motions, occurs on board, and the leg positions themselves are in nither the perception or actuation space. For gait generation we use the UPenn gait algorithm [37] to generate leg positions from walk velocities, and motions are represented as a time-extended set of poses. Actual motor control itself is handled by a PID (Proportional-Integral-Differential) position controller built into the robot.

Note, that in not providing access to certain low-level values such as the leg positions and raw camera image, we may be limiting the capabilities of the platform. Additionally, we do not utilize the IR distance sensors, touch sensors, microphones, speakers, or LEDs, which could possibly be of use in some tasks. However, while including additional sensors and actuators may ease learning and performance of certain tasks, by revealing previously hidden state and enabling more control of the environment, we argue that there will always be tasks for which the current sensor configuration is insufficient for complete knowledge of the state of the world, and the current actuators insufficient for complete control.

SMURV

We have also implemented a DL-platform for the Brown SMall Universal Robotic Vehicle, or SMURV[3]. As the robot itself is developed in-house, use of the SMURV was designed to test the

[2] Actuator commands are handled in sets of 4, making the available resolution of position control 8ms.

[3] http://robotics.cs.brown.edu/projects/smurv

flexibility of the DL architecture in general, and our particular implementation. The robot itself, pictured in Figure 3.2b, was designed to be a low cost educational robot platform, suitable for floor-level manipulation and navigation. Built on an iRobot Create base, it uses a mini-ITX[4] computer for processing, data storage, and wireless ethernet communication. A webcam is the major sensor, and there are two bump sensors that provide contact information (front left and right only). Additionally, downward facing IR cliff sensors can detect stairs while a forward-facing IR emitter and 360 degree top-mounted IR receiver can be used for communication between robots. All sensors are processed on board via the Player/Stage framework [53].

Color vision is similar to that of the AIBO, in that 6 colors are used to detect objects. However, instead of our in-house segmentation library we used OpenCV [23], and instead of blob locations we extract bounding boxes. The sensor space of the SMURV is thus 26-D (6 color boxes with 4 corners and left and right bumpers). Note that, like in the AIBO platform, there are unused sensors (the cliff sensors) that could be used in the future. Actuation is 2-D, as the user controls forward/back motions and rotation, which is converted on board into control signals for the left and right wheels.

We highlight that the sensor and action spaces of the SMURV are different from that of the AIBO, both in dimensionality and content. In fact, while the object information (colors) is the same, it is presented in different manners. We further note that the methods used to perform perception are somewhat irrelevant. While both platforms use color segmentation to detect objects, they could just as easily use shape recognition or SIFT/SURF [17]. Using the DL framework, these differences are abstracted out, and the underlying perceptual system could be changed at a later date, without any needed modifications to DL, or the policies learned on the robots.

After implementing the SMURV, we ran quick learning experiments to show that the already existing algorithms could be immediately applied to this new platform. Using joystick teleoperation and interactive tutelage, we were able to train a SMURV to locate and approach green objects, and to back away from contact with walls. To do so, we needed to make no changes to any other component of the DL architecture, showing that our system is agnostic to the platform, as intended.

Harvey

Both the AIBO and SMURV are mostly limited to pushing balls around on the floor. To further examine the extensibility of our system, and to show its applicability to other tasks, we included in our system support for the Harvard Harvey humanoid robot, shown in Figure 3.2c. In contrast to the SMURV and AIBO robots, whose systems were almost entirely developed in house, Harvey was developed at another institution, for research independent of this dissertation, but still related to robot learning. The robot itself is a small wheeled robot, with two arms that serve as a gripper, enabling it to pick up and manipulate small objects.

Like the SMURVs, Harvey uses a color webcam to extract information about the world. However, due to the limited processing capabilities, the captured image is transmitted over wireless ethernet and processed offboard. The actual perceptual process is also more advanced than our other robots',

[4]Newer versions use the ASUS eeePC.

as Harvey tracks objects even when they are out of view. Using global localization and odometry in conjunction with vision, three colored objects are tracked in the space around Harvey and their locations in robot-centered coordinates make up the 6-D perception space of this platform. For actuation, Harvey has a 3-D space, corresponding to the left and right wheels, and a discrete pick-up motion. An object is determined to be held, or "picked-up" if it is in view before the pickup motion is executed, and not in view after, and is then taken to be at the same location as the robot.

Note, again, the different representations of the same data. In particular, Harvey makes use of increased perceptual capabilities to reduce the dimensionality of the perception space. The same information about objects that Harvey uses (2D location in the plane with respect to the robot) could also be extracted from the AIBO's color blobs or SMURV's bounding boxes. Reducing the perception space in this fashion may simplify learning of certain tasks, and make learning others harder or impossible. For example, Harvey cannot be used to learn tasks that require knowledge of the actual size of the observed objects, while both the AIBO and SMURV platforms can. Deciding on the appropriate level of abstraction is tantamount to deciding where to draw the line between the control policy and the computational architecture, determining which perceptual processings count as innate, and which are learned. By improving a robot's ability to learn arbitrary mappings from perception to actuation on its own, we hope to decrease the amount of high-level processing that must be assumed, and enable learning of a wider variety of tasks on a given platform.

Inclusion of Harvey into our system took around 5 hours for two researchers (the author and the developer of Harvey). Mostly, time was spent wrapping existing code and dealing with network issues. After completion, all aspects of DL were available for use on Harvey. We successfully used the same learning techniques to interactively teach Harvey to pick up and transfer the balls from location to location, again without any additional modifications to the other aspects of the system.

Virtual

We have also extended our system into the virtual domain. Doing so requires us to model the environment, but as they do not require a real running robot, these platforms are useful for performing quick tests of the system, and evaluating new features or additions. Alternatively, we can have more abstract platforms, one that we use often has both perception and actuation consisting of a single random number in the range $[-1, 1]$. With it, we have taught a variety of mathematical functions to our learning algorithms, including the square root multimap, as will be discussed in Section 4.3.1.

Further, the use of virtual systems shows how learned policies can be applied to different robots with the same perception and actuation spaces. In particular, we have simulated versions of both Harvey and the SMURV. Using the simulations, we have trained a controller for the SMURV, and then run that controller directly on the real robot.

3.4.2 Demonstrator Interfaces

Considering the interface that the end user will use to provide demonstration, we have worked both with graphic designers and potential users to improve it. Our current feedback display for the AIBO

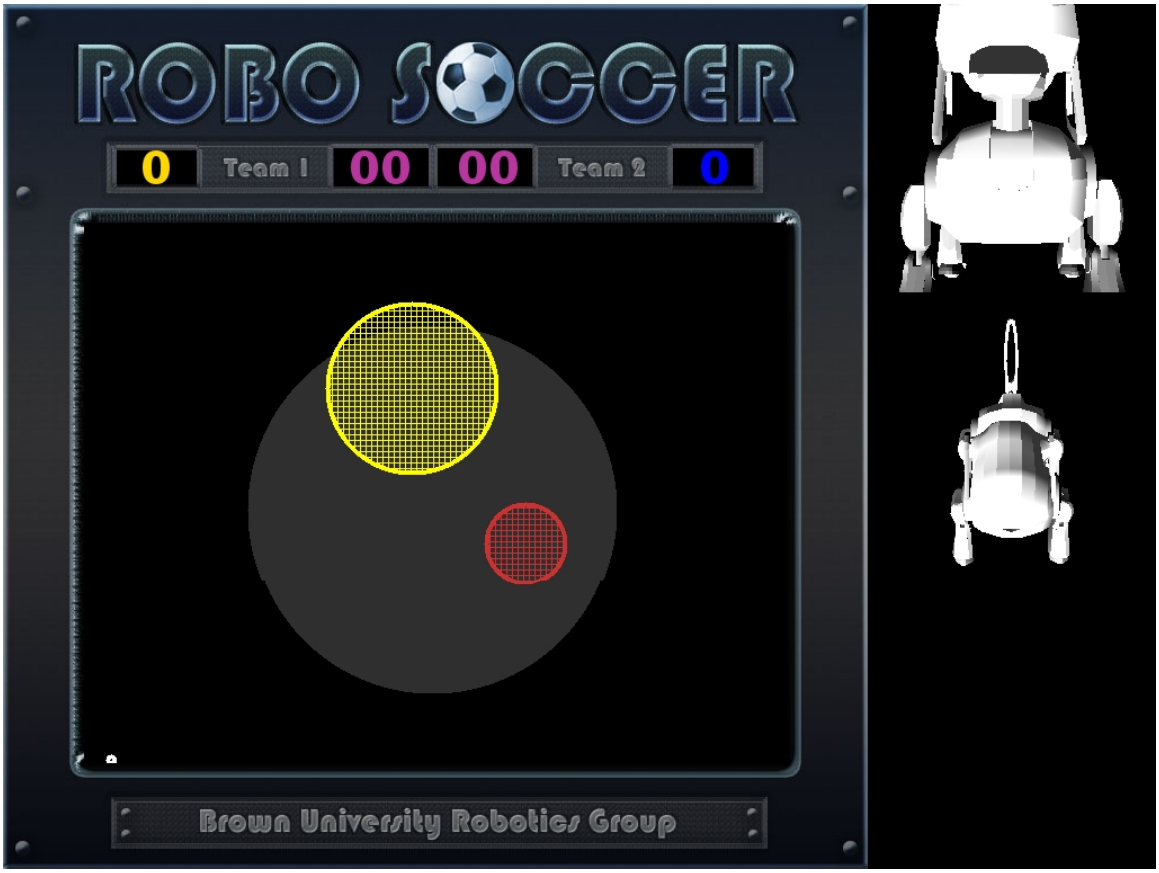


Figure 3.3: A screenshot of our AIBO feedback display. The extracted state estimate (color blob location and motor poses) is displayed to the user so that they can control an appropriate response.

is shown in Figure 3.3, and has a "video-game" feel. Color blob data is presented centrally, and motor pose information utilizes a model of the robot. As you can see, we present the data in the context of a game, robo soccer, in an attempt to make teaching the robot fun and exciting [5].

In terms of learning, we display secondary feedback on the robot itself, so that it is immediately accessible while observing the robot's performance. Shown in Figure 3.4, we use LEDs on the AIBO's ears to indicate the result of arbitration, and the LEDs on the back to indicate the associated confidence level.

For control, we have instantiated support for several devices, shown in Figure 3.5. Initially, we used the a two-joystick control pad in Figure 3.5a. However, we used the right joystick to control the head position, but as it only has 2 degrees of freedom, only two of the motors can be controlled at a time. The other two are accessed by "clicking" the joystick down. In feedback from users, this scheme was described as difficult to use, as it doesn't allow the user to control the tilt of the neck and chin at the same time, which is necessary for manipulating the ball. We then developed a control interface based on the wiimote (Figure 3.5b), which allows for more "natural" control of the robot [79]. For comparison with standard video game interfaces, we also implemented a keyboard

(a) High Confidence, Correct! (b) Low Confidence, Error!

Figure 3.4: The ear and back LEDs of the robot are used to make the learning process transparent, conveying information about arbitration results and confidence levels to the user

and mouse based controller, although in our experience all users prefer one of the other devices.

The mappings for each of the devises was fine-tuned based on feedback collected at public demonstrations. During demonstrations we distributed instructions for the current system and allowed untrained users to control the robots, as shown in Figure 3.6. As the control interface is kept distinct from other aspects of the system, desired changes can often be made quickly, to allow the user that proposed the change to evaluate it. In one case, a discovered bug in the interface was fixed in the time it took the robot to reboot.

Global Perception

During many demonstrations, we heard from users that they found the feedback display too restrictive. Instead, they often chose to watch the robot itself during teleoperation. By doing so, they were able to utilize information not available to the robot (such as global localization) to inform control decisions. In terms of the diagram in Figure 3.1, operating in this fashion would be equivalent to drawing an arrow connecting the environment directly to the teacher's feedback input.

(a) Control Pad (b) Wiimote (c) Keyboard

Figure 3.5: The user input devices we have experimented with. Copyrights: Logitech, Nintendo, Cymotion

Figure 3.6: Top: Instructions for using the control interfaces for our demos. Some users do not need them, and others require some additional verbal information, but most are able to control the robot in under a minute. Bottom: Novice users utilizing our control interfaces to teleoperate robots. If their perception were limited to that of the robot, we could learn policies from their data.

In this scenario, the abundance of hidden state often precludes successful policy learning. This failure is evidenced by the robot's performance of seemingly random actions, as it attempts to generalize over apparently contradictory demonstrator data. Learning in the presence of hidden state is one of the goals of this dissertation, and the approaches we expouse herein may eventually be able to address this scenario. Alternative means to approaching this issue would be to make the interface more compelling, utilizing work in HRI and UI design, physically separating the demonstrator and robot so that the user *must* use the feedback provided, or allowing the demonstrator to explicitly provide hidden information.

3.5 Discussion

Put together, the Dogged Learning architecture is an abstract description of the information flow during robot learning. It is then similar in many respects to the General Task Learning Framework (GTLF) of [154]. Both have as a goal to describe a system for enabling the development of autonomous robot controllers without explicit coding, while being as agnostic as possible to the robot, task, and learning system.

The GTLF is, in some senses, more general than DL. For instance, GTLF allows for learning to take place through insight, trial and error and instruction, in addition to the observation (or demonstration) that we do here. We have already discussed the addition of reinforcement learning techniques to DL to enable improvement of learned policies beyond the level of demonstration, which could be seen as instantiating the trial-and-error aspect. Insight and instruction techniques may also be incorporable.

GTLF also allows for reconfiguration of the perception and actuation space to ease task performance. This adaptation is accomplished by inserting filters in between the raw spaces of the platform and the ones on which learning operates. When learning indicates (perhaps through confidences) that an area of task performance is difficult, the filters can be modified to allow for improved learning. In DL, these filters are equivalent to the perceptual and actuation levels of the platform. We currently take them as fixed, but allowing them to be changed as needed may be an avenue of future development.

Further, GTLF is designed for episodic learning, where time-extended performance trials are executed before learning takes place. That is, data from a learning episode, consisting of demonstration data in our case, is only processed (in batch) after demonstration ceases. However, as mentioned both there and here, incremental learning, where learning occurs after each datapoint is generated, can be seen as a limiting case of episodic learning.

Batch processing, however, may lead to better learning, as the data can be considered more holistically. We have thus been considering an adaptation of DL that incorporates both incremental and batch processing. The main concept is that during demonstration, incremental learning takes place, but the data is logged as it is processed. Using the incrementally approximated policy, the robot can behave autonomously as soon as the user stops demonstrating. However, during

"down time," when there is additional processing power available, batch processing can be run, to improve the approximated policy. We note that for many learning algorithms, including the ones we examine here, prediction is much faster than inference. Thus, during autonomous execution, there is additional, unused processing power that could be utilized in this fashion. An alternative would be to have the batch processing occur when the robot is inactive, perhaps docked for charging. Alternating fast approximate learning and slow improving processing can be seen as a sort of sleep/wake cycle, where the robot improves what it has learned by further processing without user intervention. This approach is similar to those where the robot improves by "practicing" the task in the absence of the user, to improve without further interaction [19].

3.5.1 Data Collection

One issue that we have only hereto discussed tangentally is how the data for learning is actually gathered. We predict that for learning to perform arbitrary tasks in varied environments, large amounts of data may be required. Using simulated robots and HCCs is one way to generate the needed data, but the development of an HCC is the very task we are trying to avoid. Additionally, the use of simulators is not guaranteed to result in a policy that can be used on a real robot, depending on the fidelity of the simulator.

We must, then, collect data from human users. Traditional methods for doing so involve either bringing users to the robot (into the lab) or robots to the users (at demos, home visits, etc). We have used both approaches, but they each have drawbacks, as robots and the learning system may not be easily portable, and users who are willing to come to the lab may be scarce. Further, the collection of *large* data sets either requires much time from a few users, or many users for less time.

Towards this second approach, an alternative that we have started to explore is collecting data over the internet, a form of distributed human computation [5]. Such approaches, taping into the idle processing power of millions of humans that might be otherwise engaged in solitaire, has successfully been used to generate other large datasets, such as image labels and transcribed documents. Our idea, and the motivation behind Version 3 of our system, is to enable users to remotely demonstrate robot control policies from anywhere in the world. Thus, nither the robots nor the demonstrators need to travel, and users may be engaged for as much time as they wish.

The concept is that users will log in and be presented with the perception data from a robot running in our laboratory. By keeping the robots under local observation, they can be serviced as necessary. On the user's side, they would use whatever interface device they wish (keyboard, Wiimote, iPhone), to generate actuation commands to accomplish some task, perhaps framed as a game. From multiple users we may get data representative of different approaches to the same task. All data would be logged, and used to train autonomous policies. One concern is how to combine the data from the various demonstrators, some of whom may be more reliable than others [9]. The learned controllers can further be evaluated by having them "compete" against the humans.

3.5.2 Internal State

A key theme of this dissertation is learning in the presence of internal state, information that is known to the demonstrator but not the learning system. It arises most obviously when the demonstrator directly perceives things that the robot cannot, as when the user has global perception as in Section 3.4.2. However, it can also arise due to processes internal to the user, such as advanced perceptual capabilities and memory. Using the distributed data collection technique described above, hidden state is limited to this latter sort.

One method to address internal state no matter what the cause is to make it explicit, and available to the learning algorithm, which can then use this information to make control decisions. If we were to extend the actuation and perception vectors to include this state and draw a vertical line in Figure 3.1 between the actuation and perception channels, we could pass this hidden state into the decision making subsystem at the next timestep. An alternative would be to emit the state into the world, modifying it in some fashion that would be detectable, as a form of stigmergy [55].

In our experiments with unimap regressors, extending the actuation and perception of the platform in this manner has been used to turn a multimap policy into a unimap. That is, given the necessary internal state, perceptual aliasing is resolved, and the mapping from perceptions to actions becomes many to one. However, this approach to hidden state is not generally applicable.

Firstly, the identified state variables, their number and possible values, are often task-specific. This fact is in contrast with our stated goal, which is to develop learning for unknown tasks. Secondly, human demonstration often lacks explicit representation of these state variables. That is, while learning may be able to discover the evolution of the hidden state variables from demonstration, it requires demonstrated values for the variables to learn from. It may be possible to ask humans to provide this information in addition to their control signals, but doing so may require them to analyze the task in exactly the fashion we are trying to avoid. Further, there is no guarantee that the user-derived hidden state variables will be those needed for learning.

We instead seek to develop a learning algorithm that obviates the need for "passing around" this state, or getting it from humans. It can be seen as discovering perceptually aliased states in the execution of the demonstrated task and hypothesizing state variables that disambiguate them. The number of state variables and their values are all derived from the data, any may not correspond to those provided by users, but are sufficient for task performance.

Chapter 4

Realtime Overlapping Gaussian Expert Regression

In a sense, artificial intelligence will be the ultimate tool because it will help us build all possible tools. Advanced AI systems could maneuver people out of existence, or they could help us build a new and better world. Aggressors could use them for conquest, or foresighted defenders could use them to stabilize peace. They could even help us control AI itself. The hand that rocks the AI cradle may well rule the world.

K. Eric Drexler, Engines of Creation, 1990, page 76

This chapter introduces ROGER (Realtime Overlapping Gaussian Expert Regression), an incremental multimap regression model and algorithm for interactive robot learning from demonstration. Shown in overview in Figure 4.1, the model represents a multimap as a collection of overlapping unimaps, and key to the algorithm is its approach to *model selection*, or determining how many unimaps there are. By always considering the possibility that a datapoint is representative of a previously unseen unimap, ROGER effectively places no bound on the modality of the output distribution at a given input.

To reiterate the need for multimap regression, consider the toy example in Figure 4.2. Similar to the object avoidance task of [33], the demonstrator has indicated two possible outputs for the robot's current state. In contrast to [33], we consider that the options may not be truly equivalent, that the choice between them may depend on state not available to the robot, such as higher-level objectives or user preference. Using a unimap regressor, neither of these options will be learned, as performing unimap regression is equivalent to assuming that the observed outputs are unimodally distributed around the true target. Fitting a Gaussian (one choice of unimodal distribution) to the observations would result in their average being taken as the noise-free action. Parameterized by angle of turn, the two observed outputs are $\theta = 30°$ and $\theta = -30°$, and the two contradictory demonstrations will be averaged so that $\hat{\pi}(\texttt{puddle}) = 0°$, leading to incorrect behavior. Multimap regression, in contrast, seeks to learn that there are two possibilities, and return one or the other.

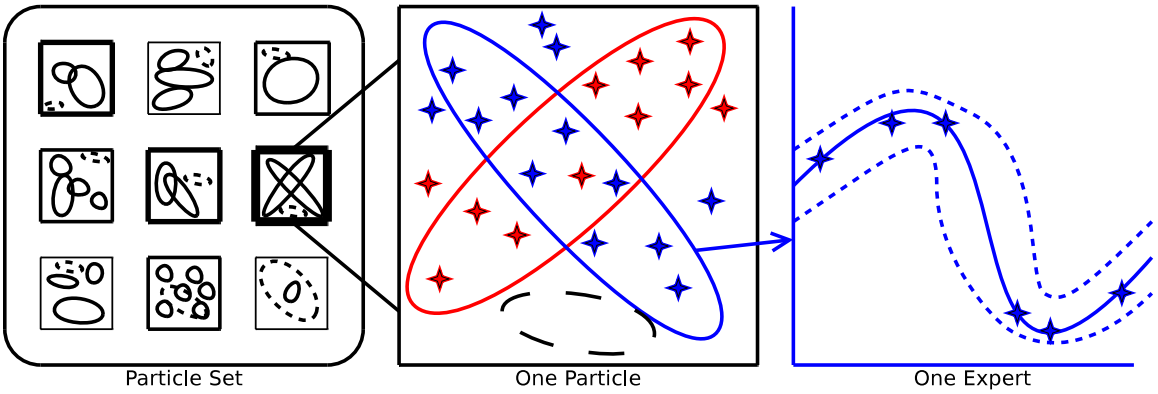


Figure 4.1: ROGER represents a distribution over all possible partitions of the data with a weighted particle set (left). A single particle (center) contains a set of experts in input space (Gaussian Mixture Model), along with a potential empty expert (dashed). Data (stars) are assigned to individual experts. Every expert (right) itself is a nonparametric SOGP regressor that maps from inputs to a Gaussian distribution over outputs using a sparse basis set.

We have the following desiderata for a multimap regression algorithm for robot tutelage.

1. Interactive speed: The ability to update the learned policy as data is generated (inference), and control the robot in realtime (prediction). Both software and hardware effect speed.
2. Scalability: The ability to handle data sets of size on the order of the lifetime of the robot. Particularly, the final speed of computation should be data-size independent.
3. Noise: The ability to deal with noise in perception, actuation, and demonstration.
4. Unknown parameterization: There is no reason to assume that the mapping from perception to actuation is of a known form, i.e, linear or unimodal.

ROGER has been designed to address these issues, and thus be suitable for use in a robot tutelage setting, particularly the DL architecture. In developing ROGER, we note that the above aspects of

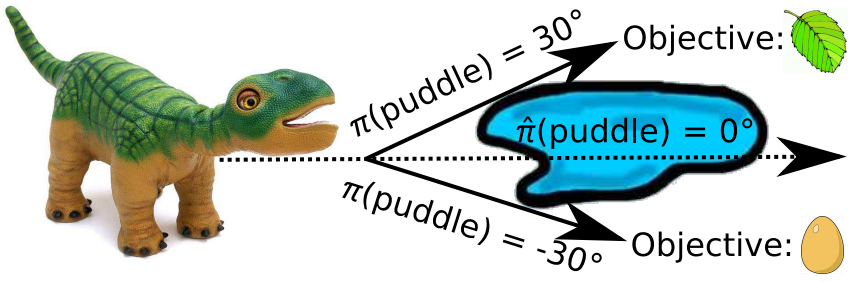


Figure 4.2: When presented with demonstrations of multiple actions for a given perception, unimap regression combines them together to estimate the assumed one correct output. The resulting mapping may then lead to incorrect behavior. ROGER, a multimap regressor, seeks to learn that there are multiple correct actions, associated perhaps with different objectives. Pleo copyright Ugobe

an algorithm are interrelated. For instance, an algorithm may initially run at interactive speeds, but slow down as it scales to larger datasets. In contrast, we desire an algorithm that continues to be interactive even as the data size grows. We have thus chosen to make ROGER an incremental, sparse algorithm. Incremental in the sense that it updates the current approximation $\hat{\pi}$ to incorporate new data as it arrives, instead of recomputing it anew, and sparse in that it does not require that all previous data be kept for future consideration.

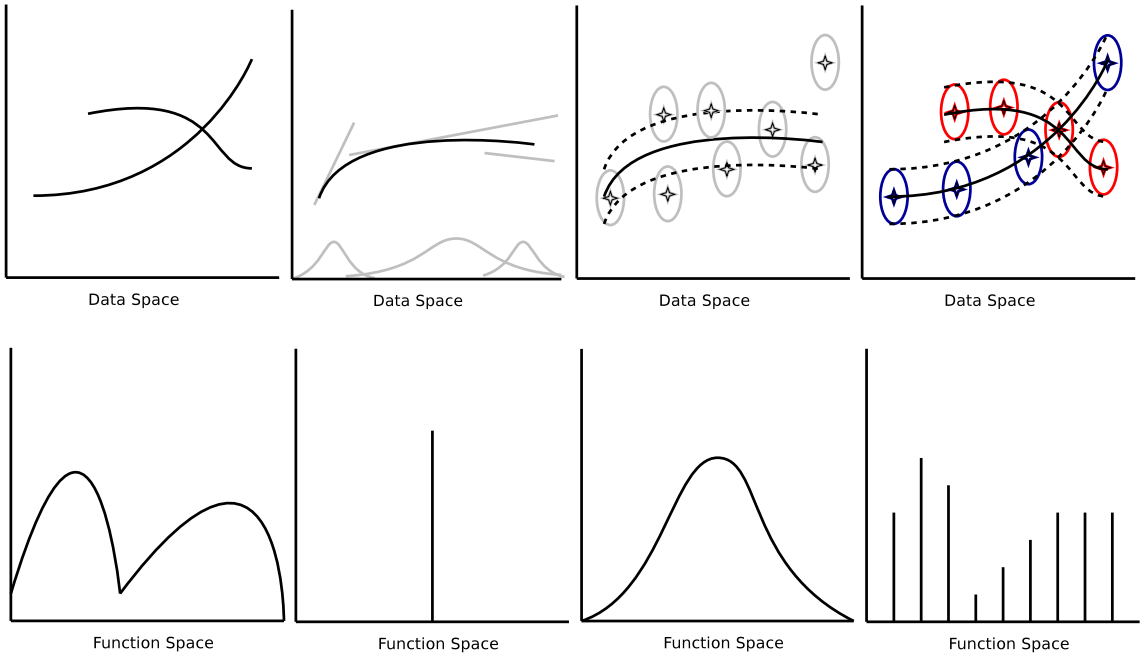
We note that the speed of a learning algorithm depends not only on its time and space complexity, but on the underlying hardware as well. That is, batch algorithms, that process all data after each new datapoint arrives and thus require that all data be stored, can be interactive, if the underlying computational and memory devices are fast enough. However, we argue that in the limit, as robots operate over longer lifetimes, the amount of data generated will overwhelm any batch algorithm with finite storage and computational power. For fixed-lifetime robots, this may not be an issue, and advances in computational and memory hardware may alleviate this problem to a certain degree.

In terms of noise, we acknowledge that a robot's sensors and actuators are inherently noisy. Thus, the control policy and its learned approximation must be robust to motors that do not do what is commanded, and world states that appear different over time. A further concern is that the noise may be nonstationary, or dependent upon the values of the variables themselves, or even time. The human demonstrator is also source of potentially nonstationary noise. That is, while human users may be *attempting* to perform optimal control for the task at hand, the outputs they generate may be corrupted by some error. It is unlikely that technological progress will be able to remove this concern, thus the learning system must operate in the presence of this noise, and attempt to learn what the demonstrator means to accomplish.

Lastly, we do not wish to assume a known model for the mapping itself, as we desire robots that can learn unknown tasks over their entire lifetime. Simply performing multimap as opposed to unimap regression addresses this concern partially. We will also, however, consider the mappings of the individual unimaps of which the overall multimap is comprised, and eschew linear and parametric models in favor of nonlinear and nonparametric ones. However, by avoiding known models, we must conted even more so with noise, as it will be harder to separate out the signal without knowing the mapping's form. We thus rely on a preponderance of data to enable successful learning, and put an emphasis on interpolation between observed data rather than extrapolation beyond the limits of what has been seen. Also, note that ROGER is not without assumptions as to what mappings are more likely, which may still lead to biases in learning.

4.1 Model

ROGER can be viewed as a set of overlapping unimap regressors, each of which captures one of the multiple possible outputs in an underlying multimap. It is then a Mixture of Experts (MoE) model, where each unimap regressor is an expert [68]. For the unimap regression in each expert we use SOGP, a sparse nonparametric regressor [42]. Datapoints are assigned to experts, or *gated*, using a



(a) Truth (b) LWPR (c) SOGP (d) ROGER

Figure 4.3: A function space view of regression. True multimap data corresponds to a multimodal distribution over functions (a). LWPR (b) averages the multimap data with a set of linear regressors with Gaussian areas of influence in input space. The result is a single point estimate in function space. SOGP (c) instead has a unimodal (Gaussian) distribution over the mappings, providing variance in the prediction, and uses Gaussian kernels in the joint space. ROGER (d) associates each kernel point with a particular unimap, and approximates the full distribution over functions with a set of point estimates. For all algorithms, the number of fields, kernels, and unimaps is derived nonparametrically from the data.

Gaussian Mixture Model (GMM), where each expert holds sway over a Gaussian-shaped region of input space. When these regions overlap, multimap scenarios occur.

Rather than setting the number of experts in advance, or deriving it in a brute-force or ad-hoc manner, we place a Dirichlet process prior over the number of experts, effectively considering an infinite number of them [105]. The input space gating then becomes an Infinite Gaussian Mixture Model [113], and the overall ROGER model an example of an Infinite Mixture of Gaussian Process Experts [112]. Inference in this model then performs both model selection, discovering an appropriate number of experts, and policy learning, or unimap regression, in each one.

A similar model has previously been used to address both multimap regression and nonstationary noise in a batch framework [89]. For robot tutelage, we require an incremental formulation and thus developed a corresponding sequential technique [153]. In doing so, we sacrificed the ability of the model to deal with nonstationary noise in favor of using conjugate priors to speed up learning, although it may be possible to incorporate both aspects at a later date.

As ROGER is primarily a regression algorithm, it can be seen as as attempting to determine

the relationship of dependent variables (outputs) to independent variables (inputs). Another view, presented in Figure 4.3, is to see it as trying to find, in the space of all functions, those functions $f(\mathbf{x})$ that fit the observed data $(\mathbf{x}, \mathbf{y})$. In the function spaces of Figure 4.3, functions are nearer to those that produce similar outputs for all inputs.

Given a true distribution from a multimap (Figure 4.3a), non-Bayesian approaches, such as continuum regression [127] or LWPR [150], find a single point-estimate, a "best fit," that ideally lies at the most likely function (Figure 4.3b). Such an approach is the same as assuming that the true underlying distribution is unimodal, and trying to find the peak. However, incremental approaches, such as gradient ascent [130] or expectation maximization [131], can get caught in local maxima and find less optimal solutions.

Alternatively, Bayesian approaches such as SOGP or GPR (Figure 4.3c) track the distribution as a whole, possibly assuming a known unimodal form. In SOGP, this form is Gaussian, with a mean function and associated variance. This assumption means that functions with outputs that differ significantly from the output of the mean function for a given input must necessarily be much less likely. By tracking the distribution instead of just a single point, incremental Bayesian approaches may be able to escape from local optima, and discover better approximate functions. Additionally, the distribution can be used to associate error bars, or variances, with predictions.

ROGER, and other multimap regression algorithms, instead assume a multimodal distribution in function space, where functions with highly different outputs can be equally likely, as occurs in multimap scenarios. However, as ROGER does not know the true number of modes, it does not use a smooth curve to track the distribution, as SOGP does. Instead, a finite number of samples, drawn from the distribution, make up the approximation via Monte-Carlo integration (Figure 4.3d).

While a potentially infinite number of experts (or modes of the distribution) are implicitly considered, at any particular point in time only a finite number, K, of them matter, corresponding to those experts which have actually generated data. In the limit then, K cannot be greater than N, the total number of data points seen, if each has been generated by a distinct unimap. Often, however, $K << N$, and ROGER makes use of a set of latent indicator variables ($\mathbf{z}$) indicating which of the experts gave rise to each particular input/output pair. The $\mathbf{z}$ represent a partitioning of the data and are themselves generated by a Chinese restaurant process (CRP) [105] with concentration parameter α. The concentration parameter puts a prior over the number of experts, and how uniform the assignment of input/output pairs to experts is thought to be (large α implies many experts).

Each of the K experts is an SOGP regressor and a corresponding multivariate-normal input model (the gate). In other words, there are K multivariate-normal classes that generate input points, and an SOGP expert for each class which is responsible for generating outputs given the inputs. Each input space component is a multidimensional Gaussian that has mean parameter μ_k and covariance parameter Σ_k. These parameters themselves are drawn from a normal-inverse-Wishart conjugate prior. This choice of prior allows the user to influence how input space is partitioned by the model, without having to specify it exactly. Further, because it is a conjugate prior, all possible values for the parameters can be analytically integrated out.

The generative model underlying ROGER can be summarized in the following equations:

$$\begin{array}{lcl} z_i & \sim & \text{CRP}(\alpha) \\ \Sigma'_k & \sim & \text{Inverse-Wishart}_{\nu_0}(\Lambda_0) \\ \mu'_k & \sim & \text{Multivariate-normal}(\mu_0, \Sigma_k/\kappa_0) \\ \mathbf{x}_i|z_i & \sim & \text{Multivariate-normal}(\mu'_{z_i}, \Sigma'_{z_i}) \\ \mathbf{y}_k|\mathbf{X}_k, \theta & \sim & \text{Multivariate-normal}(0, \mathbf{Q}_k) \end{array} \tag{4.1}$$

From this it is straightforward to write down the joint distribution of the inputs $\mathbf{X}$, outputs $\mathbf{y}$, and class labels $\mathbf{z}$ defined by ROGER:

$$P(\mathbf{X}, \mathbf{y}, \mathbf{z}; \Omega) = P(\mathbf{X}|\mathbf{z}; \Omega)P(\mathbf{z}|\Omega) \prod_{k=1}^{K} P(\mathbf{y}_k|\mathbf{X}_k, \Omega). \tag{4.2}$$

Here $\Omega = \{\alpha, \mu_0, \kappa_0, \Lambda_0, \nu_0, \theta\}$, is the collection of all parameters, and $(\mathbf{X}_k, \mathbf{y}_k) = (\mathbf{X}_i, \mathbf{y}_i) \forall i, z_i = k$ are the data associated with each expert. The GP parameters, θ, are shared by all experts.

Each part of this joint corresponds to portion of the overall model in Figure 4.1. $P(\mathbf{X}|\mathbf{z})$ is the input Gaussian Mixture Model (center), and will be examined in Section 4.1.1. $P(\mathbf{z})$ gives us the distribution over partitions (number of experts and the assignment of data to them), which is represented as a set of particles (left) and will be discussed in Section 4.1.2. Lastly, $P(\mathbf{y}_k|\mathbf{X}_k)$ is the distribution over mappings in one particular expert (right), and corresponds to the SOGP regressor explained in Section 4.1.3.

4.1.1 Input Space Density Estimation

The input space model, or gating network, of ROGER is a Gaussian Mixture Model (GMM), where the possible parameters for the individual model components have been integrated out. Starting with a standard GMM and assuming K components for now, the initial gating network is

$$P(\mathbf{X}|\mathbf{z}) = \prod_{k=1}^{K} P(\mathbf{X}_k|\mu_k, \Sigma_k) = \prod_{k=1}^{K} \prod_{m=1}^{m_k} P(\mathbf{x}_{kn}|\mu_k, \Sigma_k) \tag{4.3}$$

m_k is the number of datapoints assigned to expert k and the probability of a datapoint under its expert is the standard Gaussian distribution

$$P(\mathbf{x}|\mu, \Sigma) = \frac{1}{(2\pi)^{(D/2)}|\Sigma|^{(1/2)}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^\top \Sigma^{-1}(\mathbf{x}-\mu)\right) \tag{4.4}$$

where D is the dimensionality of the input space.

Instead of providing particular values for the gating parameters, they are instead drawn from the prior distributions shown in Equation 4.1. The inverse-Wishart prior is:

$$P(\Sigma|\Lambda, \nu) = \frac{|\Lambda|^{(\nu/2)}|\Sigma|^{-((\nu+D+1)/2)} \exp(\mathtt{tr}(\Lambda\Sigma^{-1})/2)}{2^{(\nu D/2)}\Gamma_D(\nu/2)} \tag{4.5}$$

and Γ_D is the multivariate Gamma function.

Combining Equations 4.4 and 4.5, the probability of an input point under an expert whose mean and variance parameters are drawn for the corresponding priors is

$$P(\mathbf{x}|\Lambda,\nu,\mu_0,\kappa_0,\mu,\Sigma) = P(\mathbf{x}|\mu,\Sigma)P(\mu|\mu_0,\Sigma/\kappa_0)P(\Sigma|\nu,\Lambda) \tag{4.6}$$

The first term of Equation 4.6 is the probability of the datapoint with particular expert parameters (Equation. 4.4), the second term is the probability of the expert's mean parameter under the prior (Equation. 4.4) and the third term is the probability of the expert's variance parameter under the prior (Equation. 4.5).

As these prior distributions are *conjugate*, it is possible to analytically integrate over all possible values for μ and Σ. Conjugacy is the property that the posterior distribution that results from combining the likelihood and prior has the same form as the prior itself, so that the posterior can then be used as a prior in the next incremental step. Performing the integration, the resulting input space model for ROGER is then

$$P(\mathbf{X}|\mathbf{z};\Lambda,\nu,\mu_0,\kappa_0) = \prod_{k=1}^{K} P(\mathbf{X}_k|\Lambda,\nu,\mu_0,\kappa_0) = \prod_{k=1}^{K} \left(\frac{\kappa_0}{\kappa_k}\right)^{\frac{D}{2}} 2^{\frac{D}{2}(\nu_k-\nu_0)} \frac{|\Lambda_0|^{\frac{v_0}{2}} \prod_{d=1}^{D} \Gamma(\frac{\nu_k+1-d}{2})}{|\Lambda_k|^{\frac{v_k}{2}} \prod_{d=1}^{D} \Gamma(\frac{\nu_0+1-d}{2})} \tag{4.7}$$

where we follow [52] in making the following variable substitutions

$$\begin{aligned}
\kappa_k &= \kappa_0 + m_k \\
\mu_k &= \frac{\kappa_0}{\kappa_k}\mu_0 + \frac{m_k}{\kappa_k}\bar{x}_k \\
\nu_k &= \nu_0 + m_k \\
\mathbf{\Lambda}_k &= \mathbf{\Lambda}_0 + \mathbf{S_k} + \frac{\kappa_0 m_k}{\kappa_k}(\bar{x}_k - \mu_k)(\bar{x}_k - \mu_k)^T \\
\mathbf{S_k} &= \sum_{j:z_j=k} (x_j - \bar{x}_k)(x_j - \bar{x}_k)^T \\
\bar{x}_k &= \frac{1}{m_k} \sum_{j:z_j=k} x_j
\end{aligned} \tag{4.8}$$

The individual datapoints are then drawn from the distribution $P(\mathbf{x}_i|z_i = k, \mathbf{X}_k) = \text{Student-t}_d(a, B)$ with degrees of freedom $d = \nu_k - D + 1$, mean $a = \mu_k$, and scale matrix $B = \Lambda_k(\kappa_k+1)/(\kappa_k d)$.

4.1.2 Model Selection

The above describes a finite GMM, where the number of components, K, is taken as known. If it is not, we can instead posit a distribution over K, and track this distribution over the number of components as data arrives. In doing so we can consider multiple possible orders of models simultaneously, to discover the number of experts that best fits the data.

ROGER uses a Dirichlet Process (DP) prior to generate the number of experts in its mixture of experts model. Like a Gaussian Process, a DP is a distribution over distributions, $G \sim DP(\alpha, G_0)$. From G the individual expert assignments z_n are drawn, and the resulting $\mathbf{z}$ defines a partitioning

of the data, which includes the number of partitions, K. Integrating over all G that can be drawn from the DP gives rise to a distribution over partitions that is equivalent to the one that the Chinese Restaurant Process creates, with the probability of the partitioning represented by $\mathbf{z}$ equal to:

$$P(\mathbf{z}|\alpha) = \frac{\Gamma(\alpha)\alpha^K}{\Gamma(\alpha+N)} \prod_{k=1}^{K} \Gamma(m_k) \tag{4.9}$$

The CRP itself can be described as a sequential process that generates sequences of integers where the probability that the next integer in the sequence is k is proportional to the number of times k has already appeared in the sequence. The probability that the next integer takes on a new value of k is proportional to α.

$$P(z_i = k|\mathbf{z}_{-i};\alpha) = \begin{cases} \frac{m_k}{N+\alpha-1}, & k \leq K \\ \frac{\alpha}{N+\alpha-1}, & k = K^+ \end{cases} \tag{4.10}$$

K^+, the number of experts that must be considered, is thus the number of unique values that appear in $\mathbf{z}$, plus one, for the new, empty expert.

This CRP view of the DP is more amenable to a sequential implementation, since the individual components of $\mathbf{z}$ can be generated in a sequence. However, the resulting distribution over $\mathbf{z}$ is equivalent to that determined by alternate constructions, such as stick-breaking [140]. There the values of the m_k are computed directly.

The utility of this view may be best seen via the CRP metaphor, which has an infinite stream of customers arriving at and choosing seats in a Chinese restaurant with an infinite number of tables, each with infinite seating capacity. Each customer sits at a table with probability proportional to the number of customers already at that table ($k \leq K$), and with some probability sits at a new table ($k = K^+$). At any given time, after N customers have been seated, the next customer only has to consider K^+ tables when making their choice, as all of the infinite empty tables are equivalent.

We note that in using the CRP, the *rate* at which experts form is non-uniform. At the beginning, when $\alpha > m_k \forall k$, it is more likely that a new expert will be hypothesized. Later, once more data has been seen, it is more likely that a new datapoint will be assigned to an already extant expert. This fact biases the learning process, but we consider it to be an appropriate bias. Over the lifetime of a robot, we expect it to initially have to learn many new subtasks, while later in life it will reuse them and not have to learn many more.

Returning to the GMM input space model of ROGER, using the CRP to generate the assignment of data to experts effectively allows for the consideration of an infinite number of experts. At any given time, only K experts, those with data assigned to them, and an additional empty expert, need to be considered. As new data arrives and is processed, points may be assigned to previously empty experts, increasing K.

4.1.3 Expert Output Regression

The distribution of outputs given inputs in the ROGER model is again a mixture of experts, where each expert generates outputs for the data gated to it by the input model. ROGER uses a sparse approximation to Gaussian Process Regression (GPR) [82] which tracks a Gaussian Process (Gaussian distribution over functions) over all functions that could have generated the data. Before explaining the approximations that must be made to achieve sparsity, we describe GPR itself.

A GP is defined by mean and covariance function and describes a normal distribution over possible functions, $\mathcal{N}(f, \Sigma)$. Starting with a mean zero prior, and given a set of data $(\mathbf{X}, \mathbf{y})$, GPR first defines a kernel function which represents similarity between two points in input space. A popular kernel that we use in our work is the squared exponential, or Radial Basis Function (RBF)

$$\texttt{RBF}(\mathbf{x}, \mathbf{x}'; \sigma_k^2) = \exp\left(-0.5 * \frac{||x - x'||^2}{\sigma_k^2}\right) \tag{4.11}$$

where σ_k^2 is termed the kernel width, and controls how strongly nearby points interact.

A posterior distribution over functions is then defined

$$f(\mathbf{x}') = \mathbf{k}_{\mathbf{x}'}^\top \mathbf{C}^{-1} \alpha \tag{4.12}$$

$$\Sigma(\mathbf{x}') = k^* - \mathbf{k}_{\mathbf{x}'}^\top \mathbf{C}^{-1} \mathbf{k}_{\mathbf{x}'} \tag{4.13}$$

where $\mathbf{k}_{\mathbf{x}'}$ is shorthand for $[k_1, k_2 ... k_N], k_i = \texttt{RBF}(\mathbf{x}', \mathbf{x}_i; \sigma_k^2)$, or the kernel distance between the query point and all previously seen points $\mathbf{X} = \{\mathbf{x}_i\}_i^N$. α is the output vector, which in this case equals the known data's outputs, $\mathbf{y}$, and $\mathbf{C}$ is the covariance matrix of the data, where $C_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) + \delta(i == j)\sigma_0^2$. This second term, σ_0^2, represents observation noise (modeled as a Gaussian with mean 0 and variance σ_0^2). The first term, without the noise, is the Gram matrix ($\mathbf{Q}$), or all-pairs kernel distance.

In terms of the joint probability in Equation 4.2, the probability of the observed outputs given the inputs for expert k is:

$$P(\mathbf{y}|\mathbf{X}; \theta) = \prod_{m=1}^{m_k} P(y_m|x_m; \mathbf{X}_{/m}, \mathbf{y}_{/m}, \theta) = \prod_{m_1}^{m_k} \mathcal{N}(y_m; f(\mathbf{x}_m), \Sigma(\mathbf{x}_m)) \tag{4.14}$$

However, as calculating these probabilities involves using $\mathbf{C}^{-1}$, and as $\mathbf{C}$ occupies $O(N^2)$ space and requires $O(N^3)$ time to invert, GPR is not directly suitable for our learning scenario. Using the partitioned inverse equations, $\mathbf{C}^{-1}$ can be computed directly and incrementally as data arrives, removing the inversion step [82]. The space requirements must be dealt with separately using one of a variety of approximation techniques [110].

Many of these techniques operate by approximating the full posterior distribution (based on N points) with one based on fewer ($\beta < N$). These fewer points are called the basis set, or the basis vectors (BV). This reduction limits the size of the Gram and covariance matrices to β^2, which can be tuned for desirable properties, such as speed of computation or percentage of system memory used, for each particular implementation. The approximating distribution itself is chosen to minimize the KL-divergence with the true distribution.

Algorithm 4.1 Sparse Online Gaussian Processes

Inference	**Prediction**
Require: Training pair $(\mathbf{x}, y)$	**Require:** Query point $(\mathbf{x})$
Basis Vectors ($\mathtt{BV}$), model $(\alpha, \mathbf{C}^{-1}, \mathbf{Q})$	Basis Vectors ($\mathtt{BV}$), model $(\alpha, \mathbf{C}^{-1})$
GP parameters $(\theta = \{\sigma_k^2, \sigma_0^2\})$	GP parameters $(\theta = \{\sigma_k^2, \sigma_0^2\})$
capacity (β),$\lvert\mathtt{BV}\rvert < \beta$	capacity (β),$\lvert\mathtt{BV}\rvert < \beta$
Ensure: Updated model and $\mathtt{BV}$, $\lvert\mathtt{BV}\rvert < \beta$	**Ensure:** predicted output $\hat{y}$, stddev σ^2
add $\mathbf{x}$ to $\mathtt{BV}$ and update $\alpha, \mathbf{C}^{-1}\mathbf{Q}$	**for** $b = 1 : \lvert\mathtt{BV}\rvert$ **do**
if $\lvert\mathtt{BV}\rvert > \beta$ **then**	$k_b = \mathtt{RBF}(\mathtt{BV}_b, \mathbf{x}'; \sigma_k^2)$
for $b = 1 : \lvert\mathtt{BV}\rvert$ **do**	$k^* = \mathtt{RBF}(\mathbf{x}', \mathbf{x}'; \sigma_k^2)$
$\epsilon_b = \alpha_b / \mathbf{C}_{b,b}^{-1}$	$\hat{y} = \mathbf{k}^\top \alpha$
Delete j from $\mathtt{BV}$, $j = \mathrm{argmin}_j\, \epsilon_j$	$\sigma^2 = \sigma_0^2 + k^* - \mathbf{k}^\top \mathbf{C}^{-1} \mathbf{k}$

It should be noted that all the discussion of GPs in this section, and the output model of ROGER in general, is with respect to scalar outputs. We can apply these techniques to vector outputs by providing a interdependence matrix, or, as we do here, assuming independence between the outputs. This assumption, while almost always false, often provides good results, and has done so in our case. Future work could look into including dependencies between output dimensions, perhaps learning the interdependence matrix from the data [22].

SOGP

We use the Sparse Online Gaussian Process (SOGP) algorithm proposed by [42] to perform an incremental approximation to the entire GP. The use of GP or SOGP, or another approximate method, is transparent to the rest of the ROGER algorithm, so we explain it briefly here. In essence, when the $\beta + 1$ point arrives, SOGP initially includes it as a basis vector. All points are then assigned a score corresponding to the residual error between the distribution based on all points and the distribution based on all points except the one being considered. The point with the lowest score is selected for removal. An overview of the algorithm can be seen in Algorithm 4.1, and full details of the derivation of the score equation are in [41].

A naive implementation of sparsity simply removes the deleted data. However, information from the removed points can still be used to adjust the distribution approximated by $\mathtt{BV}$. This modification is accomplished by changing the outputs associated with each basis vector, and also editing the covariances between them. The approach is similar to that of using pseudo-inputs [124], where totally new basis vectors are derived in batch. For us, we must now make a distinction between the output vector α and the $\mathtt{BV}$'s outputs $\mathbf{y}$, as the "edited" outputs are no longer the same as those observed. Likewise, $\mathbf{C}$ no longer tracks $\mathbf{Q} + I\sigma_0^2$ and the two must be stored separately. While now two matrices must be stored, they are both of size β^2, so total memory usage is still $\mathrm{O}(\beta^2)$.

Algorithm 4.2 Realtime Overlapping Gaussian Expert Regression

Inference

Require: Training pair $(\mathbf{x}, y)$
  P particles ($\mathtt{P}$) and weights ($\mathbf{w}$)
  Hyperparameters ($\Omega = \{\alpha, \mu_0, \Sigma_0, \Lambda_0, \kappa_0, \theta\}$)
Ensure: Updated particles and weights
  for $p = 1 : |\mathtt{P}|$ **do**
    for $k = 1 : K^{+(p)}$ **do**
      Weigh putative particle ρ_{pk} by 4.15
  Sample P putative particles and weights
  Assign data to appropriate experts

Prediction

Require: Query point $(\mathbf{x})$
  P particles ($\mathtt{P}$) and weights ($\mathbf{w}$)
  Hyperparameters ($\Omega = \{\alpha, \mu_0, \Sigma_0, \Lambda_0, \kappa_0, \theta\}$)
Ensure: predicted output $\hat{y}$, stddev σ^2
  $p^* = \text{argmax}_p\, w_p$
  for $k = 1 : K^{+(p)}$ **do**
    $e_k = P(z' = i | z^{(p)}) P(x' | \mu_0, \Sigma_0, \Lambda_0, \kappa_0)$
  Sample e^* according to $\mathbf{e}$
  Predict from expert e^* as in SOGP

4.2 Algorithm

ROGER consists of not only the model described above, but also an algorithm for performing both inference and prediction. Observe that inference is the process of finding values for the latent variables, or an assignment of datapoints to experts, $\mathbf{z}$, that maximizes the joint probability of Equation 4.2. Prediction is using the current values of $\mathbf{z}, \mathbf{X}$, and $\mathbf{Y}$ to generate new data, either full $(\mathbf{x}, \mathbf{y}, z)$ pairs, or parts thereof. We most often generate $(\mathbf{y}, z)$ given an $\mathbf{x}$, corresponding to choosing an action to perform based on the current perception from among the possibly multiple applicable subtasks. Alternatively, we could also generate $\mathbf{y}$ given $(\mathbf{x}, z)$, which assumes that we already know which subtask is active. Lastly, we could also "hallucinate" a perception leading to a known action.

Multiple algorithms could possibly be used to infer the underlying distribution on $\mathbf{z}$, although we tend towards Bayesian ones to leverage the distributional properties of the model. One brute-force approach would be to enumerate all of the possible partitions of the data and calculate the joint likelihood of the observed variables for each one. However, the combinatorics of the partitioning makes this approach unsuitable for all but the smallest datasets. For example, with only 1000 datapoints, or roughly 30 seconds of data, there are $\sim 2 \times 10^{31}$ possible sizes of partitions alone, not counting the different assignments of data to those partitions.

In keeping with our desire for incremental estimation, we chose to base our algorithm, shown in pseudocode in Algorithm 4.2, on the concept of a particle filter [47]. As the form of the distribution over possible partitions is unknown, it is represented using a finite set of P weighted particles as discussed in Section 4.1.2. We use a weighted particle set rather than an unweighted one to achieve increased breadth in our representation of the distribution. As an illustration of the advantage, consider representing a distribution over the integers, using only 5 samples. In an unweighted particle set, a more likely integer must appear more times, as in the set $\{3, 3, 3, 2, 4\}$, which states that the number 3 is three times as likely a 2 or 4, which are both more likely than all other integers. A weighted particle set, on the other hand, such as $\{3_3, 2_1, 4_2, 1_{0.5}, 5_{0.3}\}$ contains a finer representation of the distribution with the same number of particles.

In ROGER, each particle maintains a separate estimate of the assignment of datapoints to

experts $\mathbf{z}^{(p)}$, and thus also a separate estimate of the total number of experts, $K^{(p)}$. With $\mathbf{z}^{(p)}$ and the observed data it is possible to derive all the necessary values for calculating the joint itself. However, to speed up computation and allow for sparsity, ROGER also tracks in each particle the individual SOGP regressors as well.

4.2.1 Inference

During inference, a new datapair, $(\mathbf{x}', \mathbf{y}')$ is to be added to the model, updating the current distribution over functions. Because the members of $\mathbf{z}$ are discrete and finite, all possible values for z', the expert responsible for this new data, can be considered. ROGER thus initially temporarily assigns the datapoint to all possible experts, in all current particles. This enumeration results in $P_p = \sum_1^P K^{+(p)}$ putative particles being considered. Each possible assignment is given a weight using the joint:

$$P(z' = k|\mathbf{x}', \mathbf{y}', \mathbf{z}^{(p)}, \mathbf{X}, \mathbf{Y}; \Omega) \propto w_p * P(z' = k|\mathbf{z}^{(p)}, \alpha) P(\mathbf{x}'|\mu_0, \Sigma_0, \Lambda_0, \kappa_0, \mathbf{X}_k^{(p)}) P(\mathbf{y}'|\mathbf{x}', \hat{\mathbf{X}_k^{(p)}}, \hat{\mathbf{Y}_k^{(p)}}, \theta) \tag{4.15}$$

where w_p is the current weight of the parent particle and the other components are computed as in Equations 4.1. Note that although the input and output space distributions depend on all of the previously seen data, the actual data itself does not need to be kept. For the input space distribution, only sufficient statistics (the sum of squares and mean of Equation 4.9) need to be stored. Likewise, for the output space component, only β datapoints are contained in $\hat{\mathbf{X}}$ and $\hat{\mathbf{Y}}$, the basis set.

These putative particles then represent the distribution over the assignment of the new datapoint to an expert, taking into account the previous distribution over the number and content of experts as well. To limit the growth of the particle set, only P of these P_p putative particles are selected and reweighed for use in the next iteration. One approach to choosing the P particles would be to sample them directly according to their weight. This scheme, however, may result in the same putative particle being sampled multiple times, leading more towards an unweighted representation, and reducing the breadth of the particle set overall.

Instead, ROGER utilizes the optimal resampling technique of [48], which minimizes the expected error in expectations computed using a weighted particle set of size P downsampled from weighted particle set of size $P_p > P$. The method involves retaining a number of particles whose weights are above an optimal threshold c and using stratified resampling to resample from the rest. By choosing the threshold for retention of particles optimally, the particle set is guaranteed to have no duplications, or two or more particles with the same partitioning of the data. The overall inference algorithm for ROGER is shown pictographically in Figure 4.4.

4.2.2 Prediction

We take prediction as being the generation of an appropriate output, $\mathbf{y}'$, for a given input $\mathbf{x}'$. Viewed as a whole, ROGER's particle set represents a multimodal distribution of unknown order over functions supported by the data seen so far. However, for predictive purposes, it may be

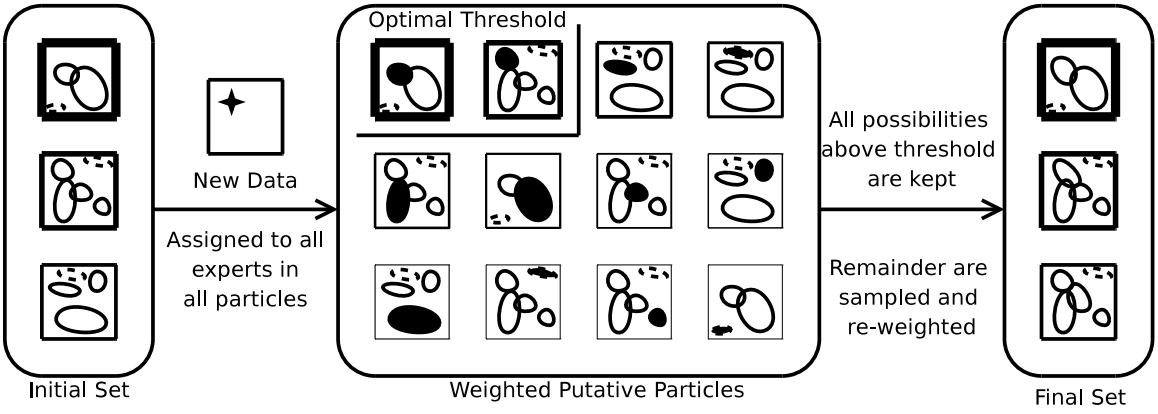


Figure 4.4: Inference in ROGER. New data is provisionally assigned to all possible experts, including previously empty ones, resulting in a set of putative particles. This set is sorted by likelihood under the joint, and a threshold value c is determined with the optimal resampling technique of [48]. All particles above the threshold are carried over, and the remainder are sampled from to fill out the new particle set. Data is only incorporated into each expert after the final assignments have been determined.

easier to consider it as instead a distribution over multimaps of unknown order. That is, instead of sampling a (unimap) function directly for prediction, we will first sample a multimap, and then use that multimap to sample a function.

Consider the particles themselves. Each particle is a sample from the partition space, both in terms of the number of partitions and the exact partitioning. While no two particles have the same partitioning, or all $\mathbf{z}^{(p)}$ are distinct, it is possible for multiple particles to share the same number of partitions, or for $K^{(p)}$ to be the same for several p. Choosing a particle is then choosing a partitioning that defines a set of possibly overlapping experts. For a given input point, multiple experts may be applicable, resulting in a multimap.

This multimap then defines a multimodal distribution over possible outputs for the given input. To generate an appropriate output we must sample from this distribution. One method would be to first sample a mode (unimap), and then sample the output from that mode.

This approach is exactly how ROGER performs prediction, as indicated in Figure 4.5: We first sample a particle, and then an expert from that particle, and finally sample from the distribution over outputs generated by that expert. However, each of these sampling steps can be accomplished in multiple fashions. Different schemes may result in different predictions being generated for the same inputs, even given the same set of particles.

To choose the predictive particle, p^*, we use the current weights of the particles as calculated during the last inference cycle. The most direct method, which we use, is to always select the particle with the highest weight, representing the most likely partitioning. A more correct method may be to select particles according to their weight. However, we have noted that the distribution over particles is often highly peaked, meaning that the most likely particle greatly outweighs the others,

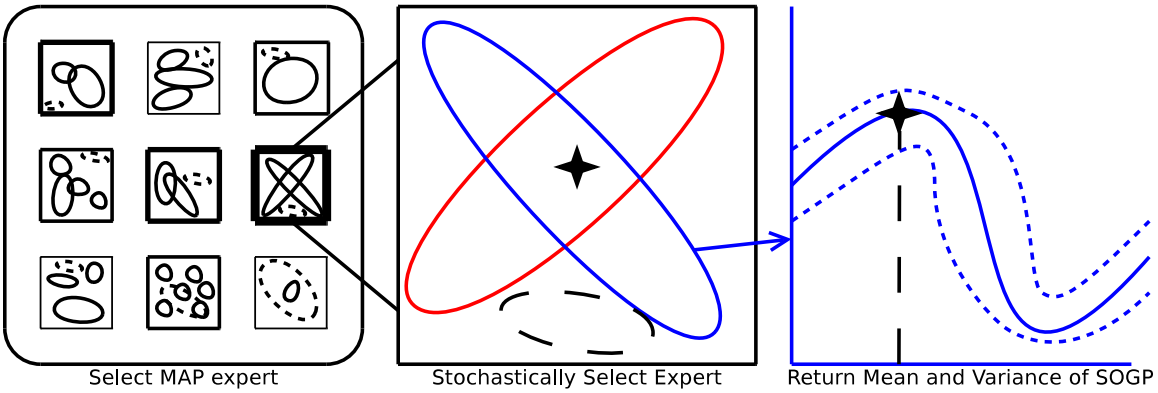


Figure 4.5: To predict in ROGER we first select the most likely particle, corresponding to the partitioning that maximizes the joint likelihood of the data. We then choose an expert stochastically based on the likelihood of the input point under the gating Gaussian mixture model. The output is the result of the expert's mean function, and confidence is given by the associated variance.

and would therefore almost always be chosen anyways.

Within the particle, we must choose an expert. Calculating $e_k = P(\mathbf{x}'|z' = k), k \in 1 : K^{+(p^*)}$, the likelihood of the query point being generated by each expert in the chosen particle, results in a distribution over experts. While the most likely option could be selected in all cases, as was done with particles, such a scheme would result in incorrect behavior in multimap scenarios. Recall that a multimap with two options has a mode for each. If the most likely mode is always chosen, one of the possible correct options will never be returned for the query point. Instead, the distribution over experts can be normalized, and the expert to be used (e^*) selected stochastically.

The expert itself is an SOGP regressor, and as such generates a Gaussian distribution over possible outputs $\mathbf{y}'$. Sampling directly from from the resulting distribution would introduce some additional variance in the outputs produced. Instead, ROGER returns the mean of the Gaussian as its prediction, and the associated variance as a measure of *confidence*. This confidence value is the one used during arbitration in the DL architecture. In effect, predictions for datapoints for which the predictive distribution is broader and flatter are deemed less confident than for those whose distributions are more sharply peaked and less variant.

One sampling scheme that we have not talked about is performing weighted averaging between the particles and/or experts. That is, predictions can be generated from all of them and then combined to generate the final outputs. Generally, this approach is undesirable, as it would sacrifice the multimap nature of the distribution. That is, two outputs from equally likely modes would be averaged together to create a potentially inappropriate output, as in Figure 4.2.

4.2.3 Batch Inference

As an alternative to the sequential particle filter based approach described above, we consider batch techniques to deriving the best partitioning of the data. One option is Gibbs sampling [94], where

changes to the current partitioning are proposed, and stochastically accepted. In terms of the CRP metaphor, this approach is to equivalent to taking an initial seating arrangement, and randomly evicting patrons from the restaurant, who then reenter and chose a seat as usual.

This sampling scheme is used by [89]. Overall, their technique differs from ours in three fashions:

1. Incrementality: We perform inference using a particle filter instead of Gibbs sampling.
2. Conjugate Priors: We place conjugate priors over the parameters of the input space partitioning, allowing us to analytically integrate over all possible Gaussians.
3. Sparsity: We use sparse experts (SOGPs) insted of GPs to limit the computation time and memory space of each expert.

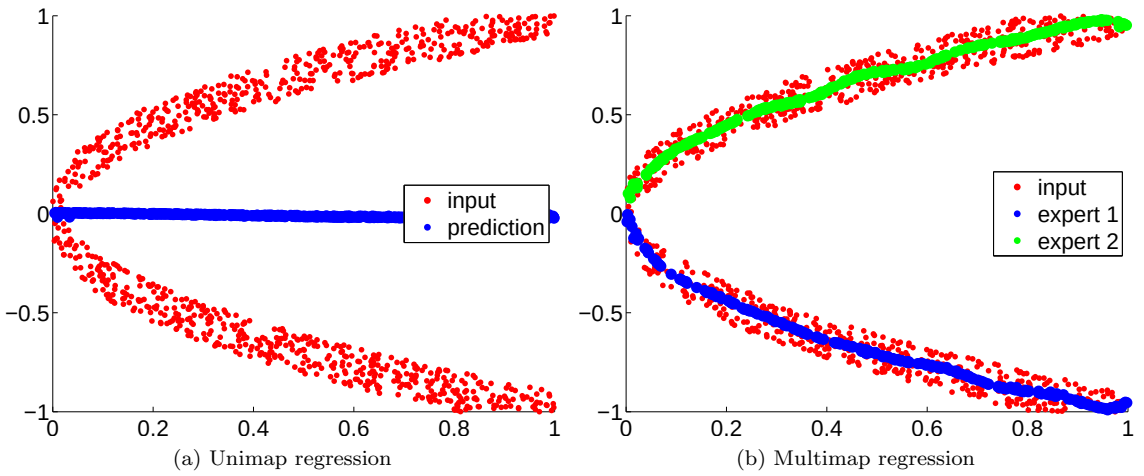
These three modifications are all aimed at enabling realtime learning from demonstration. First, by using a particle filter instead of Gibbs sampling, new data can be incorporated as it arrives, instead of requiring that all data be collected before performing inference. Secondly, the use of conjugate priors decreases the number of possibilities we must consider. Instead of each particle tracking both a possible assignment of data to experts, and the parameters for each expert, we only track the possible assignments, and integrate over all possible parameters. In this fashion ROGER can approximate the full joint probability with fewer particles, leading to sparsity. The use of sparse experts further sparsifies our technique, although if the SOGP capacity, β, is set high enough (or to infinity), SOGP and GP experts are equivalent.

4.3 Analysis

We intially experimented with ROGER on some non-robot data sets, drawn from standard functions and relevant work in machine learning. Our goal is to compare ROGER's performance with that of standard regression techniques and batch inference approaches. We are particularly interested in determining if ROGER can correctly learn multimaps, and what, if any, sacrifices or gains are made in using an incremental instead of batch approach.

4.3.1 Square Root Dataset

To validate ROGER's multimap learning capabilities, we used the the square root example shown in Figure 4.6. For training data we generate input (x, y) pairs where x is distributed uniformly at random in [-1,1] and $y = \pm\sqrt{x} + \epsilon$, a noisy square root mapping whose sign is uniformly random as well. We apply both unimap (SOGP) and multimap (ROGER) regression to the data and predict $\hat{y}$ for a further random set of x'. As seen in figure 4.6a, the unimap regressor averages both possible outputs, and predicts 0 for all x. ROGER, instead, automatically determines that there are two experts, assigns data to each, and learns separate models. At prediction time, one of the two experts produces an appropriate output, as seen in Figure 4.6b.



(a) Unimap regression (b) Multimap regression

Figure 4.6: The square root, a multimap. Unimap regression (a) averages the two outputs, while multimap regression (b) learns a separate function for each branch.

4.3.2 Incremental vs Batch

To compare incremental versus batch approximation techniques, we ran the particle filter and Gibbs methods on the synthetic dataset of [89] in Figure 4.7. This dataset contains both a multimap scenario and a gap with missing data, over which interpolation must be performed. We trained 500 particles incrementally, and also performed 500 batch Gibbs updates and compare the learned models. The results of the two techniques are very similar, both in terms of the MAP prediction of new outputs, and the overall distribution.

In addition, our experiments suggest that the incremental approach may perform better in resource constrained environments. Running both the incremental and batch formulations on this data, we varied the number of particles and sampling steps. Using this variable as a proxy for computational cost, the log probability of the discovered model is plotted versus computational cost in Figure 4.3.2. All data is shown averaged over 5 random seeds. While both approaches perform well with lots of computational time (batch slightly better), particle filters outperform batch processing when computational power is limited. As robots are typically resource constrained (in terms of memory and computational speed), algorithms that can perform in such environments may be more suitable for robot applications. Based on these results, we only use incremental inference in the rest of our experiments.

One drawback to the sequential approach to model selection is that once a particular assignment of a datapoint to an expert is present in all particles, it can never be undone. The batch method can escape from this situation by evicting the datapoint and reallocating it. This fact may lead to the slight underperformance of the sequential method at the right of Figure 4.3.2. It may then be advantageous to combine the two techniques. Such an approach may also better leverage available computational time as discussed in Section 3.5. That is, the incremental particle filter approach could

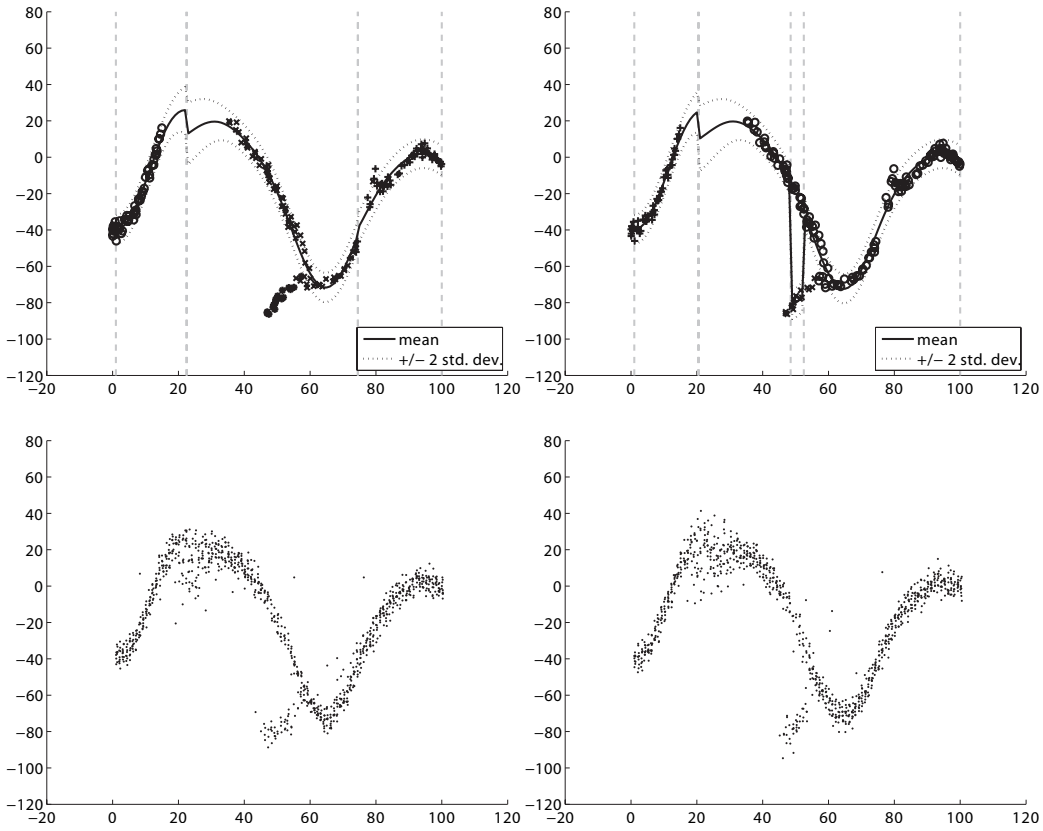


Figure 4.7: Top row: maximum aposteriori (MAP) models of synthetic data. The left figure shows the MAP model from 500 incremental estimation particles, the right figure from 500 batch samples. The black symbols are the training input/output observations and the vertical dashed lines indicate the areas of input space where each expert's likelihood is greatest. Bottom row: horizontally jittered samples drawn from the entire estimated posterior at regularly spaced input points for both incremental (left) and and batch (right) estimators.

be used to incorporate new data as it is generated with a small number of particles. Then, during prediction or other downtime, Gibbs sampling would be performed to counteract approximation issues that arise from the limited particle set.

4.3.3 Comparison with LWPR

We also compared against another popular robot learning algorithm, Locally Weighted Projection Regression (LWPR) [149] on nonlinear regression and multimap data. LWPR bears some similarity to ROGER, in that it is also an infinite mixture of experts technique. However, the method of determining how many experts, and performing regression in each of those experts, is very different. Further, experts are defined only in input space (outputs are not considered during assignment) and thus LWPR is a unimap regression algorithm. However, extensions that allowed for differentiation

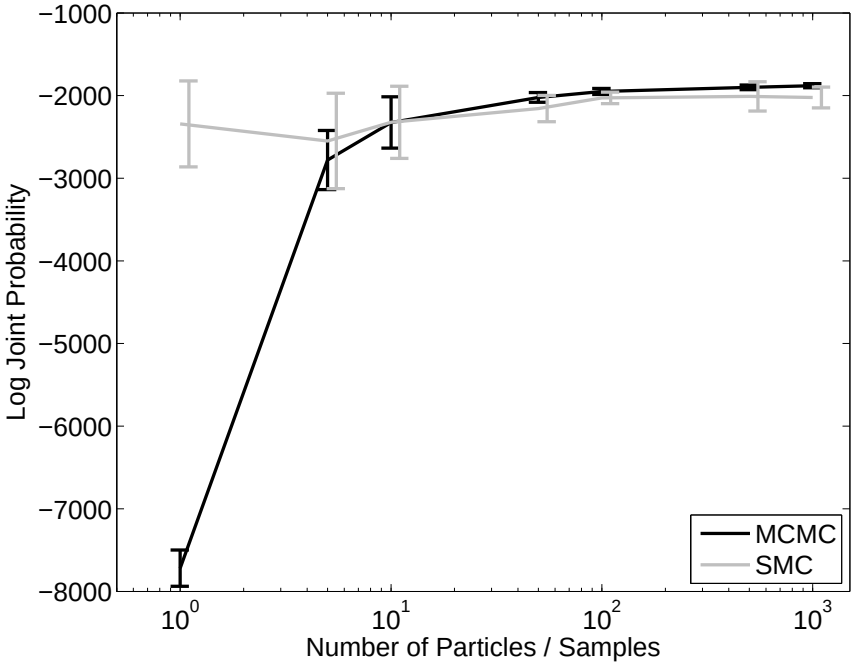


Figure 4.8: A comparison of Sequential (SMC) and Batch (MCMC) estimation in the ROGER model, showing negative log probability as a function of computational effort. In resource constrained settings, sequential techniques may outperform batch ones.

based upon the outputs as well might make it suitable for multimap regression.

LWPR is a local linear function approximation technique, in that it attempts to model a potentially nonlinear function with a collection of linear regressors. Each regressor or expert, called a Receptive Field (RF), has its own parameters and makes its own predictions. The final nonlinear prediction is made by weighing and combining the predictions from all of the RFs. LWPR also deals explicitly with high dimensional input spaces, as each RF projects data into a lower-dimensional subspace for processing. New RFs are created as needed to ensure acceptable prediction quality.

More formally, an RF is defined by a center point ($\mathbf{c}$) and a Gaussian area of influence described by a covariance matrix (D). The weight, or activation, of a point ($\mathbf{x}$) under this RF is calculated:

$$w(x) = \exp(-0.5(x - \mathbf{c})^\top D(x - \mathbf{c})) \tag{4.16}$$

In addition, each RF maintains a set of projection directions (U), correction vectors (to preserve orthogonality) (P) and linear regression coefficients (β) which are computed using Partial Least Squares (PLS) and used in prediction. Initially data is projected into 2 dimensions, so there are $R = 2$ of these in each RF, but R is increased automatically to adapt to the data.

The LWPR algorithm is summarized in Algorithm 4.3. When incorporating a new data point, RFs are generated based on a hand-set generation threshold, $w_{\texttt{gen}}$, which controls the amount of overlap between RFs. For prediction on a novel point ($\mathbf{x}'$), each RF computes a local prediction by projecting into a reduced dimensional space using U and P and regressing linearly with β. A

Algorithm 4.3 Locally Weighted Projection Regression

Inference	**Prediction**
Require: Training pair $(\mathbf{x}, y)$ K receptive fields (RF) generation threshold ($w_{\texttt{gen}}$) initial distance matrix (D^*)	**Require:** Query point ($\mathbf{x}'$) K receptive fields (RF) Blending boolean

Ensure: K updated receptive fields (RF)
for $k = 1 : K$ **do**
$w_k = \exp(-0.5(\mathbf{x} - \mathbf{c}_k)^\top D_k(\mathbf{x} - \mathbf{c}_k))$
Update local model $(D_k, \mathbf{u_k}, \mathbf{p_k}, \beta_k)$
if all $w_k < w_{\texttt{gen}}$ **then** {Create new RF}
$K = K + 1$
$\mathbf{c}_K = \mathbf{x}$
$D_K = D^*$
initialize $\mathbf{u_K}, \mathbf{p_K}, \beta_K$ with 2 dimensions.
	Ensure: predicted output $\hat{y}$, stddev σ^2
	for $k = 1 : K$ **do**
	$w_k = \exp(-0.5(\mathbf{x} - \mathbf{c}_k)^\top D_k(\mathbf{x} - \mathbf{c}_k))$
	Compute $(\hat{y}_k)$ via Equation. 4.17
	if Blending **then**
	$\hat{y} = \sum_{i=1}^{K} w_k \hat{y}_k / \sum_{i=1}^{K} w_k$
	else
	$\hat{y} = \hat{y}_j, j = \text{argmax}_j\, w_j$
	Calculate σ^2

loop-based algorithm for doing so is provided in [150], but is equivalent to a linear matrix operation:

$$\begin{aligned} \hat{y} &= \Lambda x' + \beta_0 \\ \Lambda &= \beta U^\top + \sum_{r=2}^{R} \beta_r u_r^\top \left(\sum_{s_1 \ldots s_{r-1}=0}^{1} -1^{\sum_{i=1}^{r-1} s_i} \prod_{i=1}^{r-1} (p_{r-1} u_{r-1}^\top)^{s_i} - I \right) \end{aligned} \tag{4.17}$$

Final predicted output can be performed in two manners. The first is a blended method, where the outputs of all RFs are weighted and combined. Alternatively, the prediction of the RF with the highest activation for the query point can be returned. The calculation of the confidence bounds (variance) at the prediction is similarly involved, and we refer the reader to [150] for full details. Roughly, it is a measure of the error of the prediction of each local expert from the global prediction, combined with the running error of each expert (computed incrementally) and weighed by each expert's activation.

As the training and prediction steps of LWPR can be alternated, it is suitable for tutelage. In addition, note that sparsity is achieved not only by creating RFs only as needed, but also by only storing the sufficient statistics (U, P, β) in each RF and discarding datapoints once they are incorporated. Lastly, the projection step has the potential to further reduce memory requirements by detecting and ignoring extraneous dimensions in the data.

Experiments

As LWPR is a unimap regressor, we compare it to the unimap regressor in ROGER, which is SOGP. A comparison of the two based on our desired attributes is shown in Table 4.3.3. Using the cross dataset of [149] we compare them quantitatively, the results of which are in Figure 4.9. A $\mathcal{R}^2 \rightarrow \mathcal{R}$ regression problem, the cross function has several nonlinearities that provide a good test for nonlinear function approximators. To make a more equal comparison between the two algorithms,

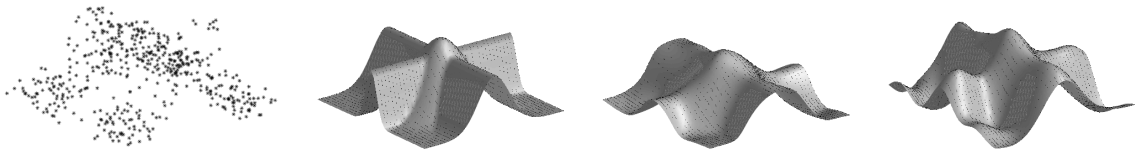
Attribute	LWPR	SOGP
Incremental	Yes	Yes
Sparse	RFs remain fixed once created	BVs are replaced as needed
Scalability	RFs created as needed	Fixed number of BVs
Noise Model	Gaussian	Gaussian
Approximation	Locally Linear	Globally Gaussian

Table 4.1: A comparison of attributes of LWPR and SOGP

we require that LWPR and SOGP have the same "computational capacity," which we control by limiting the number of local models in each. As LWPR automatically determines the number of receptive fields needed, we use that determined number (27) as the maximum capacity parameter (β) in the SOGP algorithm. Results indicate both techniques to be comparable in terms of learning capability. However, we note that we are running LWPR in a somewhat crippled fashion. That is, in order to get full benefit of the projection scheme, LWPR must be run over the complete dataset in multiple passes. For this 2D dataset, it is unnecessary, but on higher-dimensional robot data it may improve regression. However, doing multiple passes through the data sacrifices the incrementality and sparsity of the approach, as all of the data must now be kept, and the entire dataset reprocessed after each datapoint arrives. We choose not to run in this fashion and thus sacrifice any approximation advantages that could be gained.

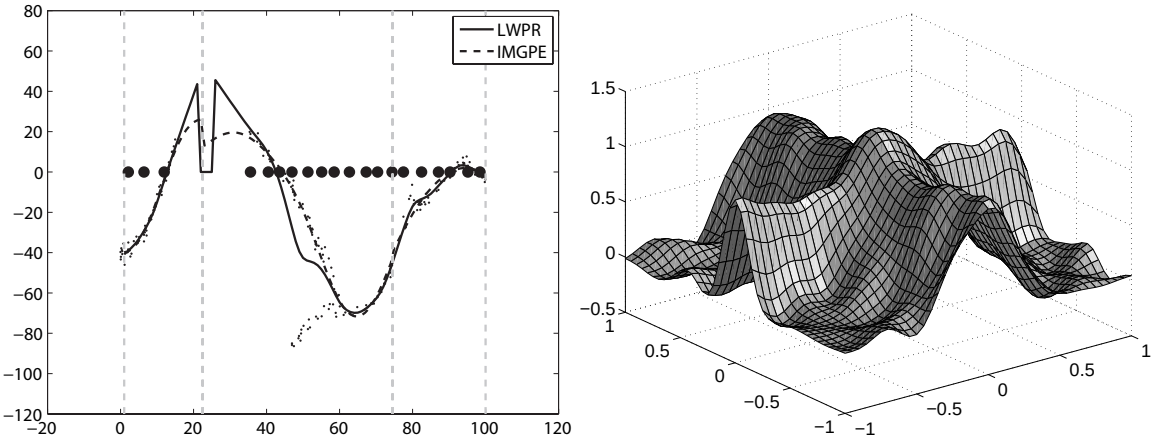
We also ran LWPR on the synthetic multimap, and ROGER on the cross data, the results of which can be seen in Figure 4.10. On the synthetic data, LWPR performs averaging in the multimap region, resulting in outputs that may not be appropriate. Note that the behavior of both ROGER and LWPR in the gap region is similar, where a discontinuity appears when the nearest local center (RF or BV) changes. The local linear nature of LWPR can be seen in the data surrounding this discontinuity as well.

As for the cross data, we point out that this is not a true test of the multimap regression algorithm. This data is a unimap, and therefore only one expert is needed to fit it. In that case, ROGER collapses to the SOGP algorithm, and obtains similar results. We further examine this phase shift in Table 4.2, where we summarize results from the cross and synthetic data sets, as well as analysis on the Boston data set from the UCI machine learning dataset repository [12]. This data



(a) Random Noisy Data (b) Ground Truth (c) LWPR, MSE = 0.0200 (d) SOGP, MSE = 0.0150

Figure 4.9: SOGP and LWPR compared on the 2D input, 1D output cross function. We limit SOGP's capacity to the number of RFs used by LWPR (27).



(a) Synthetic (b) Cross

Figure 4.10: LWPR compared with ROGER on the two test datasets. (a) A local linear model, LWPR requires more experts (dots) to fit the synthetic data, and as a unimap regressor averages the multimap region. (b) ROGER collapses to the SOGP algorithm on unimap data.

set has a 13 dimensional input space, higher than that of the synthetic and cross data, to give a sense of how the algorithms will scale. Results are shown in terms of mean-squared error and indicate that on unimap regression, LWPR and ROGER perform similarly (and thus, by extension, so do SOGP models). However, on multimap data, ROGER significantly outperforms LWPR. When calculating MSE on multimap data, error is taken as the distance to the closer of the multiple possible outputs.

4.4 Discussion

We have developed ROGER with an eye towards performing direct policy approximation for robot control policies from interactive tutelage. However, there are other techniques that can be used to address the underlying multimap regression problem. Specifically, while we have a particular generative model for the data and a particular approach to model selection and subtask policy learning, other choices are possible. Since we have factored our joint and separated the input and output space distributions, it should be possible to change one or the other, assuming appropriate methods for calculating the required values.

For example, our output space distribution is currently Gaussian, as modeled by our SOGP

Dataset	LWPR	ROGER
Boston [12]	73.1	68.5
Cross [119]	0.017	0.004
Synthetic [89]	92.9	22.2

Table 4.2: Comparison of LWPR and ROGER on various datasets. Shown are average mean squared error of predictions on held-out data.

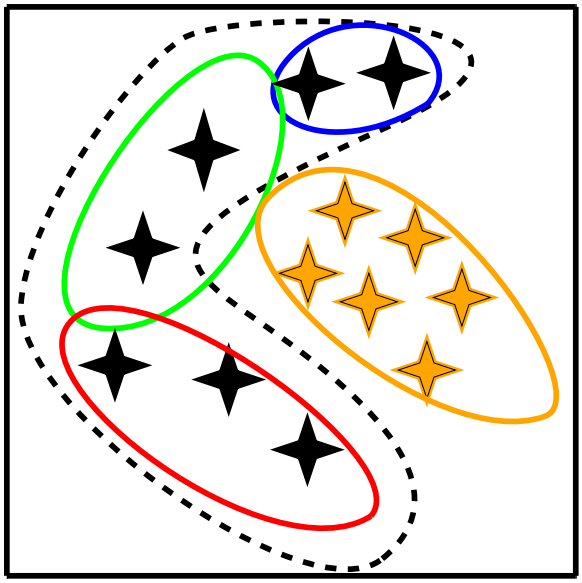


Figure 4.11: With only Gaussian-shaped experts, ROGER may need several (red, black and green solid lines) to fit data originating from one subtask (black stars), when the distribution is non-convex. Using an alternate input model may allow for one expert (dashed black line) to be used instead.

experts. Other regressors, such as the partial least squares regressors of LWPR, or LWPR itself, might be appropriate, as well as making different assumptions about the observation noise. To use one of these, or other, regressors in our model, we would need to be able to calculate $P(\mathbf{y}|\mathbf{X})$, as in Equation 4.14. The other portions of the joint would remain unchanged.

Likewise, we currently use a Gaussian mixture model to assign regions of input space to experts, meaning that the input for a particular subtask must originate from a single Gaussian area of perception space. Making this assumption maybe overly limiting, as it means that a subtask with a non-Gaussian input region would have to be fit by multiple Gaussian shaped experts, as shown in Figure 4.11. One possibility that we have started to consider is letting the input gate for each expert be, itself, a mixture of Gaussians, so that each of these component Gaussians would be associated with the same expert. Alternatively, some other density estimator could be used to generate non-Gaussian gates. Again, only the a portion of the full joint, the input space distribution, would change, the rest of the calculations would be untouched.

4.4.1 Model Selection

One of the touted advantages of ROGER is its approach to model selection, or determining how many experts are represented in the data. By maintaining a distribution over the partitioning of the data, ROGER tracks models with different numbers of experts, up to the possibility that each datapoint comes from its own expert (effectively infinite experts). Further, ROGER also represents variability between multiple models with the same number of experts, by considering different assignments of

data to each expert.

Like the input and output models, the distribution over partitions can be changed independently of the rest of the joint. Rather than using the CRP, a different approach to determining an appropriate number of experts would be to calculate the likelihood of the data (inputs and outputs) under models with varying numbers of experts. To avoid overfitting, models can be penalized according the the number of free parameters, using, for example, the Bayesian Information Criterion:

$$BIC = -2\ln(P(\mathbf{X}, \mathbf{Y}, \mathbf{z}) + K^* \ln(N)$$

where K^* is the number of free parameters. While related to the number of experts, it also depends on the dimensionality of the data, and the exact parameterization of the experts themselves.

To give infinity the same sort of consideration as ROGER, the proposed BIC-based approach would have to run EM to fit a finite mixture model to the data for models of all orders from 1 to N. The likelihood of the resulting data would be calculated and penalized, and the resulting most likely model would give us the "optimal" number of experts to use. While described here as a batch approach, the models could be trained incrementally, with a new model being added each time a new datapoint was received.

However, when training a mixture model with EM (particularly incrementally), there's a possibility of the algorithm getting stuck in local optima. To counteract this, starts from multiple random seeds are often used. This BIC-based approach can then be seen as filling in a large table of probabilities, of size $N \times R$, where R is the number of random seeds. For computational efficiency with large N, a coarser granularity (1,5,10..., as opposed to 1,2,3,4...) may be necessary, which could result in the "true" optimal number of experts being missed. Techniques such as binary search over the number of experts may address this concern, but can still be considered a "brute-force" method.

In contrast, ROGER's current approach to model selection can be seen as filling in only P cells of this $N \times R$ table, where $R \to \infty$, and approximating the entire distribution over the number of experts from these cells. Further, ROGER considers the distribution over all possible partitions of the data for a given number of experts, while the finite approach above tracks at most R. Also, prior information as to the partitioning (specifically as to the rate at which new experts appear) is possible with the CRP, but not the BIC technique. And lastly, by tracking the entire distribution over partitions, ROGER leaves the door open for changing the assignment of points to improve modelling and escape from local optima, using perhaps the MCMC approaches as discussed in Section 4.3.2.

4.4.2 Temporality

Currently, the temporal nature of the data is not used effectively in determining the partitioning. While ROGER says something about the expected number of experts over time, each datapoint itself is considered independently, without regard to the experts to which its temporal neighbors were assigned. Theoretically, the complete dataset can be randomly permuted with no effect on the discovered partitioning. In practice, this is not the case as both the sparsity of the experts (where data is discarded) and the finite number of particles introduce order-dependent effects.

For tasks like the square root of Section 4.3.1, where the data come randomly from both experts, ignoring temporality is appropriate, as the temporal relationship between points has no further information. However, for robots, we do not expect this to be the case. Instead, we expect data from a latent FSM robot controller to have temporal continuity, where contiguous subsections of the data all come from the same expert. Our current model relies on the fact that data from the same expert occupy similar positions in input-output space to group them together.

Likewise, when controlling the robot (performing prediction), we would expect to sample repeatedly from the same expert for some time, until the subtask it represents is complete, or interrupted. Then we would begin to draw from another expert, and so on. Our current model does not incorporate this idea, and instead chooses the current active expert independently of the one before it. In terms of robot behavior, this may lead to rapid oscillation between subtasks. For the case when the subtasks represent two different approaches to the same problem, as in Figure 4.2, rapidly switching between them may result in the robot effectively averaging their commands, due to physical inertia.

This issue, of ensuring expert continuity, is tied to that of inferring transitions in an FSM built over the experts discovered by ROGER. One approach to ensuring similar temporal structure between inference and prediction is to look at the expert assignments as discovered by ROGER itself. From the sequence of experts over time we can determine the transition probabilities, and use those during prediction.

However, this method still does not leverage the temporal information in the data. For a temporally aware ROGER (T-ROGER), it might be better to change the distribution over partions, to favor those where temporal continuity is perserved. One technique, discussed in Section 6.2.2, might be to change the CRP evolution equation, so that the probability of sitting at the same expert as the previous patron is increased, as in the "sticky-HMM" [49].

4.5 Review

We have introduced ROGER, an incremental approach to multimap regression. It seeks to decompose a multimap (where one input can have multiple appropriate outputs) into a set of unimaps, inferring the appropriate number from the data. For prediction, outputs come from only one of the applicable unimaps, resulting in multimap behavior, instead of averaging.

We have compared ROGER to a variety of other regression techniques, the algorithms discussed in this chapter are laid out in a plot in Figure 4.12. As stated, there are a number of desirable properties we seek in a learning algorithm for our interactive robot learning scenario. On the horizontal axis we have "computational feasibility," a rough measure of how appropriate the algorithm is for tutelage-based RLfD. This feature is mostly related to the speed of the algorithm, but also incorporates ideas of sparseness, incrementality, and scalability. We indicate a shift from techniques that store all of the data seen on the left, to those that learn sparse models on the right.

The vertical axis of the graph indicates the quality of estimation. This metric includes not only the accuracy with which the the algorithm learns the underlying policy, but also its ability to

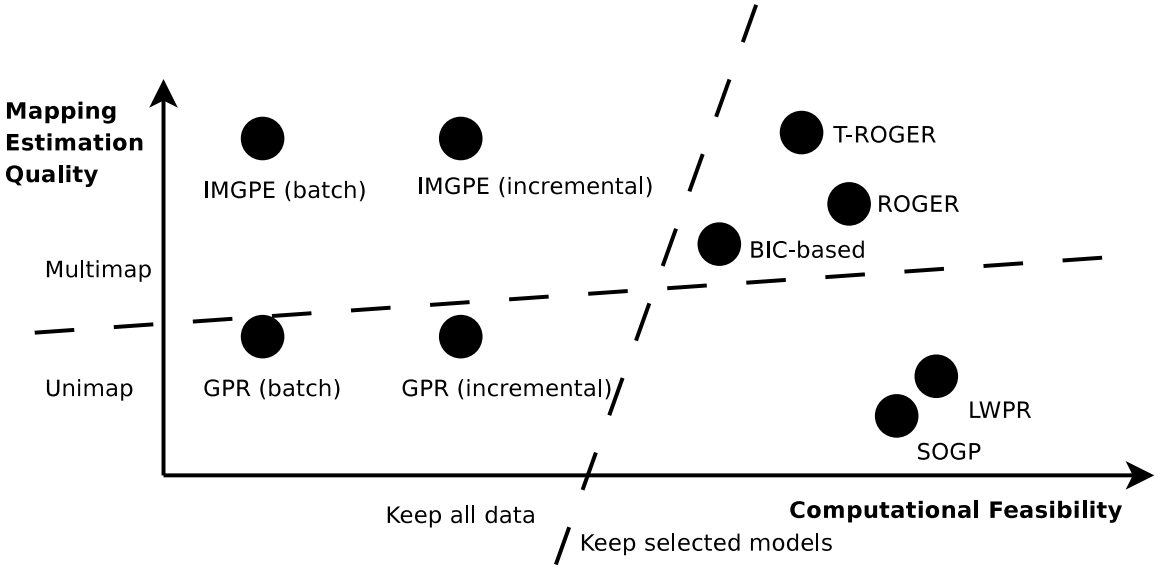


Figure 4.12: Algorithms discussed in this section, compared with respect to computational feasibility (speed, sparsity, incrementality, scalability) and estimation quality (accuracy and confidence). For robot learning of unknown tasks from tutelage we seek a sparse multimap algorithm such as ROGER.

determine when the estimate is good or not (confidence). We divide these algorithms into those that learn unimap policies, and those that learn multimaps. ROGER and its possible temporally conscious successor occupy locations in the top right of the graph, as they are sparse, incremental multimap regressors.

Looking forward, the next necessary step is to include temporal information in the model and algorithm. In terms of finite state machines as introduced in Section 1.3.4, ROGER currently only performs model selection and policy learning, determining the subtasks of an FSM. Leveraging the temporal information will enable it to determine the transitions as well, and perform full FSM learning. As FSMs are used in a variety of domains, such map-building [44], ROGER may then be applicable to other areas, not just robot learning from demonstration.

Chapter 5

Evaluation

> *Games are deep in the heart of us. From solitaire to the Super Bowl we're nourished on games, those abstract expressions of real life where we know the rules and can test our wits against an opponent or against chance, or watch our agents do it for us. Real life, of course, is never that tidy. Games let us work up to life.*
>
> Pamela McCorduck, Machines Who Think, 1979, page 146

We wish to learn, from demonstration, robot control policies similar to those that are today currently hand-coded. The previous two chapters have introduced our robot tutelage architecture, as well as several learning algorithms for use in therein. In this chapter, we set forth our experiments in learning a robot soccer team, like those that have been programmed for the Robocup[1] competition.

In the Robocup standard platform league, teams of identical robots compete in robot soccer games. Until recently, the platform was the Sony AIBO, which we use here. By requiring all teams to use the same robots, winning is thus framed as a problem solely of software: algorithms, design, and development. While advanced teams consider the set of robots as a whole and use techniques such as coordinated plays [27] and dynamic role allocation [129], first year teams often consider each robot in isolation and utilize little, if any, inter-robot communication. A standard technique, used for example in [81], is a so-called "swarm team," where each robot, except for the goalie, individually attempts to get the ball and score with it. It is called a swarm team because the resulting behavior is that all of the robots "swarm" around the ball, fighting for control. As all of the fieldsbots run the same controller, only the goalie needs to be developed separately.

Despite the simplicity of the strategy, such a team can still take months to develop. While much of that time is usually spent on lower-level development such as vision, localization and locomotion, a fair portion is dedicated to the development of the high-level behaviors. It is the development of the behavior, given low-level perception and actuation, that we are focused on here. Our platform, showing the assumed perceptual capabilities and actuation modalities discussed in Section 3.4.1 is shown in Figure 5.1.

[1] http://www.robocup.org

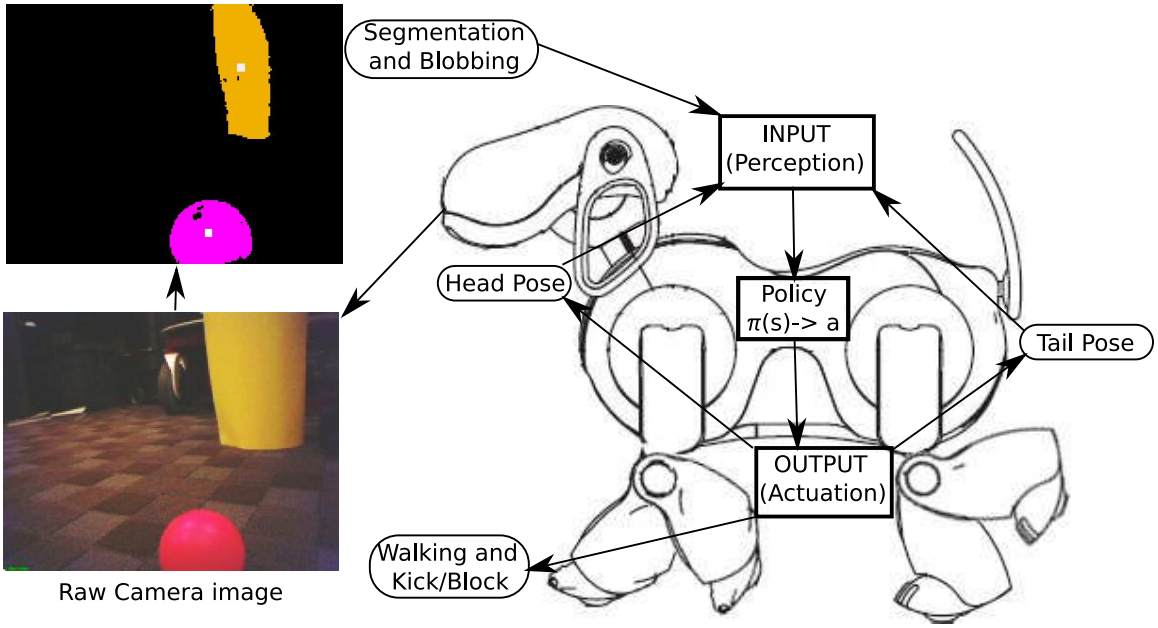
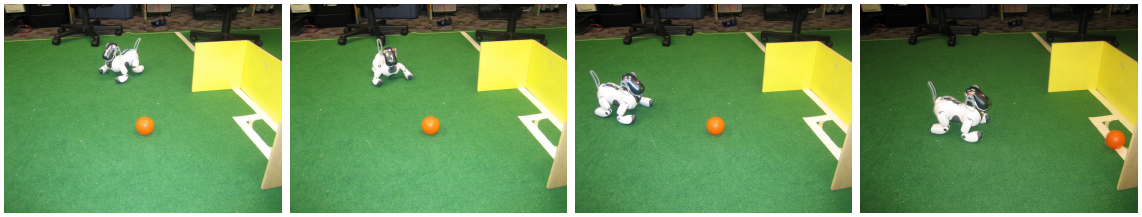


Figure 5.1: Our evaluation robot platform, a Sony AIBO equipped with rudimentary vision (color segmentation and blobbing) and walk-gait generation. Control policy estimation involves approximating the observed policy mapping from state estimate INPUTs to action OUTPUTs.

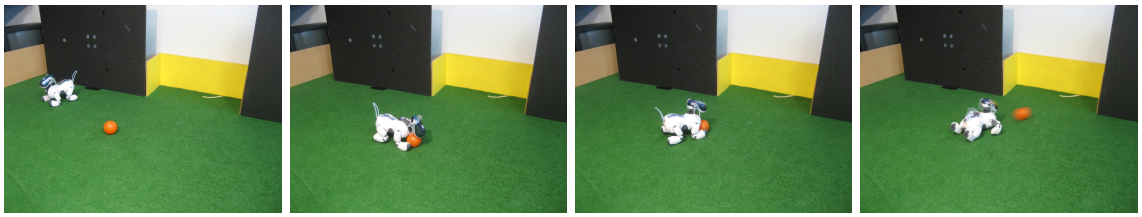
Figure 5.2 shows simplified versions of the swarm-style robocup soccer behaviors. Specifically, we are ignoring concerns as to other players on the field, the ball-holding penalty limit of three seconds, and global localization on the field. We also present two different approaches to goal scoring. The first policy, in Figure 5.2a, is designed to be amenable to unimap regression. To be so, we avoid ambiguous sensor states, or perceptual aliasing, which may lead to incorrect behavior when standard regression is used to directly estimate the control policy from demonstrated perception-actuation pairs. The second approach, in Figure 5.2b, is more representative of the control policies that were developed for the Robocup competition in its early years. However, on our platform this more effective policy contains perceptually aliased state estimates, resulting in multimap scenarios.

To collect data to train these behaviors, we experimented with three different approaches. First, a standard batch approach, where all training data is collected before learning takes place. Second, an interactive approach, where a human user toggles an HGC until they are satisfied with the learned controller or believe it to no longer be improving. Lastly, learning from interactive human demonstration, where the human user teleoperates the robot to perform the task, again until satisfaction or no additional improvement.

The evaluation of the learned policies themselves can also take multiple forms. Qualitatively, a human observer can watch the learned controller in action, and subjectively decide if it is performing the task correctly. More quantitatively, we can use task-level metrics such as the number of goals scored (or blocked) in a given time period, of from specific locations. We can also evaluate the approximate policy at the action level, by comparing the commanded actuation outputs with those



(a) Unimap Fieldsbot



(b) Multimap Fieldsbot

(c) Goalie

Figure 5.2: The robot soccer behaviors we learn herein. Unimap goal scoring (a) specifically avoids perceptual aliasing, to enable learning with standard regression. Learning the more successfully policy in (b) requires multimap regression, such as ROGER. The goalie (c) rounds out our swarm-style team.

of the demonstrator, and calculating the mean squared error.

For task-level evaluation of the policies, we use 13 locations, distributed across the field as in Figure 5.3. The exact locations of the ball and robot in each trial is corrupted by additional placement noise. We ran each of the coded policies (HCC) on these locations for 2.5 minutes and computed their effectiveness, in terms of the percentage of goals scored or blocked. The results are shown in Table 5.1, along with those of learned controllers. To give a sense of the capabilities of each controller, we also indicate the results from each of the 13 test locations.

Recall that our goal is to learn policies that perform as well as the demonstrator. From this summary, we can see that SOGP can be used to learn the two unimap policies, approaching to within 10% of the efficacy of the demonstrator. However, on the multimap policy, SOGP fails to develop a useful approximation. With ROGER, the resulting policy is still not at the level of demonstration, but it is much improved over the one learnt by SOGP. We now provide more details and analysis of our experiments in learning these controllers.

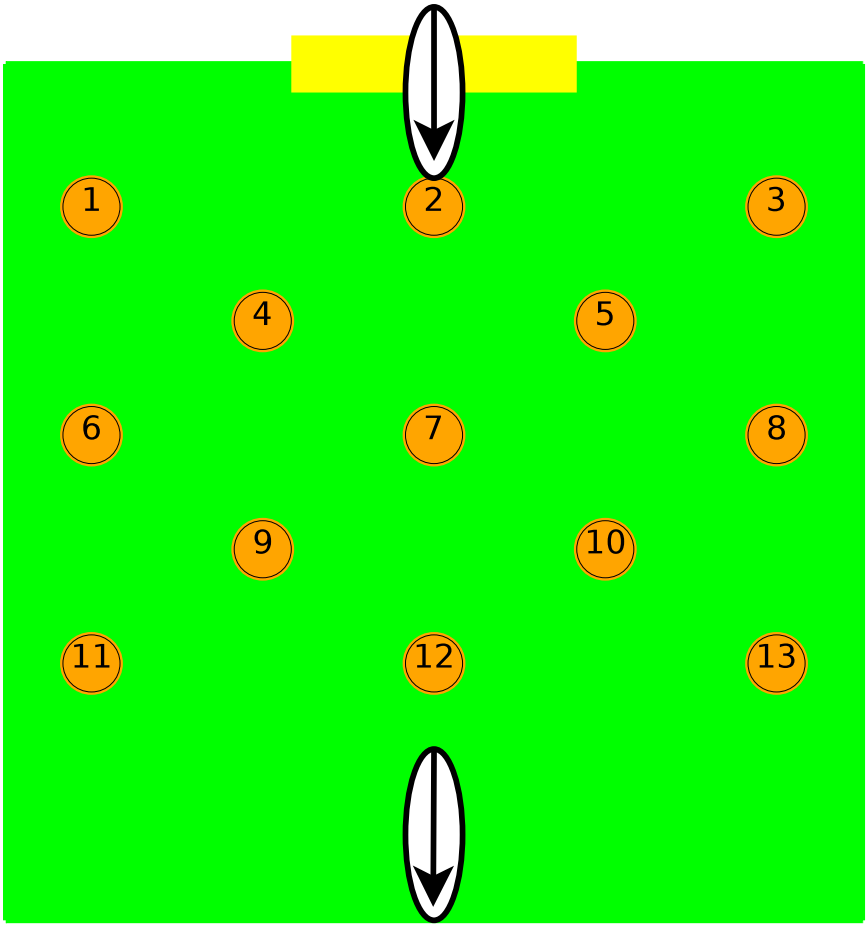


Figure 5.3: The field setup for training and testing our soccer team. Not to scale, the yellow square is the goal, orange circles are initial ball positions, white ovals are initial robot locations/orientations.

Location	1	2	3	4	5	6	7	8	9	10	11	12	13	%
Unimap Scorers														
HCC	0/3	3/3	0/3	2/3	1/3	0/3	3/3	0/3	3/3	3/3	0/3	3/3	0/3	46
SOGP	0/3	3/3	0/3	0/3	0/3	0/3	3/3	0/3	3/3	3/3	0/3	3/3	0/3	38
Multimap Scorers														
HCC	1/3	3/3	0/3	2/3	3/3	3/3	3/3	2/3	3/3	1/3	2/3	3/3	1/3	69
SOGP	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0
ROGER	0/3	2/3	0/3	1/3	2/3	1/3	2/3	0/3	1/3	0/3	1/3	2/3	0/3	31
Goalies (Unimap)														
HCC	2/5	4/5	4/5	4/5	5/5	3/5	3/5	2/5	5/5	4/5	4/5	1/5	4/5	69
SOGP	3/5	5/5	4/5	3/5	5/5	3/5	1/5	5/5	2/5	2/5	2/5	0/5	5/5	62

Table 5.1: Comparison of the efficacy of the various robot soccer tasks, both as coded and learned. In addition to percentages of success, we show the results of multiple trials from each of the 13 locations in Figure 5.3 in terms of successes (scores or blocks) over attempts. On unimap controllers, SOGP successfully learns policies, but it fails on the multimap policy, where ROGER is needed.

Algorithm 5.1 Unimap Goal Scorer (UGS)

```
Require: Perceptual variables BALL and GOAL
Ensure: Action output ACTION
  loop
    Update BALL and GOAL
    if isLinedUp(BALL,GOAL) then
      if isKickable(BALL) then
        ACTION ← "kick"
      else
        ACTION ← "approach ball"
    else if isVisible(BALL) AND isVisible(GOAL) then
      ACTION ← "sidestep"
    else if isVisible(BALL) then
      ACTION ← "circle"
    else
      ACTION ← "spin"
```

5.1 Unimap Goal Scorer

Given that we will be using unimap regression techniques, we specifically develop a unimap goal scoring policy (UGS), shown in Figure 5.2a and in pseudo-code in Algorithm 5.1. Conceptually, we can think of the controller as having four stages. The first stage is the ball-location stage, and rotates the robot in place until the ball is in view. Then, the robot walks around the ball (to the right in a circle with the ball at its center), until the goal is in view behind the ball. The robot then approaches the ball, and when in range, kicks. During execution, if the ball or goal should be moved, the robot immediately adapts, switching stages if necessary.

While we can think of this policy as having four distinct steps, in actuality the control algorithm is a set of nested if-else loops, with no internal state. That is, the current inputs (observed ball and goal location) are all that is necessary to determine what the outputs (walk parameters and kick) should be. It is this direct, reactive mapping, that enables the immediate adaptation.

Testing this control policy on the 13 ball locations of Figure 5.3, we find it 46% effective. The policy has the most trouble from positions near the edge of the field, where the sharp angle between the ball and the goal makes it difficult to line the two up. Additionally, for positions on the left side of the field, the robot must circle all the way behind the ball before it can be lined up, taking more time and increasing the chance that the robot will become stuck on the way (on the wall or carpet).

Using SOGP ($w_k = 0.1, \sigma_0^2 = 0.1, \beta = 300$), we train a controller with data from one shot from each of the locations. Because we use an SOGP, this is *not* equivalent to storing all of the data for future comparison. Instead, only $\beta = 300$ of the $\sim 28,000$ total points are kept to represent the distribution over mappings. Testing the learned controller from each location, we find it 38% effective, just slightly worse than the HCC. Experiments with interactive training obtained similar results, although we had decreased effectiveness when learning directly from human teleoperation, most likely due to increased noise and errors in the demonstration data.

5.1.1 Conclusion

Using unimap regression, we are able to transfer autonomous control policies onto robots with fixed perception and actuation, from both hand coded controllers and interactive human teleoperation, where the learned policy performs nearly as well as the demonstrator. However, the policy discused here is a unimap policy, devoid of perceptual aliasing. In the next section we will examine a multimap control policy that is more representative of those actually programmed by first-year robocup teams, and which will require multimap regression to learn.

5.2 Multimap Goal Scorer

The multimap goal scorer (MGS), shown in Figure 5.2b, can also be thought of as having four stages. In the first stage, the robot turns in place to locate the ball, similar to the UGS controller, but then approaches the ball directly, without regard to the goal location. Once the ball is reached, the MGS controller executes a trap motion to put the ball under the robot's chin, so that it can be manipulated. The third stage turns with the ball towards the goal, and the last stage checks the goal's location and kicks if it is lined up.

In contrast to the UGS controller, where the 4 stages were only conceptual distinctions, the MGS controller actually consists of four separate subtask controllers, arranged in an FSM as shown in Figure 5.4. Thus, in addition to the subtask controllers themselves, we specify transition conditions, indicating when the active subtask should change. These conditions impose an amount of inflexibility on the MGS controller, as it cannot always change stages immediately, as the UGS controller does.

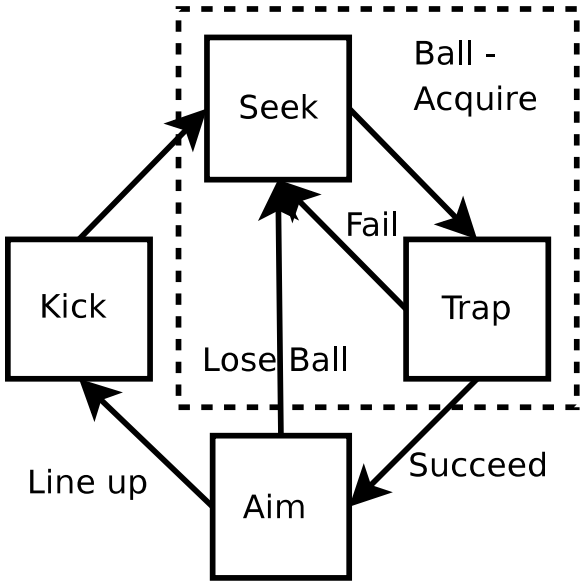


Figure 5.4: The multimap goal-scoring (MGS) task as a finite state machine. Without knowledge of the correct subtask to perform, perceptual aliasing occurs.

For instance, if the ball, robot, or goal are moved during the trap stage, the complete trap motion must still be finished before transitioning to a new subtask.

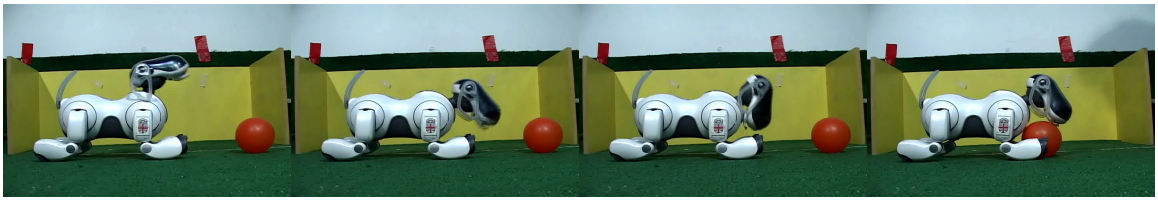
Our initial attempts to learn this behavior with unimap regression failed. That is, the task could not be taught to user satisfaction in half an hour of interactive training using a HGC. Instead of proper task performance, we observed the robot performing portions of the different subtasks at inappropriate times. For instance, whilst navigating to the ball, the robot would sporadically trap. We hypothesize that in some portions of the perception space there is not enough information to correctly determine the appropriate action. That is, perceptual aliasing causes outputs from multiple subtasks to be combined, resulting in the observed incorrect behavior.

To test this hypothesis, we focused on learning the first two individual subtasks in isolation, shown in Figures 5.5a and 5.5b, without the transitions between them, as well as their combination in Figure 5.5c. We also examined some related tasks, shown in Figures 5.5(d-f), to further probe the utility of the DL architecture and the learning algorithms we can use therein. A summary of the tasks, brief descriptions, and the two-letter abbreviations by which they will be referred are in Table 5.2. In the experiments presented in this section, we used the same parameters for the algorithms across all tasks: LWPR ($w_{\mathtt{gen}} = 0.2, D^* = 100$) and SOGP as above.

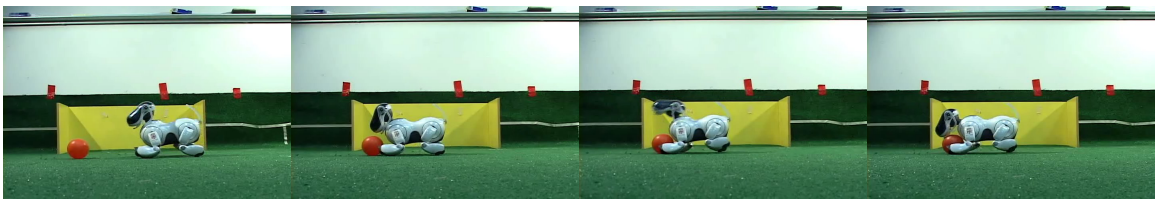
Task	Name	Description
BA	Ball-Approach	Rotate in place until the ball is seen, then walk towards it. While walking, lower the head to keep the ball in view, and stop when the ball is directly under the nose.
TR	TRap	Lift the head fully, and attempt to "scoop" the ball under the chin. Using the mouth, detect if the ball is locked in place. If so, stop, otherwise attempt to trap it again.
AQ	ball-AcQuire	A combination of the above two tasks. Approach the ball, and when stopped, execute the trap.
AQ+	AQ with state	Same as above, but the perception and actuation space of the platform have been extended to explicitly indicate which of the subtasks is being executed
HT	Head-Tail	Using only the motor sensors, move the head to mirror the tail.
BT	Ball-Track	Move only the head to keep the orange ball centered in view.
GC	Goal-Charge	Rotate in place to locate the goal, then approach it and stop when it fills the view.
WK	WalK	Execute a cyclic walk gait to move the robot forward. This motion pattern is taken as part of the platform.
KI	KIck	Execute a kick by bumping the chest into the ball. This predetermined motion is taken as part of the platform, as is a block.

Table 5.2: The various tasks discussed in this section, where they are referred to by their 2 letter abbreviation. Ball-Approach is the same as the seek subtask of the multimap goal scorer.

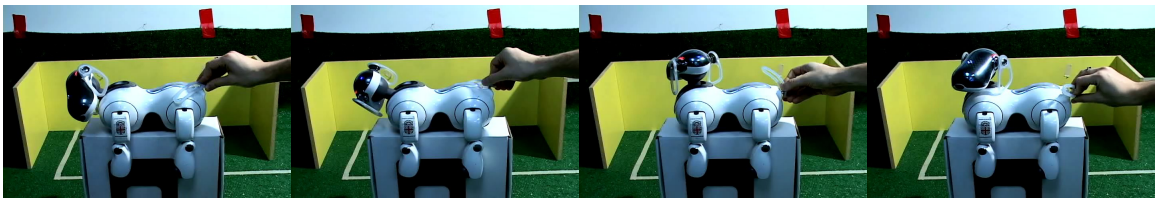
(a) Ball Approach or Seek



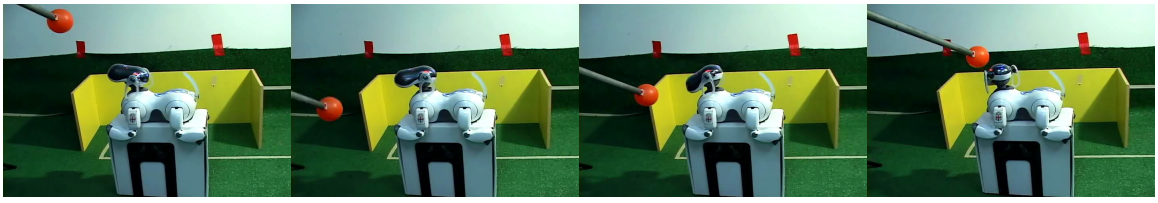
(b) Trap



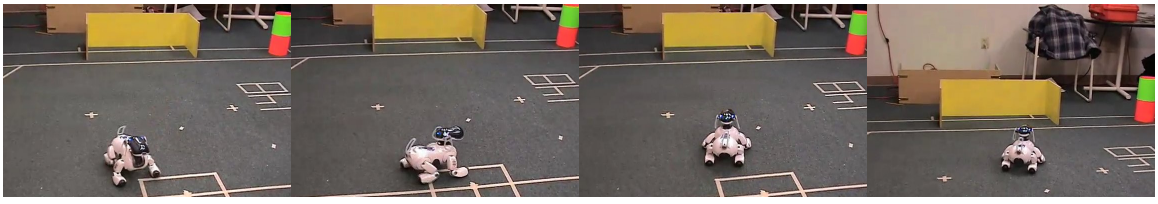
(c) Ball Acquire = Seek + Trap



(d) Head-Tail mirror

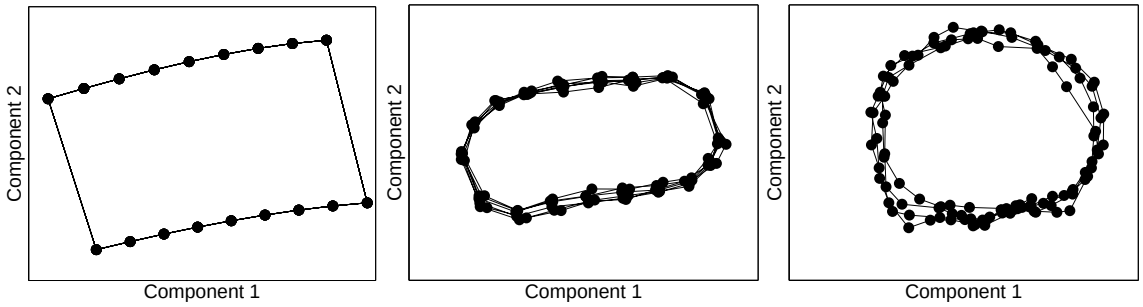


(e) Ball Track



(f) Goal Charge

Figure 5.5: The Goal Scoring subtasks and some related skills learned. Not shown are the walk (WK) and kick (KI) skills, which are initially learned, and then taken as part of the platform.



(a) Demonstrated (b) Observed (c) Learned

Figure 5.6: Learning the Walk controller. The HCC demonstrator generates the open loop sequence shown in (a), giving rise to the observed data in (b). The learned controller results in the sequence in (c). All data are projected onto the first 2 principle components of the motor space.

5.2.1 Open Loop

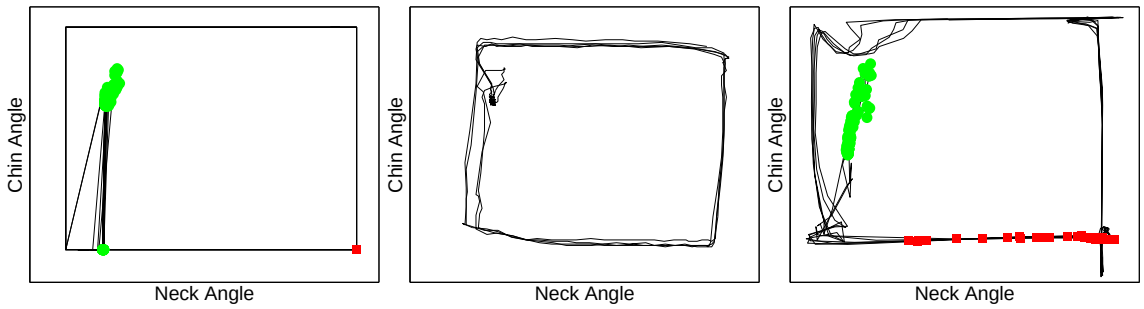
Two of the tasks in this section, WK and KI, are not learnt from human demonstration, because the teleoperative interface does not allow for them to be produced. Both of these tasks involve coordinated movement of the robot's legs. For this reason, we take them as given and have included them as actuation options in our platform. However, before doing so we first showed that they were, themselves, learnable from demonstration.

Both of these tasks' HCCs are actually open loop controllers, where the commanded motor positions are unrelated to the perceptual input. In Figure 5.6 we show results from learning the walking task. While the open loop HCC generates the same commands each cycle, the learned controller is more reactive to the environment, and generates more varied data.

Trap

Similar to the above actuations, the TR subtask involves a pre-determined sequence of motor positions. Designed to give the robot control of the ball, the technique is to lift the head and chin up, reach the chin out, and pull the ball back so that it is trapped against the robot's chest. We considered including the trap motion as an intrinsic capability of the platform, but chose instead to learn it, as our teleoperative interface allows for humans to demonstrate the entire behavior.

The data space is 2D, corresponding to the two motors of the head (neck and chin) that are used, and is shown in Figure 5.7, along with the commanded, observed, and learned trajectories. Like WK and KI, TR's controller provides the same commanded motor positions each cycle, and the observed perceptions are smoothed by physical forces. The learned trajectory falls in between. We also indicate trap success (green) and failure (red), which will be required for transitioning from the trap into one of the other subtasks.



(a) Demonstrated (b) Observed (c) Learned

Figure 5.7: Learning to trap. Similar to walking, we show the open-loop commands (a), observed motor positions (b), and learned controller (c). The space is the 2 used motors of the head, along with a success/failure indicator.

5.2.2 Mean Square Error

The other tasks, such as the BA and GC tasks, are fully closed-loop controllers, where the generated outputs at each timestep depend on what is actually perceived. In these two cases, the robot turns in place until the object of interest (ball or goal) is detected, and then walks forward. In the BA task, the robot's head and neck track the ball, eventually pointing directly downward at it, at which point forward motion ceases. For GC, the robot's head does not move, but the robot stops approaching the goal when it has reached a threshold distance (as determined by the size of the visible goal).

While learning these tasks, we also introduce two simple closed-loop controllers as test cases. The HT task ignores all visual input, and only controls the head's pan and tilt to match that of the tail. The BT task, on the other hand, is a visual servoing task, where the robot's head moves to keep the ball centered in view.

Learned controllers for all of these subtasks (HT, BT, TR, BA, GC) were generated with LWPR and SOGP from HGCs and teleoperation, save for GC, which we used as a "pure" test of the human teleoperative learning aspect. Quantitative results in terms of MSE and learner confidence are in Figure 5.8. All results are shown averaged over 5-folds, where we train each controller on 4/5ths of the demonstration data (collected in batch) and test it on the remaining fifth. LWPR, overall, has higher MSE, which may be reduced by parameter tuning.

5.2.3 Features of Learning Algorithms

As discussed in Chapter 4, two desirable features of learning algorithms for robot tutelage are speed and robustness to noise. Particularly, we want the algorithms to maintain realtime capability as a lifetime of data is collected. Further, we expect that data to come from human demonstration, which we do not assume to be perfect.

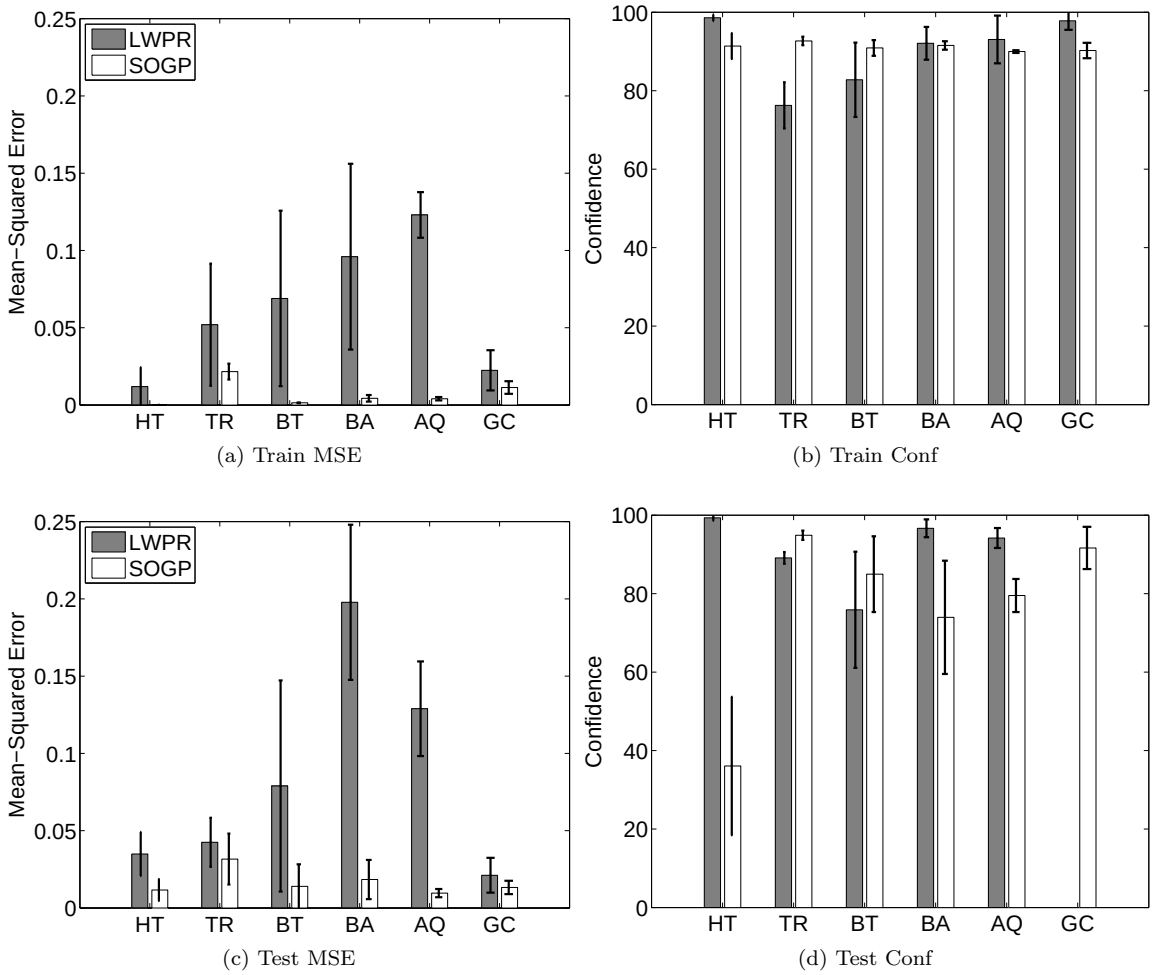


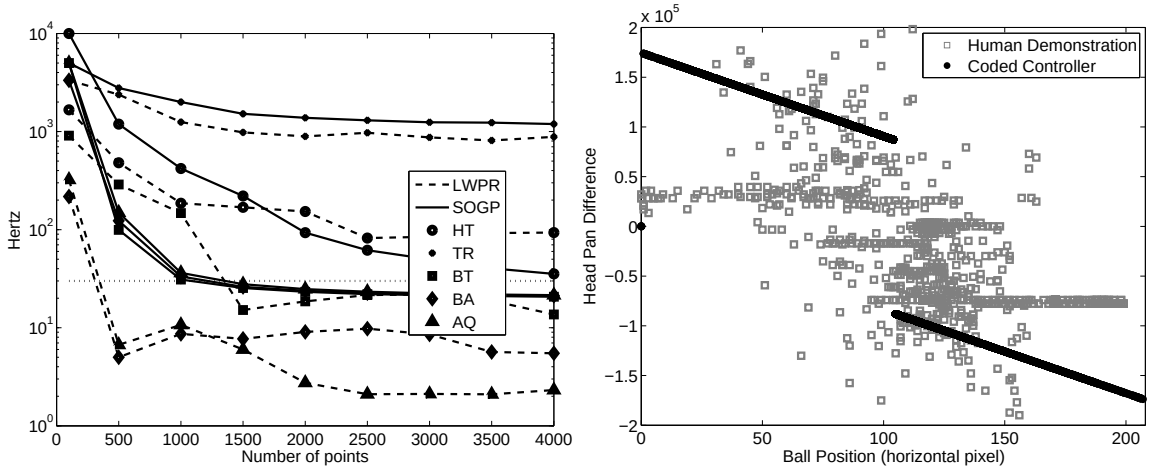
Figure 5.8: MSE and confidence of each algorithm on the training data sets and on leave-one-out testing. All results are averaged over 5 folds and 1 standard deviation error bars are shown.

Speed

To test the speed of the algorithms, we collect a further 5000 datapoints, roughly 2.5 minutes, from the HCCs for several tasks. Training both LWPR and SOGP learners, we record the speed of incorporating each point in hertz, shown with respect to data size in Figure 5.9a.

Both algorithms display asymptotic behavior, although the limit appears to be task-dependent. We posit that this relates to the difficulty of representing the control policy, and that once the control policy is well represented, further data is incorporated with little additional effort. For instance, the TR task, as a pre-determined trajectory of motor poses, is learnt quickly. The BA task, in contrast, requires coordinating locomotion with observed ball position and may require more representational power on behalf of the algorithms.

We also note that the asymptote for SOGP on the more difficult tasks appears fixed, which is



(a) Speed as a function of dataset size

(b) Human Data vs HCC data

Figure 5.9: a) The speed of the algorithms as a function of the amount of data trained on. Dotted line represents 30Hz, or realtime performance. b) Humans are much noisier demonstrators than hand-written code. We seek to, and successfully, learn from this noisy data.

related to the capacity of the algorithm. We have deliberately chosen $\beta = 300$ to keep this limit near 30Hz, the speed of our system. In contrast, the task dependence of LWPR's limit provides no such guarantee, although careful selection of parameters may keep the algorithm realtime. These reasons are some of the ones why we have chosen to base ROGER on SOGP experts. Particularly, as the final speed of inference is independent of data set size (through the hard capacity limit), SOGP will not slow down beyond this limit, even as a lifetime of data is collected.

Noise

To illustrate the noisy demonstrations that arise from human teleoperation, we plot the control signals for the BT task from both the HCC and human teleoperator in Figure 5.9b. The HCC provides very clean data, excellent for learning. Of course, to challenge the learning algorithms, we could add additional noise to this signal.

In contrast, the human demonstrator's data exhibits nonstationary noise. Not only is the noise input dependent, but certain values are preferred over others. It is unclear then what noise model should be used if we were to attempt to simulate it in our HCCs. Additionally, some of this noise may be due to our interface, which may make it easier for the user to express certain values over others. Still, we argue that no matter what the interface, noise such as this will exist. The fact that we *can* learn from this data tells us that our algorithms are up to the challenge. However, for ease of experimentation and development, we often prefer to use HGCs.

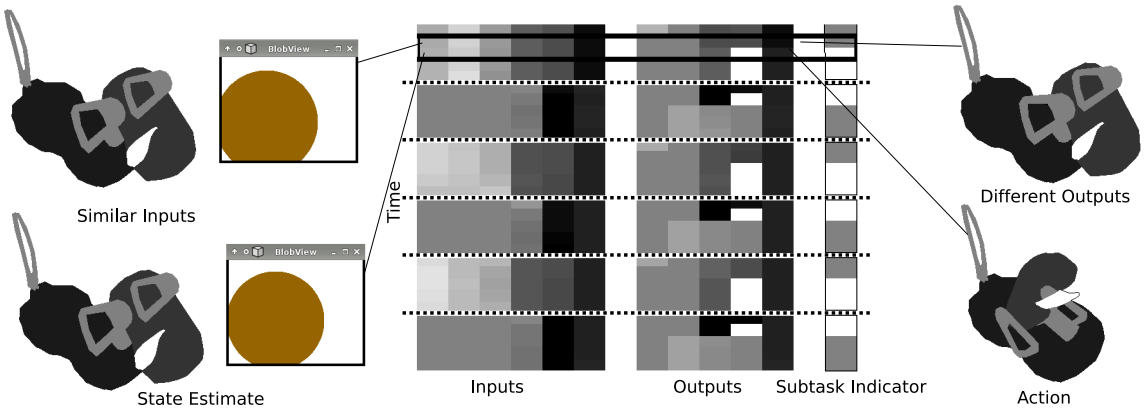


Figure 5.10: Data from the seek/trap transitions of the AQ task. Raw inputs and outputs, with extraneous dimensions removed, are shown in the middle, as is the true subtask indicator (light is seek, dark is trap). Six subtask transitions are shown. One datapoint on either side of the first transition is highlighted and shown graphically. Inputs, or state estimate (head pose and perceived ball location) are on the left, and outputs, or actions (commanded head pose [walk velocities are zero]), on the right. For similar perceptions on either side of the transition, different actuations are demonstrated, leading to perceptual aliasing.

5.2.4 Subtask Switching

The ball-acquire (AQ) task is the first subportion of the goals-scoring behavior that is also an FSM, as indicated in Figure 5.4. AQ can be seen as a composition of the BA and TR tasks, where the robot first locates and approaches the ball and once the ball is in the right location, it performs the trap maneuver. Attempts to learn this task with SOGP and LWPR failed. However, as the subtasks themselves were learnt in the previous section, this failure indicates that it is the transition between them that is causing problems.

We therefore focus on data from the transition between seeking and trapping in the AQ policy, to highlight the features that make it difficult to learn with a unimap regressor. Figure 5.10 (center) shows raw data from around this transition, with extraneous dimensions such as non-ball color blobs and tail position removed. We highlight one transition and examine more closely the data on either side, when the controller has switched from performing the seeking subtask to the trapping subtask. On the left we show the perception inputs, with the head and ball positions. Both states have very similar inputs, and on the right we show the corresponding outputs. When seeking, the controller keeps the head down, and the walk parameters are zero. When trapping, the walk parameters are still zero, but instead the head is raised (to initiate the trap).

When performing unimap regression, there is an implicit assumption that data that is similar in input space is similar in output space as well. In both LWPR and SOGP, this is formalized by the squared exponential, or radial basis function:

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2d}\sum_{i=1}^{d}\frac{||\mathbf{x}_i - \mathbf{x}'_i||^2}{\sigma_i^2}\right) \tag{5.1}$$

which computes the similarity between two datapoints. We see that for the two points on either side of the transition, the RBF measure is 0.9978. LWPR and SOGP both then assume that the outputs would be similarly similar, which is not the case. Instead, the outputs have a measure of 0.1295 between them.

If, however, the hidden state were observed and included in the perception of the system, these two inputs would be separated in perception space. Their similarity would become 0.4866, and as the inputs are less similar, their associated outputs can be differentiated more, and unimap learning may be able to learn the appropriate mapping.

Explicit State

We extended the perception and actuation of the platform to include this explicit state, as discussed in Section 3.5.2. Specifically, we add one more discrete dimension to both that indicates which of the subtasks is being performed. Note, that now the learning system has to learn not only the perception to actuation mapping, but also the state evolution. In terms of learning a finite state machine, we are essentially taking as given the subtasks, and learning their individual policies and the transitions between them.

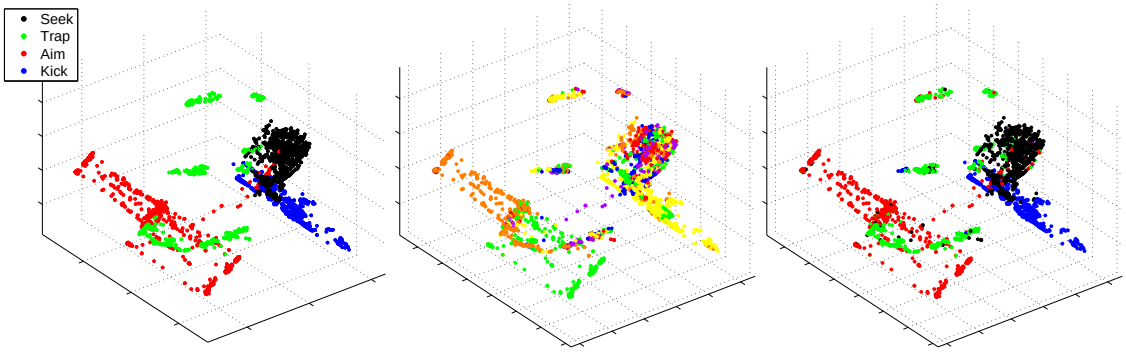
We call the AQ task with extended perception and actuation AQ+, and successfully train a learned controller using a HGC and SOGP. Because of the need for an explicit state variable, direct human teleoperation is untenable. We can conceive of an interface where the user explicitly indicates which of a number of subtasks they are currently performing, and that information is used in learning. However, requiring such information would necessitate that the user analyze the desired task, which may be beyond their ability.

5.2.5 Conclusion

Unimap regression is unsuitable for policies with perceptual aliasing that arises from the switching between multiply applicable subtasks. Instead of correctly performing one of the appropriate actions, controls from all subtasks are merged, resulting in inappropriate behavior. Explicitly using internal state may enable such policies to be learnt, by turning the underlying multimap into a unimap, but requires that the user provide such state. In its absence, multimap regression is needed to determine the number of possible subtasks and their individual policies.

5.3 Multimap Learning

We use ROGER to perform multimap regression on data from the multimap goal scorer's hand coded controller. While collecting data from demonstrations, we also log the true subtask indicator used by the demonstrator to make decisions. Anticipating a bias towards subtasks that have more data, we use only 1000 points of each subtask. Using the ground truth subtask indicator we color the input data (projected into 3D using PCA for visualization) in Figure 5.11a.



(a) Human (b) ROGER (c) Grouped

Figure 5.11: Data from the multimap goal scoring controller projected into 3D with PCA. Color indicates the subtask or expert to which each point is assigned. a) The ground truth coloring uses the known subtask indicator. b) ROGER automatically determines a number of experts (85), and assigns data to each. c) Grouping the experts according to subtask shows that the discovered experts accord with the ground truth.

Using ROGER ($\alpha = 0.5, \kappa = 0.1, \Lambda = 1, \beta = 300, P = 10, w = 0.1, n = 0.1$) we are able to automatically determine a number of experts, the assignment of data to experts, and their individual policies. The resulting assignments are shown, projected into the same PCA space, in 5.11b.

5.3.1 Analysis

ROGER discovers 85 experts in our data, which is more than the 4 that were codded in the HCC. However, this difference is not an indication that ROGER has failed. Rather, the discovered experts may just represent an alternate partitioning of the overall task into subtasks than that which we used when coding.

To examine the data assignments, we plot them per expert in as a stacked histogram in Figure 5.12 along with a zoomed in view. On the horizontal axis is the expert number, and the vertical axis is the number of datapoints assigned to each expert. The data is colored to indicate which of the ground truth subtasks generated it.

We first note that almost all of the data from the kick subtask is assigned to one datapoint, number 43, corresponding to our a priori segmentation of the task. The data associated with this subtask is thus sufficiently similar to itself (in terms of inputs and outputs), and different from the data from other tasks that it is isolated and assigned to an expert. However, not all data from this subtask is in this expert, and this expert contains data from other subtasks as well. The data assigned to other experts can be thought of as outliers from this subtask, data which is sufficiently different that the gating GMM and output SOGP do not well describe it. Likewise, data from other subtasks that are included in this expert can be thought of as overlap data, from the regions where two (or more) subtasks are equally applicable.

Concerning the aiming subtask, the data is assigned predominantly to two experts, numbers 37

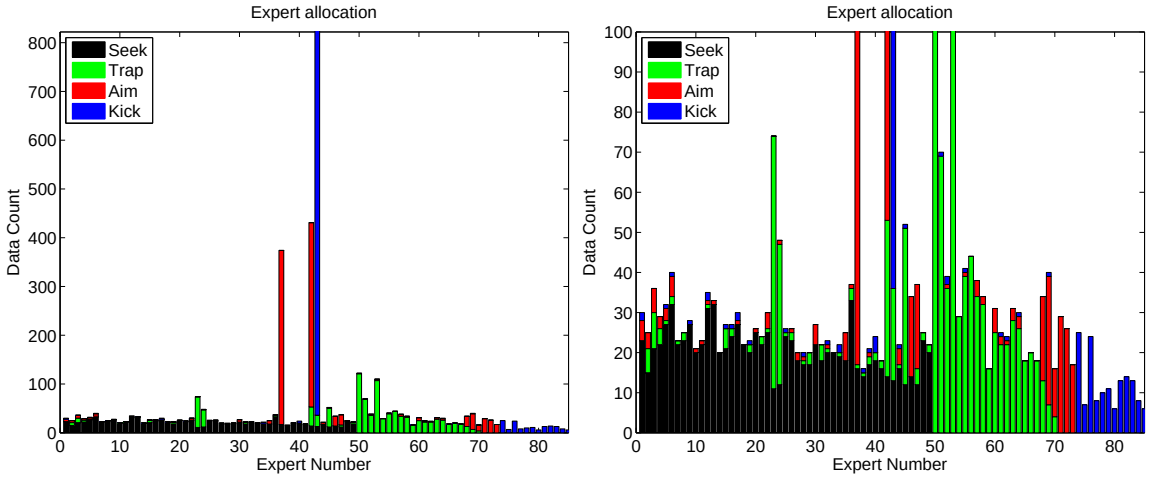


Figure 5.12: The ROGER discovered experts and the data assigned to them, colored by subtask. The right image is a zoomed in view of the left.

and 42. Examining the plot in Figure 5.11b we see that one expert, colored orange, corresponds to one side of the data, and the other expert (green) to the other side. These sides correspond physically to the situations when the robot has the ball and must turn left to the goal, and the one where it must turn right. Upon consideration, this is a reasonable distinction to make. While the hand-coded control policy treats these two cases as the same, and labels them as part of the same subtask, it could also have contained different subtasks for each. ROGER is thus able to discover distinctions not originally thought of by the demonstrator. This property is key, as it will allow ROGER to determine subtasks even when the demonstrator is unaware of them.

Regarding the other subtasks (seek and trap) we note that data for these subtasks are not distributed randomly over experts, but rather split over a series of them. That is, all of the experts contain data from predominantly one subtask, rather than a blend. Exceptions are rare, and likely correspond to regions where the subtasks themselves are very similar. Extrapolating from our previous point, we posit that ROGER is finding distinctions in the data that are not as obvious to us as the left/right split of the aiming task. Thus, a problem here may be that ROGER is finding too many distinctions, rather than not learning to distinguish. Future work, discussed in Chapter 6, considers methods to reduce the number of experts discovered, while improving task performance.

5.3.2 Evaluation

Given the disparity in the number of experts discovered by ROGER and the number sufficient for the task as designed, we investigate the resulting partitioning more deeply. To do so we consider both the quality of the partitioning under the model, and the utility of the learned experts for performing the task.

Joint Probability

In terms of the model, we can calculate the joint probability of the data $\{\mathbf{X}, \mathbf{Y}, \mathbf{z}\}$ (inputs, outputs, and latent subtask indicators), under our parameters Ω. Doing so, we compute the negative log joint probability of the data and the ROGER-discovered latent assignments to be 81359. Using the hand-coded controller's subtask assignments, we find that the NLP of the joint is 81983. The ROGER-discovered assignments have a lower NLP, meaning that these assignments are *more* likely than the ground truth, given our model and the parameters.

We note that our model makes many simplifications and assumptions that are not true in reality, which may combine to make the discovered assignments more likely than the ground truth. For instance, we use a spherical prior in the input model, and equal kernel widths in all dimensions, thus ignoring the differences between them. In addition, we assume that the GP parameters are fixed across all experts. In doing so, we may need multiple experts to fit a region of data, where one expert, with different parameters, may suffice.

Task Performance

With regards to task performance, we can use learned subtask policies to perform the complete multimap goal scoring policy. However, in order to evaluate the learned subtask experts as part of a whole policy, a significant amount of hacking must be performed. Because we do not learn the transitions between the experts, we must provide a hand-coded transitioning system. Evaluation of the total policy then becomes, in part, an evaluation of the transitioning. To control for this confusion, we can use the same transitioning system that was utilized in the HCC. Both policies should then transition the same, leaving us with an evaluation of the underlying subtasks instead. However, as the learned experts do not match up one-to-one with the coded subtasks, we must instead provide a mapping of some sort.

We therefore created meta-experts corresponding to the subtasks used in the HCC's transitioner by combining multiple discovered experts. To do so, we assigned each expert, $i \in [1, 85]$, to one of the transitioner's subtasks, $j \in [1, 4]$, depending on the ground-truth source of the data in each expert. That is, if the plurality of expert i's data comes from the execution of subtask j, expert i is assigned to subtask j. All data associated with the experts assigned to a particular subtask were combined and used to train the meta-expert for that subtask. The assignment of data to meta-experts is shown in Figure 5.11c, where visual inspection indicates that they accord well with the HCC's subtask assignments, although with some misclassifications.

This approach is just one method for making the discovered experts work with the coded transitioner, and it undoubtedly introduces errors. Experiments with the resulting controller are thus not true evaluations of the underlying learned experts, although they will be lower bounds. Alternate approaches to expert selection are also possible, and may give different results. For example, instead of training meta-experts, we could select an expert from within the group stochastically, based on the input model likelihood.

We tested the resulting ROGER-learned controller on the same 13 locations as before, and

achieved a 31% success rate. While much less than the efficacy of the HCC, this result still represents a *significant* improvement over learning with SOGP, which succeeded in 0% of trials. A better technique for selecting experts may improve these results, without any modifications to ROGER.

Analysis of Errors

With 69% of goals missed, there is much room for improvement in the learned goal scorer. Some of these errors (at least 30%) are due to a poor demonstrator, who would miss the same shots. Additionally, we have errors due to improper switching and poor learning.

Improper switching errors are the most common, and reflects on our "hackish" meta-expert and transitioning methodology. An example of this error occurs when the robot approaches the ball, but does not transition to the trap when the ball is in the correct location. Or, once executing the trap, the success of the trap is not accurately detected, and the robot transitions incorrectly. These particular errors are due to our transitioner, which assumes specific values for the inputs and outputs at transition points, which may not result during autonomous execution. To address this issue, we could rewrite the transitioner, or learn the appropriate transition matrix from the data, so that we can use the learned experts directly.

Improper subtask learning can be further attributed to two sources. The first is in the learning of each expert, where SOGP can fail to generalize properly, due perhaps to lack of data. Additionally, mis-allocation of a point to an expert by ROGER can result in cross-contamination of the experts. That is, one expert can contain data from multiple subtasks, and the resulting policy for that expert will perform incorrect averages, as when we used SOGP to learn the multimap goal scoring policy as a whole. Adaptations to ROGER, such as changing the shape of the input space gates, and improved data collection, perhaps from full tutelage, may reduce these errors.

5.3.3 Conclusion

As the multimap control policy outperforms the unimap version, we would like to be able to learn such policies from demonstration. Using standard unimap regression, this is not at all possible. With ROGER, we can automatically determine a number of subtasks and assign data to them, such that the individual learned subtasks policies, with proper switching, can be used to perform the overall task. Improvements to ROGER as well as utilizing the tutelage framework more thoroughly may improve the learned behavior to the level of the original demonstrator.

5.4 Goalie

To round out our learned swarm-team inspired robot soccer team, we learn the goalie behavior shown in Figure 5.2c. The robot starts centered in the goal, and initially sweeps its head to locate the ball. Once the ball has been found, the robot locks on to it and rotates in the goal towards it. If the ball approaches too closely (as defined by a threshold on the blob's size), the robot executes the block behavior. Additionally, if the ball is very close, the goalie attempts to kick it away.

From the 13 field locations in Figure 5.3, we collect 5 shots on goal from each, in random order. The balls were "kicked" by a human[2], from each location towards the center of the goal, attempting to maintain constant velocity over all trials. The resulting efficacy of the HCC was 69%, meaning that it blocked 69% of the shots.

The total collected data has $\sim$ 22,000 datapoints, representing $\sim$12 minutes of training. Training on all of the data, as we did with the UGS policy, would also take 12 minutes. Rather than doing so, we generate a new dataset, interactively using the tutelage framework. From only 4000 datapoints, or only 2 minutes of interaction, we trained a learned goalie that qualitatively performed as well as the HCC. Testing the learned policy from the same 13 locations, we evaluate this learned policy quantitatively, and achieve 62% efficacy, on par with the HCC demonstrator. While these results are not conclusive, they indicate that the the tutelage approach to robot learning may result in policies that perform as well as those learned from batch collection, but requiring smaller datasets and therefore less time.

5.4.1 Analysis of errors

There are two sources of remaining errors in our learned goalie. First, there are the errors that the demonstrator makes, where we cannot expect the learner to be better. Examples include situations where the robot is incorrectly positioned with respect to the shot, and while the robot executes the block motion, the arms are not able to block the ball completely. Likewise, the HCC rarely successfully clears the ball, as after the block it tends to roll away.

Further issues relate to the learning itself. A major source of poor behavior is in the location phase, where the robot is supposed to sweep its head side to side to locate the ball. Occasionally, the learned controller stops mid-sweep, and never locates the ball, leading to a goal being scored. Additional training data may alleviate this problem, but we also believe it may be an issue of multimap demonstration. That is, for similar perceptions (no ball visible), the demonstrator sometimes sweeps left, and sometimes right. In the learner, these two options are averaged so that the robot's head stays still.

5.4.2 Shoot out

To approximate robot soccer, we run a 1-on-1 game[3], where our learned multimap goal scorer faces off against the learned goalie. Similar to a penalty shootout, we start the fielder and goalie in their locations, and reset them after each goal attempt. Extrapolating from the results in our individual trials, we can predict the results of this experiment in two different fashions. First, based on the percentages of successful shots on goal and saves, we would expect $\sim$ 30% of the shots to go in if there were no goalie present, and $\sim$ 60% of those to be blocked, for a total of 4 shots on goal, 2 of which are blocked, and 2 of which score. Understanding that the success or failure of both goal

[2]Jesse Butterfield, member of the 2007 champion Robocup AIBO team.

[3]Technical limitations, mainly wireless bandwith issues, prevent us from running more robots.

Players	1	2	3	4	5	6	7	8	9	10	11	12	13
ROGER	×	✓	×	×	✓	×	✓	×	×	×	×	✓	×
SOGP	✓	✓	✓	✓	✓	✓	×	✓	×	×	×	×	✓
1on1	NS	Block	NS	NS	Block	NS	Score	NS	NS	NS	NS	Score	NS

Table 5.3: Location-based predictions for our one-on-one shootout, using the learned ROGER meta-experts as the scorer, and the learned SOGP system as the goalie. Based on the experiments with the individual controllers, results simplified and reproduced above, we predict the outcomes in the last line, where NS means "No Shot." During actual experimentation, no shots on goal were taken, for reasons discussed in the text.

scoring and blocking may be location dependent, we could also predict which locations will result in scores or saves by comparing the behaviors of the two controllers in isolation, as in Table 5.3. In this case, the two techniques agree that 4 shots on goal will be taken, with our location-based analysis further specifying that the two from locations 7 and 12 will score, while those from locations 2 and 5 will be blocked by the goalie.

When we ran the two learned controllers simultaneously, our observed results fell short of these predictions, as no shots on goal were taken. Therefore the efficacy of the goalie cannot be examined, and we instead focus on the scorer. In addition to the sources of error discussed in Section 5.3.2, we consider two additional ones that arose during the shootout, one technical, the other theoretical.

On the technical side, our experimental setup has both robots transmitting segmented images back to a desktop computer for processing 30 times a second. With more robots, bandwith becomes an issue, and for more than two robots, the resulting increase in dropped packets renders them unusable. With even only two robots, the increase in lag is noticeable, and could cause the learned policy to incorrectly respond to perceptions that the robot is no longer having. This lag underscores the need for realtime, synchronous inference and prediction.

Theoretically, as the policies were trained in the absence of the opposing robot, it may be that the very presence of the robot has changed the perceptual space enough that the learned autonomy cannot generalize appropriately. While we have not calibrated the robots to perceive each other, their glossy finish reflects a wide variety of colors, and may be causing "ghosts" to appear in the other's visual field. Additionally, for the scorer, the goalie's presence alters the view of the goal, by blocking portions of it, casting shadows upon it, or even physically moving it. These changes can make the goal appear offset from where it truly is, smaller and further away, or altogether absent.

With no goals being scored, we present a more qualitative analysis of the 1-on-1 experiment. A sequence of stills from one of the attempts, illustrating both correct and incorrect behavior, is shown in Figure 5.13. We see that the both controllers started off behaving appropriately, with the scorer navigating in a ball-directed fashion and the goalie orienting itself for a block. However, as errors accumulate and the scorer repeatedly fails to transition appropriately, both robots lose track of the ball, and the trial ends with the ball very nearly in the goal, but not quite.

(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 5.13: Video sequence from a learned one-on-one attempt. Initially (a-b), the goalie has locked onto the ball as the scorer approaches. In (c-d), the goalie incorrectly blocks (perhaps due to the shadow cast on the ball by the scorer), and the scorer's coded transition fails to trigger the trap. After blocking, the goalie looses track of the ball, turning into the goal (e-g). Meanwhile, the scorer re-acquires the ball, and, transitioning correctly, moves it towards the goal. The scorer fails to kick in (f-g), perhaps due to inadequate demonstration of what to do when the goalie is present. The trial ends with both controllers unable to locate and react to the ball (h).

5.4.3 Conclusion

With the addition of the goalie behavior, we have learned, from demonstration, the component controllers for a swarm-style robot soccer team. The various tasks, and the learning algorithms used to learn them, were varied independently, leveraging the abstract formulation of the Dogged Learning architecture. Furthermore, our experiments encompass multiple sources of data, humans and HCCs, and methods of gathering that data, batch and incremental.

For unimap learning, the transitioned control policies were on par with their demonstrated versions. For the multimap policy, our learned controller lags behind, but still represents an improvement over using standard unimap regression, which cannot learn the policy at all. In both cases, the demonstrated policies themselves were not the best, and improving the demonstration is one approach to improving the learned controllers, which are currently not robust enough to perform in game-like environments. Other approaches to improving learning, focused on reducing or eliminating the limitations of our current techniques, are discussed in the next chapter.

Chapter 6

Discussion and Conclusion

> *What if these theories are really true, and we were magically shrunk and put into someone's brain while he was thinking. We would see all the pumps, pistons, gears and levers working away, and we would be able to describe their workings completely, in mechanical terms, thereby completely describing the though processes of the brain. But that description would nowhere contain any mention of thought. It would contain nothing but descriptions of pumps, pistons, levers!*
>
> Gottfried Wilhelm Leibniz

This dissertation has focused on the issue of Human-Robot Policy Transfer (HRPT), or how users instantiate control policies that are latent in their minds onto robots. Specifically, we have considered learning as an approach that enables implementation of desired autonomous robot controllers without explicitly coding or otherwise procedurally analyzing the task. By using Learning from Demonstration (LfD), only task-appropriate perception-actuation pairs need to be provided, no other task-specific information is needed. However, for some tasks, data of this form may have multiple actuations associated with one perception, giving rise to *perceptual aliasing*, where the robot's inputs do not contain enough information to determine the correct output. Such situations can occur in Finite-State Machine (FSM) based robot controllers, where multiple machine states (subtasks) may be used for the same perception at different times. FSM controllers with perceptual aliasing cannot be learned using standard Direct Policy Approximation (DPA) techniques (regression) as they violate the "vertical line test." Instead, state of the art approaches require additional information from the user related to the decomposition of the overall task into subtasks, namely either the subtasks themselves or indications of the transitions between them.

We have performed RLfD using our Dogged Learning (DL) architecture, described in Chapter 3. Designed to be abstract, it specifies only the data flow between the environment, platform, autonomous decision making, and demonstrator, and is thus usable with multiple platforms, learning algorithms, and demonstration interfaces. Further, DL provides for interactive training, or robot tutelage, by using Mixed-Initiative Control (MIC), where a user provides data demonstrating portions of the task when they observe incorrect behavior, or when requested by the learning system.

To address perceptual aliasing arising from subtask switching in FSM controllers, we developed Realtime Overlapping Gaussian Expert Regression (ROGER), described in Chapter 4. Treating the problem as one of multimap regression, ROGER estimates a sufficient number of subtasks for the controller by tracking the distribution over all possibilities, up to an infinite number of them. Individual subtask policies are learned using nonparametric regression, as the form of the mapping from perception to actuation is not known in advance. Inference is performed in an incremental, sparse fashion, making ROGER the first multimap regression algorithm appropriate for tutelage.

Using these contributions, we have learned robot control policies representative of a beginner swarm-style AIBO robocup team. This team consists of two separate behaviors, a goalie and fielder, which were trained only by changing the demonstrator, no other aspect of the learning system was modified between tasks. With DL, we compared different regression learning algorithms on these and other tasks, again by only changing the appropriate aspect of the system. For tasks without perceptual aliasing, standard regression techniques are sufficient for learning autonomous controllers that are on par with the demonstrator. However, for behaviors with multiple, overlapping subtasks, such as a more robust goal scorer, multimap regression is required.

This chapter will discuss our two main contributions and the work as a whole. Highlighting the strengths of the approaches, we indicate further hypothesis to which they can be applied. We also mention some limitations of the techniques, and our implementations thereof, and point out areas of needed future development. This chapter, and the dissertation, concludes with a brief summary and vision of the future.

6.1 Dogged Learning

Dogged Learning is our approach to mixed-initiative, interactive robot tutelage of unknown tasks via human teleoperative demonstration. We take as given a robot platform with known sensor and effectors, embodied in a fixed environment, as well as perceptual and actuational processes that transform the raw sensor and effector data into the perception and action space of the control policy. Decision making operates in these spaces, yielding a mapping from perception to actuation that originates from the demonstrator, but is eventually approximated by the learner. We enable mixed initiative control by using the idea of confidence-based arbitration, where the controller (learner or demonstrator) that is more confident has control of the platform.

In considering DL, we distinguish between the abstract architecture and our particular implementation. In writing our system we have made several simplifications and design decisions that ease the running of our experiments, but may cause the current code to fall short of the full potential of the architecture. We are continuing to develop the code, adding more capabilities and removing component-specific functionality as we incorporate more platforms, algorithms, and interfaces.

There are also simplifications that while initially made to ease implementation, may be more difficult to remove. As an example, we currently only consider perception and actuation spaces of fixed dimensionality, with known scaling coefficients. These assumptions enable us to use many

off-the-shelf machine learning algorithms, without having to tune their parameters for different tasks or platforms. However, as discussed in Section 3.1, other platforms may have perception-actuation spaces of variable dimensionality, or unknown scale. While the DL architecture's specification allows for those platforms to be used, the current implementation is not easily adapted to them.

We are also developing the system to improve applicability and adoption outside of our own research lab. Our current implementation relies on libraries for all possible components being known at compile time. This fact means that all researchers using DL must have the libraries for all learning algorithms, demonstration interfaces and platforms that exist, even if they do not use them. We are working on a more run-time friendly system, where only the components needed for a particular experiment need to be implemented and present. Such modularity may also allow us to distribute the activity of the DL system, so that demonstrators, learners, and platforms need not be co-located.

6.1.1 Strengths

Considering DL as an architecture, its main strengths are its abstract definition of the data flow, and its ability to support interactive learning. Leveraging its abstractness, we directly compared different learning algorithms and human and hand-coded controllers for use in the robot tutelage paradigm in Sections 5.2.2 and 5.2.3. Using the interactive nature of the tutelage paradigm, we were able to quickly home in on the aspects of the multimap goal-scoring task that made learning impossible with standard regression in Section 5.2.4.

More broadly, we see DL as a general LfD framework, that could be applied in various manners. In addition to running on more robots and tasks with different interfaces, it can also be applied to non-robotic platforms. Specifically, as the architecture only requires that the platform sense and effect the environment, the platform can be purely software instead of physical. Virtual robots in simulation are an obvious possibility, but other, programmatic environments such as text editing or email sorting may also be applicable.

We also consider DL as being suitable for use in Human-Robot Interaction (HRI) research. Through the use of real-time interactive learning, DL can be used to rapidly "prototype" behaviors for use in user studies, and even adapt them as the users are present. By doing so, time and effort that would otherwise be spent programming the behavior or running the robot in a "Wizard-of-Oz" setting can be allocated elsewhere. Further, because DL is designed to enable non-programmers to instantiate control policies, there is the possibility that the users could edit the robot's behavior themselves, providing additional feedback as to how they would prefer the robot behave. The resulting autonomous control policy could also continue to operate after the user leaves.

Future Experiments

This dissertation has focused on using DL to examine different learning algorithms for robot tutelage. Our experiments in learning robot soccer behaviors led us to focus on perceptual aliasing, and dealing with the presence of hidden state and multimap scenarios. However, with the above in mind, we can formulate some other experiments for which DL would be a useful tool.

The first experiment would be a continuation of the work we did with SOGP and LWPR, and examine several other learning algorithms. A large empirical study of the state-of-the-art in regression approaches, a well as the publication of the resulting data sets and algorithm implementations, would enable researchers to better choose the algorithm that fits their needs, and point out areas of regression (such as multimaps) that require more work. Using DL, algorithms can all be applied to exactly the same platform and task, although the interactive training portion would have to be sacrificed if the exact data is reused.

On the interactive side, while we have used tutelage to teach our robots, our desire to do so is based only on personal experience. An evaluation of the approach on a broader set of users is a necessity if the technique is to be shown useful. Hypothesis that can be tested include: Does tutelage result in smaller datasets for a particular level of robot competence? Do users prefer interactive teaching to batch data collection? Does mixed initiative control result in improved performance?

Considering users, we have focused on machine learning, and somewhat neglected the design of our user interfaces. We have been guided in our development by the intuition that more immersive devices (wiimote as opposed to joystick as opposed to keyboard) are better, but again a user study would be able to prove or disprove this hypothesis. As with learning, different interface components can be developed in isolation, and then tested on the same platform.

6.1.2 Limitations

In addition to the limitations of our implementation discussed above, the DL architecture itself is limited in what and how it can learn. In particular, by learning unknown tasks only from user demonstrations of that task, DL can only learn behaviors that the user can themselves perform, and can only learn to perform them as the user does. It is, then, in general limited to being as good at the task as the user is themselves. Slight improvements are possible given that learning algorithms attempt to filter out noise in demonstration, but as we saw in Section 5.2.3, human demonstration may be full of unmodeled noise.

Other RLfD approaches have enabled better-than-demonstrator learning by incorporating task-specific knowledge into the system, such as indications of desired goal states [13, 74]. They then use reinforcement learning, or other reward-based techniques, to optimize the initial policy learnt from demonstration with respect to those goals. More generally, an entire reward function can be provided, and the demonstrated policy used as a seed for standard RL techniques such as policy iteration by gradient ascent. To utilize these approaches in DL, methods for obtaining the necessary information from the user must be developed.

Related to reliance only upon demonstration is the fact that DL only allows for the user to demonstrate appropriate actions for the *current* perception. That is, the action that the user is commanding is assumed to be the appropriate response for the perception that the platform is currently observing. Lag in the system due to network latency, code execution, and human synapse firings is generally not a problem, as the robot's movement through perception space tends to be even slower. However, if the environment is sufficiently dynamic, the human response time may be

too slow to effectively demonstrate the desired task. Further, if the user makes a mistake in the demonstration, they cannot correct it until the robot once again has the necessary perception.

This last issue raises a new one, which is that DL relies solely upon positive demonstration. Like other LfD techniques, it assumes that the user is demonstrating appropriate actions for the task. However, mistakes in demonstration nonwithstanding, it may actually be easier for the user to demonstrate what *not* to do, perhaps because the set of forbidden actions is smaller than those allowed. While reinforcement learning has considered negative rewards, regression approaches have traditionally not.

Future Development

To address these issues, future work in developing DL will focus on incorporating aspects of RL. Specifically, DL needs additional channels of information flow from the demonstrator to the decision making system, other than just an appropriate action and confidence. Modifications to the learning system itself may be required to take advantage of this new information.

As an example, consider adding a reward channel to DL, where during autonomous execution, the user can provide positive or negative reward (perhaps through vocal prosody [73] or a clicker [72]). To fully utilize this channel, the autonomous behavior must contain some aspect of exploration, instead of only exploiting the demonstration. One method, for use in ROGER and SOGP, would be to sample from the output distribution, instead of only returning the mean and using the variance as confidence. This approach would have the added benefit that areas of high confidence, where the policy mapping is known with greater surety, would be subject to less exploration.

Incorporating the reward into the learner itself is nontrivial. When an explored action is positively rewarded, one approach would be to add it to the learner, as if the perception-actuation pair had been generated by the demonstrator themselves. The resulting distribution over actions would then shift to cover both the original demonstration, and the recently explored and positively rewarded one. By weighing the data differently, the learning algorithm can be made to favor demonstration or exploration-generated data. From the data it may even be possible to infer which is more trustworthy [9]. However, if an explored action generates a negative signal, it is unclear how the underlying distribution over actions should change. One option would be to shift the distribution away from the negative point, but keep the modality the same. Another possibility would be to "split" the distribution, introducing a new peak, with a corresponding valley at the negative example.

As an alternative to providing point rewards for individual perceptions or perception-action pairs as they occur, users could provide more overarching feedback. Again, a method for getting this information from the user is needed, but they could, for example, indicate that a particular situation is the goal of the current task. In that case, we would likely need a reinforcement learning technique running in parallel with regression to develop the policy.

Lastly, if the user can indicate future reward by identifying goal states, they could likely also provide feedback on previously performed actions as well, addressing the synchrony limitation. As in [8], a user could be presented with a trace of the robot's path through perception-action space, and

assign rewards or provide demonstrations after the fact. Further, they could also indicate appropriate modifications to the policy (advice) to improve it's efficacy beyond that of the demonstrator.

6.2 Realtime Overlapping Gaussian Expert Regression

ROGER addresses the issue of perceptual aliasing that can arise when attempting to learn FSM-based controllers from demonstration. In particular, as only perception-actuation pairs are observed, the machine state, or subtask, which gave rise to a particular datapoint is unknown. ROGER explicitly attempts to infer these latent subtask indicator variables, modelling each subtask as a expert with a Gaussian area of influence in perception space, and a Gaussian distribution over possible mappings from perception to actuation. Using the Chinese Restaurant Process (CRP), datapoints are sequentially assigned to experts, always considering the possibility that a new expert may need to be created. Thus, in terms of the FSM, ROGER performs both model selection and subtask learning as it determines the appropriate number of experts and their individual mappings.

Like with DL, our implementation of ROGER makes some simplifications to the true model for ease of implementation and experimentation. Mostly, we have altered the parameterization, by assuming that Λ, the covariance of the input space prior, is spherical, and that the kernel width is isotropic. Additionally, we assume that the output dimensions are independent of one another. These approximations relieve us of having to specify many hundreds of additional parameters, but may have had an adverse effect on our results.

Rather than hand setting the full (or reduced) parameter set, we can extend the inference procedure of ROGER to automatically tune them during training. Small gradient ascent steps can be inserted after datapoints are incorporated to adjust the input space hyperparameters. Expectation-maximization approaches for setting Gaussian process parameters already exist and could likewise be included. Using such a technique would also address another simplification that we made, which is to assume that all experts share the same parameters.

However, increasing the amount of computation that needs to be performed at inference will necessarily slow down the process. To maintain realtime, interactive computation, we must streamline and improve our code. Regular optimization techniques can be applied, and we are investigating faster math libraries and methods to reduce overhead and data replication. Further sparsification techniques, such as automatically changing the number of particles or basis vector size on the fly, may also be necessary.

6.2.1 Strengths

ROGER's major strengths stem from the fact that it is a Bayesian, nonparametric approach. By being Bayesian, ROGER leverages uncertainty in the form of distributions to enable it to consider multiple possibilities for the number of subtasks, the exact partitioning of the data, and even the mappings in the experts themselves. Further, additional information over what these values may be can be incorporated in a principled manner, via priors. Through the use of nonparametrics,

ROGER can adapt to data of any form, with up to an infinite number of experts and arbitrarily shaped mappings in each one.

These two aspects of ROGER are somewhat independent, but their use is intertwined. We could have developed a multimap regressor that is Bayesian but not nonparametric, one that modeled uncertainty over partitions with a fixed number of experts, or assumed a known form for the mappings. Alternatively, a nonparametric but not Bayesian approach would use heuristics to determine when a new expert should be generated or the mapping should change form. Using both techniques combines their strengths (uncertainty and adaptability) to make an approach that is more powerful and flexible than either individually.

Specifically, by performing model selection within an infinite (nonparametric) Bayesian model, ROGER has an advantage over techniques that assume known models or determine an appropriate ones in a brute-force, heuristic, or ad-hoc method. Primarily this benefit is because the entire distribution over possibilities (partitions and mappings) is approximated, instead of a single point estimate. It is important to note that the approximation is done with a set of points, so in the worst case ROGER with one particle is the same as a single point "best" estimate, which can get stuck in local optima just like the non-Bayesian approaches. However, with more than one particle, ROGER maintains a weighted set that is optimally chosen to represent the entire distribution.

Future Experiments

As a Bayesian model, ROGER is amenable to multiple inference and prediction algorithms. As shown in Section 4.3.2, our incremental particle filter approach may be more suitable than batch techniques in resource constrained settings, such as robot tutelage. However, as our approach inherently generates a sparse approximation of the entire distribution, it is possible that the determined partitioning is suboptimal. In fact, when resources are available, the batch inference technique may be better. A logical experiment would be to examine if the two techniques can be combined, where slow batch updates are interspersed within the fast incremental inference. By running the batch updates during processor "down time," the resulting system may better approximate the demonstrated policy (in terms of accuracy and time to learning), with minimal impact on the interactive nature of demonstration.

We would also like to apply ROGER to additional tasks, outside of the domain of robot soccer. In this dissertation we have shown that ROGER is suitable for learning a particular FSM-style controller that was previously unlearnable using regression-based LfD. However, the limits of ROGER have not been probed. Other datasets, with more subtasks or closer overlap between them, may prove more difficult to deal with. Automatic parameter tuning, discussed above, may be necessary to learn tasks whose subtasks vary widely in noise, mapping, and input distribution.

Lastly, ROGER may be applicable outside of robot tutelage. Abstractly, we have equated multimap regression with model selection and subtask learning in an FSM, and thus could apply ROGER to any FSM learning problem, provided that the transitions can be learned separately. Beyond FSMs, multimaps may arise in other situations, when provided features are insufficient to determine

appropriate outputs. By fostering relationships within and without the greater machine learning community, we may find other datasets and problems to which ROGER can be applied.

6.2.2 Limitations

In the context of our work, that of inferring FSM robot controllers, ROGER's greatest limitation is in its lack of using temporal information. While the CRP does generate expert assignments sequentially, and there are thus temporal dependencies between assignments, the overall partitioning is independent of the order of the datapoints. Note, that the numbers assigned to experts are inconsequential, used only for bookkeeping, and can be permuted to no ill effect. Truly leveraging the information present in the sequence of datapoints would correspond to estimating the transition matrix between the discovered subtasks.

A further limitation of our model, although not as serious of one, is that we assume that inputs for a particular expert are distributed Gaussianly in perception space. For certain platforms, features, and tasks, this assumption may not be true, and an alternate model may be more appropriate. Continuing with our nonparametric approach, a generalized kernel density estimator may enable ROGER to adapt to arbitrarily shaped input distributions. Currently, however, our infinite mixture model approach allows non-Gaussian distributions to be modeled with multiple experts.

Another aspect of some concern is the current noise model in ROGER. The Gaussian Process experts of which it is composed each assume that observations are corrupted by stationary Gaussian noise. Noise, particularly from human demonstration, may exhibit nither of these properties, taking a different form and also being dependent on the particular location of the data, as seen in Figure 5.9b. Nonstationary Gaussian process models [103] may be "swapped" in for our current output model to address this issue, leveraging the modularity of our system.

Future Development

Our current work on ROGER focuses on incorporating temporality into the model. First attempts used an ad-hoc post-processing approach that calculated the transition probabilities between experts in the demonstration data, and then used those probabilities during prediction. This technique did not lead to stable transitioning or acceptable task performance. Currently, we are investigating approaches that modify the inference algorithm itself, to infer both the expert assignments and transition matrix simultaneously. One possible adaptation would be to change the distribution over partitions, as in

$$P(z_i = k|\mathbf{z}_{-i};\alpha) = \left\{ \begin{array}{ll} \frac{\beta}{N+\alpha-1} & k = z_{i-1} \\ \frac{m_k-\beta}{N+\alpha-1}, & k \leq K, k \neq z_{i-1} \\ \frac{\alpha}{N+\alpha-1}, & k = K^+ \end{array} \right\} \tag{6.1}$$

By making it more likely that a point is assigned to the same expert as the datapoint immediately preceding it, we can encourage the formation of chains of datapoints that represent temporally-extended subtask execution. The trace of subtask activity over time would then accord with our

intuition that in FSM controllers, rapid subtask oscillation does not occur. However, modifying only the CRP to effect this change may not be enough, as we would also need to take into account the datapoint's relative locations in perception space.

By performing transition and partition estimation at the same time, we hope to also address oversegmentation, where ROGER subdivides subtasks into many different experts. Oversegmentation is not an issue in and of itself, as so long as the experts are transitioned between correctly it does not matter how many there are. However, oversegmentation may be indicative of overfitting, where the model so tightly adjusts to the training data that it fails to generalize properly to new situations. Reducing oscillation between experts may also reduce their total number

Changing the input model may also reduce oversegmentation, although its effects on overfitting are less obvious. As mentioned in Section 4.4, using a non-Gaussian input model may enable one expert to fit data that would require multiple, smaller Gaussian experts. By having all of the data in one, possibly non-convex, expert, the output mapping may better generalize to new situations covered by the non-Gaussian distribution. However, nonparametric density estimation may result in an input gate that overfits the data in another sense, in that the borders of the distribution can be arbitrarily close to all of the data, preventing any generalization at all.

6.3 Summary

In this dissertation, we have used DL and ROGER to investigate the applicability of direct policy approximation techniques to learning autonomous robot control policies from interactive demonstration. From our experiments with SOGP and LWPR on a set of Robocup-inspired soccer-related tasks, including a complete swarm team, we determine that standard regression approaches are suitable for learning unimap controllers directly from perception-actuation data. The results are summarized in Figure 6.1, showing that for these unimap tasks, the learned behaviors are on par with those of the demonstrator.

However, for data from an FSM controller, which consists of multiple subtasks that are mutually applicable in certain perceptual states, standard regression is not suitable, and the learned policy that results from using them fails to perform the task at all. Our analysis indicates that the overlap between subtasks in perception space leads to perceptual aliasing, where one perception has associated with it multiple, conflicting actions in the demonstration data. Standard regression, assuming unimodally distributed outputs for a given input, incorrectly averages these possibilities.

Applying multimap regression, we learned a set of subtask controllers that when executed appropriately, reperform the demonstrated task (multimap fieldsbot), albeit not at the same level of the demonstrator. Key to the approach we used is that both the number of subtasks and their policies were automatically determined, without additional information from the user, such as when transitions occurred. The discovered subtasks may or may not match how a human would segment the task, but are sufficient to perform the task.

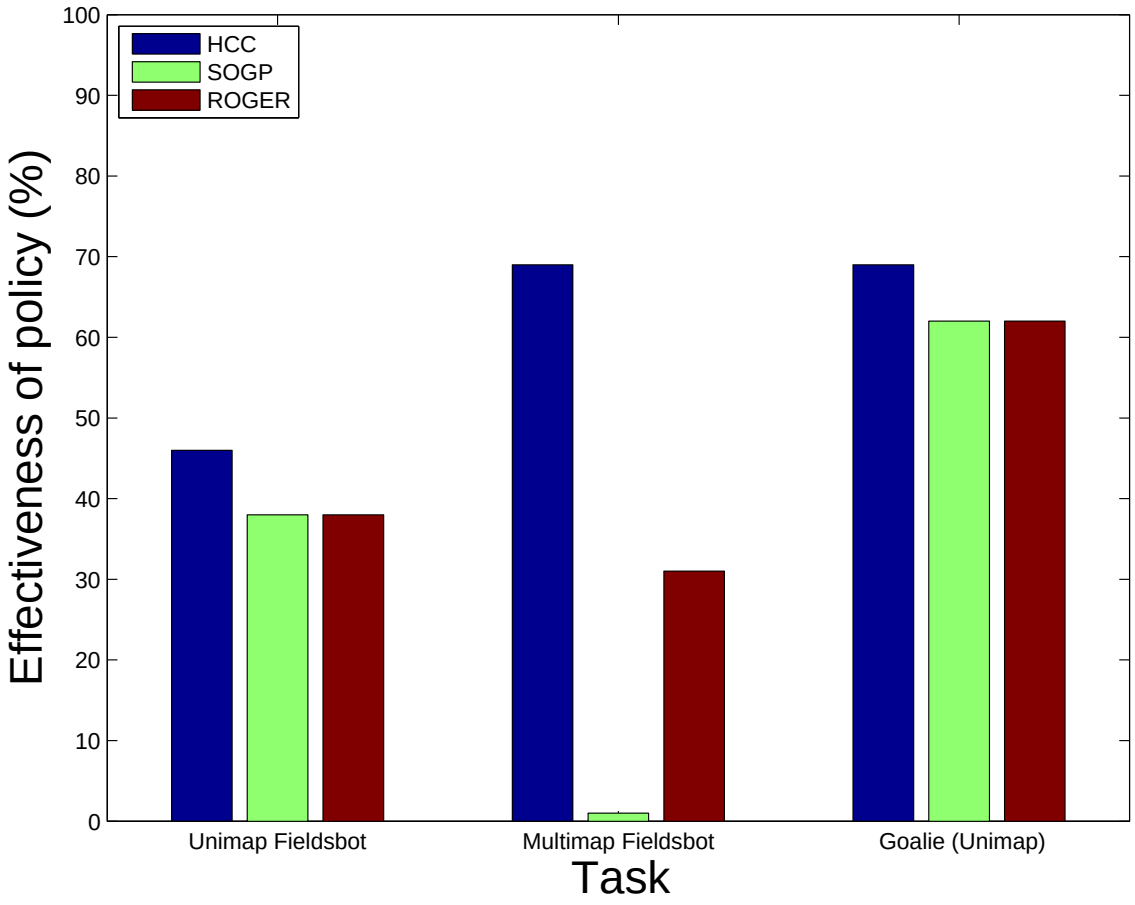


Figure 6.1: A summary of learning the swarm tasks. For the unimap tasks, unimap regression (SOGP) learns a controller that is on par with the demonstrator, in terms of task effectiveness. However, for the improved multimap scorer, SOGP fails to learn at all. Using multimap regression (ROGER) to discover subtasks, we can learn an improved controller.

6.3.1 Strengths

The above result illustrates the primary strength of the work presented in this dissertation, that we learn FSM robot controller subtasks incrementally from unsegmented demonstration data, which to the best of our knowledge has never been done before. Previous approaches to FSM learning have always taken the subtasks as known, either explicitly (by being given the subtask policies themselves), or implicitly (by knowing when transitions occur). ROGER is also the first multimap regression algorithm to be formulated specifically for, and be applicable to, interactive inference, such as that which occurs during robot tutelage.

Despite being developed for robotics, the techniques developed herein are not tied to this domain. Our developed code, which will be made freely available for academic use, is modular and abstract, to enable other researchers to build off of our work. Already, our Dogged Learning architecture has

been applied outside of our lab, and our learning algorithms have been used in alternate settings, such as combinatorial optimization. By providing an abstract interface (Dogged Learning), as well as initial building blocks (virtual robots, ROGER, keyboard/screen interface), we have contributed tools that others may find useful to extend this, and their own, research.

6.3.2 Limitations

While we have shown the applicability of our approaches to learning FSM controllers, our experimentation is not exhaustive. Using mainly one robot, and one domain of tasks, our work has primarily focused on showing that standard regression techniques are *not* suitable for learning FSMs from demonstration, but that multimap techniques *are*. Our investigation of the perceptual aliasing that occurs during subtask switching, and the resulting combination of possible alternatives that occurs in unimap regression, constitutes a proof by analysis that unimap regression is not applicable. Our experiments with learning the multimap goal scorer constitute a proof by example that multimap approaches can be used to infer an appropriate subtask segmentation.

However, our experimentation does not support the conclusion that ROGER is more generally applicable than in this case. We may have "gotten lucky" with our choice of platform, task, parameters, and even data. Recall that we used equal amounts of data from each of the subtasks when training. Applying ROGER more broadly may alleviate this concern, or indicate changes that are necessary to create a more general multimap regressor, or characteristics of data that are necessary for effective learning. Likewise, our DL architecture has only been tested with a handful of platforms, learning algorithms, and demonstrators, and a broader set of experiments would lay to rest concerns about its applicability as well.

6.3.3 Future Work

In addition to the experiments discussed in Sections 6.1 and 6.2, we are interested in using DL and ROGER to learn a wide variety of tasks from many users over long periods of time. Aside from testing the general nature of the techniques, such an experiment would also examine their applicability to lifelong learning, where a robot must learn multiple tasks and perform them over its entire existence. We must then measure time in years, not minutes, and consider datasets that number in the millions or billions of points (10^{10} for one year's worth of data at 30 hertz).

DL and ROGER are well suited for use in lifelong learning. The interactive, mixed-initiative control aspect of DL means that learning effectively never stops. Once the user has trained the robot to perform one task to satisfaction, they simply stop demonstrating, but the DL system itself can be left running. Then, if the user desires the robot to perform a new task, they can seamlessly transition back to demonstration.

Determining the existence of multiple tasks in the robot's lifelong data set can be seen as inferring subtasks in an overarching "life" task. Further, if these life-subtasks are themselves composed of subtasks or skills, there is the potential for skill sharing, where individual skills are used in multiple

higher-level subtasks. Using ROGER in a hierarchical fashion may then be a method for transfer learning [31], where skills learned for one subtask are used or refined in the performance of another.

Our RGame interface, described in Section 3.5.1, is one step towards performing the scalability experiments discussed in this subsection. By enabling users, from wherever they are in the world, to demonstrate tasks to robots that are not colocated, we can greatly increase the size of our user base. From such a diverse set of users, we can gather the needed billions of datapoints in a shorter amount of time than if we generated them ourselves. Further, different users may demonstrate the same task in multiple fashions, providing further variance in the data. Using this data we will be able to better explore what qualities are needed for successful HRPT using current techniques, and where new technique development should focus.

6.4 Conclusion

This dissertation started by examining the concept of a universal robot, one that could adapt itself to users' needs as they changed. We argued for the use of learning for Human-Robot Policy Transfer as a means to allow users without programming or analytical expertise to instantiate desired autonomous control policies. Focusing on interactive demonstration, our experiments with Dogged Learning showed that using state-of-the-art regression techniques for direct policy approximation limits such users to developing controllers that are unimaps in the underlying perception-action space. Learning alternate (multimap) controllers either required additional information derived from procedural analysis, or a modification of the platform in a task-specific manner. With ROGER (Realtime Overlapping Gaussian Expert Regression), we have shown that it is possible to learn subtasks sufficient for recreating multimap policies directly from unsegmented data.

Bibliography

[1] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Neural Information Processing Systems*, pages 1–8, Vancouver, CAN, December 2006.

[2] Pieter Abbeel, Dmitri Dolgov, Andrew Y. Ng, and Sebastian Thrun. Apprenticeship learning for motion planning with application to parking lot navigation. In *International Conference on Intelligent Robots and Systems*, pages 1083–1090, Nice, France, September 2008.

[3] Julie A. Adams, Pramila Rani, and Nilanjan Sarkar. Mixed initiative interaction and robotic systems. Technical Report WS-04-10, Vanderbilt University, 2004.

[4] Michael A.Goodrich, Timothy W. McLain, Jeffrey D. Anderson, Jisang Sun, and Jacob W. Crandall. Managing autonomy in robot teams: observations from four experiments. In *International Conference on Human-Robot Interaction*, pages 25–32, Arlington, VA, March 2007.

[5] Luis von Ahn. Human computation. In *International Conference on Knowledge Capture*, pages 5–6, Whistler, BC, Canada, October 2007.

[6] Luis von Ahn and Laura Dabbish. Designing games with a purpose. *Communications of the ACM*, 51(8):58–67, August 2008.

[7] Brenna Argall, Brett Browning, and Manuela Veloso. Learning by demonstration with critique from a human teacher. In *International Conference on Human-Robot Interaction*, pages 57–64, Arlington, VA, March 2007.

[8] Brenna D. Argall, Brett Browning, and Manuela Veloso. Learning robot motion control with demonstration and advice-operators. In *International Conference on Intelligent Robots and Systems*, pages 399–404, Nice, France, September 2008.

[9] Brenna D. Argall, Brett Browning, and Manuela Veloso. Automatic weight learning for multiple data sources when learning from demonstration. In *International Conference on Robotics and Automation*, pages 226–231, Kobe, Japan, May 2009.

[10] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469 – 483, May 2009.

[11] Ronald Arkin. *Behavior-Based Robotics*. MIT Press, 1998.

[12] Arthur Asuncion and David J. Newman. UCI machine learning repository. http://www.ics.uci.edu/∼mlearn/MLRepository.html, 2007.

[13] Chris Atkeson and Stefan Schaal. Robot learning from demonstration. In *International Conference on Machine Learning*, pages 12–20, Nashville, TN, July 1997.

[14] Paul Bakker and Yasuo Kuniyoshi. Robot see, robot do: An overview of robot imitation. In *AISB Workshop on Learning in Robots and Animals*, pages 3–11, Brighton, U.K., April 1996.

[15] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1-2):41–77, January 2003.

[16] Elizabeth A. Basha, Sai Ravela, and Daniela Rus. Model-based monitoring for early warning flood detection. In *ACM Conference on Embedded Network Sensor Systems*, pages 295–308, Raleigh, NC, November 2008.

[17] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded Up Robust Features. *Computer Vision and Image Understanding*, 110(3):346–359, June 2008.

[18] Matthew J. Beal, Zoubin Ghahramani, and Carl Edward Rasmussen. The infinite hidden Markov model. In *Neural Information Processing Systems*, pages 577–584, Vancouver, CAN, December 2001.

[19] Darrin C. Bentivegna, Christopher G. Atkeson, and Gordon Cheng. Learning tasks from observation and practice. *Robotics and Autonomous Systems*, 47(2-3):163–169, June 2004.

[20] Cindy L. Bethel and Robin R. Murphy. Affective expression in appearance constrained robots. In *International Conference on Human-Robot Interaction*, pages 327–328, Salt Lake City, Utah, USA, March 2006.

[21] Bruce Blumberg, Marc Downie, Yuri Ivanov, Matt Berlin, Michael Patrick Johnson, and Bill Tomlinson. Integrated learning for interactive synthetic characters. In *Conference on Computer Graphics and Interactive Techniques*, pages 417–426, San Antonio, Texas, July 2002.

[22] Edwin V. Bonilla, Kian Ming A. Chai, and Christopher K. I. Williams. Multi-task Gaussian process prediction. In *Neural Information Processing Systems*, pages 153–160, Vancouver, CAN, December 2007.

[23] Gary Rost Bradski and Adrian Kaehler. *Learning OpenCV*. O'Reilly, 2008.

[24] Cynthia Breazeal, Matt Berlin, Andrew G. Brooks, Jesse Gray, and Andrea L. Thomaz. Using perspective taking to learn from ambiguous demonstrations. *Robotics and Autonomous Systems*, 54(5):385–393, May 2006.

[25] Cynthia Breazeal, Andrew Brooks, Jesse Gray, Guy Hoffman, Corey Kidd, Hans Lee, Jeff Lieberman, Andrea Lockerd, and David Chilongo. Tutelage and collaboration for humanoid robots. *International Journal of Humanoid Robotics*, 1(2):315–348, June 2004.

[26] Rodney A. Brooks. Intelligence without reason. In *International Joint Conference on Artificial Intelligence*, pages 569–595, Sydney, Australia, August 1991.

[27] James Bruce, Stefan Zickler, Mike Licitra, and Manuela Veloso. CMDragons: Dynamic passing and strategy on a champion robot soccer team. In *International Conference on Robotics and Automation*, pages 4074–4079, Pasadena, CA, May 2008.

[28] Jennifer L. Burke, Robin R. Murphy, Michael D. Coovert, and Dawn L. Riddle. Moonlight in miami: a field study of human-robot interaction in the context of an urban search and rescue disaster response training exercise. *Human-Computer Interaction*, 19(1):85–116, June 2004.

[29] Sylvain Calinon and Aude Billard. Incremental learning of gestures by imitation in a humanoid robot. In *International Conference on Human-Robot Interaction*, pages 255–262, Arlington, VA, March 2007.

[30] Sylvain Calinon and Aude Billard. A probabilistic programming by demonstration framework handling constraints in joint space and task space. In *International Conference on Intelligent Robots and Systems*, pages 367–372, Nice, France, September 2008.

[31] Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, July 1997.

[32] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *International Conference on Machine Learning*, pages 161–168, Pittsburgh, PA, June 2006.

[33] Sonia Chernova and Manuela Veloso. Learning equivalent action choices from demonstration. In *International Conference on Intelligent Robots and Systems*, pages 1216–1221, Nice, France, September 2008.

[34] Sonia Chernova and Manuela Veloso. Multi-thresholded approach to demonstration selection for interactive robot learning. In *International Conference on Human-Robot Interaction*, pages 225–232, Amsterdam, The Netherlands, March 2008.

[35] Sonia Chernova and Manuela Veloso. Teaching collaborative multi-robot tasks through demonstration. In *International Conference on Humanoid Robots*, pages 385–390, Daejeon, S. Korea, December 2008.

[36] Sonia Chernova and Manuela Veloso. Interactive policy learning through confidence-based autonomy. *Journal of Artifical Intelligence Research*, 34(1):1–25, January 2009.

[37] Sachin Chitta and James P. Ostrowski. New insights into quasi-static and dynamic omnidirectional quadrupedal walking. In *International Conference on Intelligent Robots and Systems*, pages 2306–2311, Wailea, Hawaii, October 2001.

[38] Marco Colombetti and Marco Dorigo. Training agents to perform sequential behavior. *Adaptive Behavior*, 2(3):247–275, January 1994.

[39] Philip T. Cox and Trevor J. Smedley. Visual programming for robot control. In *IEEE Symposium on Visual Languages*, pages 217–224, Nova Scotia, CAN, September 1998.

[40] Jacob W. Crandall and Michael A Goodrich. Experiments in adjustable autonomy. In *International Conference on Systems, Man, and Cybernetics*, pages 1624–1629, Tuscan, AZ, October 2001.

[41] Lehel Csató. *Gaussian Processes - Iterative Sparse Approximations*. PhD thesis, Aston University, March 2002.

[42] Lehel Csató and Manfred Opper. Sparse Online Gaussian Processes. *Neural Computation*, 14(3):641–669, January 2002.

[43] Sanjoy Dasgupta, Daniel Hsu, and Claire Monteleoni. A general agnostic active learning algorithm. In *Neural Information Processing Systems*, pages 353–360, Vancouver, CAN, December 2007.

[44] Thomas Dean, Dana Angluin, Kenneth Basye, Sean Engelson, Leslie Kaelbling, Evangelos Kokkevis, and Oded Maron. Inferring finite automata with stochastic output functions and an application to map learning. *Machine Learning*, (1):81–108, 1995.

[45] Thomas Dean, Ken Basye, and John Shewchuk. *Machine Learning Methods for Planning and Scheduling*, chapter 1: Reinforcement Learning for Planning and Control. Morgan Kaufmann, 1992.

[46] Thomas G. Dietterich. The maxq method for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pages 118–126, Madison, WI, July 1998.

[47] Arnaud Doucet, Simon Godsill, and Christophe Andrieu. On sequential monte carlo sampling methods for bayesian filtering. *Statistics and Computing*, 10(3):197–208, July 2001.

[48] Paul Fearnhead. Particle filters for mixture models with an unknown number of components. *Statistics and Computing*, 14(1):11–21, January 2004.

[49] Emily B. Fox, Erik B. Sudderth, Michael I. Jordan, and Alan S. Willsky. An HDP-HMM for systems with state persistence. In *International Conference on Machine Learning*, pages 312–319, Helsinki, Finland, July 2008.

[50] Krzysztof Z. Gajos, Jacob O. Wobbrock, and Daniel S. Weld. Automatically generating user interfaces adapted to users' motor and vision capabilities. In *ACM Symposium on User Interface Software and Technology*, pages 231–240, Newport, Rhode Island, USA, October 2007.

[51] Erann Gat. *Artificial Intelligence and Mobile Robots*, chapter : On Three-layer Architectures. AAAI Press, 1997.

[52] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B Rubin. *Bayesian data analysis*. Chapman & Hall, 1995.

[53] Brian P. Gerkey, Richard T. Vaughan, and Andrew Howard. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In *International Conference on Advanced Robotics*, pages 317–323, Portugal, June 2003.

[54] Kenneth Y. Goldberg, Billy Chen, Rory Solomon, Steve Bui, Bobak Farzin, Jacob Heitler, Derek Poon, and Gordon Smith. Collaborative teleoperation via the internet. In *International Conference on Robotics and Automation*, pages 2019–2024, San Francisco, CA, April 2000.

[55] Robert L. Goldstone and Marco A. Janssen. Computational models of collective behavior. *Trends in Cognitive Sciences*, 9(9):424–430, September 2005.

[56] Eduardo Rodrigues Gomes and Ryszard Kowalczyk. Dynamic analysis of multiagent q-learning with ϵ-greedy exploration. In *International Conference on Machine Learning*, pages 369–376, Montreal, Quebec, Canada, June 2009.

[57] Daniel H Grollman and Odest Chadwicke Jenkins. Dogged learning for robots. In *International Conference on Robotics and Automation*, pages 2483 – 2488, Rome, Italy, April 2007.

[58] Daniel H Grollman and Odest Chadwicke Jenkins. Learning robot soccer skills from demonstration. In *International Conference on Development and Learning*, pages 276–281, London, UK, July 2007.

[59] Daniel H. Grollman, Odest Chadwicke Jenkins, and Frank Wood. Discovering natural kinds of robot sensory experiences in unstructured environments. *Journal of Field Robotics*, 23(11-12):1077–1089, November–December 2006.

[60] Mance E. Harmon and Stephanie S. Harmon. Reinforcement learning: a tutorial. Wright Laboratory, Centerville OH, 1996.

[61] Gillian Hayes and John Demiris. A robot controller using learning by imitation. In *International Symposium on Intelligent Robotic Systems*, Grenoble, France, July 1994.

[62] Xiaofei He, Deng Cai, and Partha Niyogi. Laplacian score for feature selection. In *Neural Information Processing Systems*, pages 507–514, Vancouver, CAN, December 2005.

[63] Frederik W. Heger and Sanjiv Singh. Sliding autonomy for complex coordinated multi-robot tasks: Analysis and experiments. In *Robotics: Science and Systems*, pages 17–24, Philidelphia, PA, August 2006.

[64] Hannah Hickey. Computer game's high score could earn the nobel prize in medicine. http://uwnews.org/article.asp?articleID=41558, May8 2008.

[65] Xuelei Hu and Lei Xu. Investigation on several model selection criteria for determining the number of cluster. *Neural Information Processing. - Letters and Reviews*, 4(1):1–10, July 2004.

[66] İlhan Uysal and H. Altay Güvenir. An overview of regression techniques for knowledge discovery. *Knowledge Engineering Review*, 14(4):319–340, December 1999.

[67] Tetsunari Inamura, Masayuki Inaba, and Hirochika Inoue. Acquisition of probabilistic behavior decision model based on the interactive teaching method. In *International Conference on Advanced Robotics*, pages 523–528, Tokyo, Japan, October 1999.

[68] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, Spring 1991.

[69] Odest C. Jenkins, German Gonzalez, and Matthew M. Loper. Interactive human pose and action recognition using dynamical motion primitives. *International Journal of Humanoid Robotics*, 4(2):365–385, June 2007.

[70] Nicholas K. Jong and Peter Stone. State abstraction discovery from irrelevant state variables. In *International Joint Conference on Artificial Intelligence*, pages 752–757, Edinburgh, U.K., August 2005.

[71] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, May 1998.

[72] Frédéric Kaplan, Pierre-Yves Oudeyer, Enikö Kubinyi, and Ádám Miklósi. Robotic clicker training. *Robotics and Autonomous Systems*, 38(3–4):197–206, September 2001.

[73] Elizabeth S. Kim and Brian Scassellati. Learning to refine behavior using prosodic feedback. In *International Conference on Development and Learning*, pages 205–210, London, UK, July 2007.

[74] Jens Kober, Betty Mohler, and Jan Peters. Learning perceptual coupling for motor primitives. In *International Conference on Intelligent Robots and Systems*, pages 834–839, Nice, France, September 2008.

[75] J. Zico Kolter, Pieter Abbeel, and Andrew Y. Ng. Hierarchical apprenticeship learning, with application to quadruped locomotion. In *Neural Information Processing Systems*, pages 769–776, Vancouver, CAN, December 2007.

[76] James Kramer and Matthias Scheutz. *Autonomous Robots*, 22(2):101–132, February 2007.

[77] James J. Kuffner Jr. and Steven M. LaValle. RRT-Connect: An efficient approach to single-query path planning. In *International Conference on Robotics and Automation*, pages 995–1001, San Francisco, CA, April 2000.

[78] Scott R. Kuindersma, Edward Hannigan, Dirk Ruiken, and Roderic A. Grupen. Dexterous mobility with the ubot-5 mobile manipulator. In *International Conference on Advanced Robotics*, Munich, Germany, June 2009.

[79] Micah Lapping-Carr, Odest Chadwicke Jenkins, Daniel H Grollman, Jonas N Schwertfeger, and Theodora R Hinkle. Wiimote interfaces for lifelong robot learning. In *AAAI Spring Symposium*, Menlo Park, CA, USA, March 2008.

[80] Tessa Lau. *Programming by Demonstration: a Machine Learning Approach*. PhD thesis, University of Washington, May 2001.

[81] Ethan Leland, Odest Chadwicke Jenkins, Brendan Dickenson, Dan Grollman, and Mark Moseley. The Brown University Robocup 2006 four-legged league team report. Master's thesis, Brown University, May 2006.

[82] David J.C. Mackay. *Neural Networks and Machine Learning*, chapter Introduction to Gaussian Processes. Springer-Verlag, 1998.

[83] Sridhar Mahadevan. Proto-value functions: developmental reinforcement learning. In *International Conference on Machine Learning*, pages 553–560, Bonn, Germany, August 2005.

[84] Adriano Mancini, Emanuele Frontoni, Andrea Ascani, and Primo Zingaretti. RoboBuntu: a Linux distribution for mobile robotics. In *International Conference on Robotics and Automation*, pages 2544–2549, Kobe, Japan, May 2009.

[85] Ruben Martinez-Cantin, Nando de Freitas, Arnaud Doucet, and José A. Castellanos. Active policy learning for robot planning and exploration under uncertainty. In *Robotics: Science and Systems*, pages 321–328, Atlanta, GA, USA, June 2007.

[86] Bruce Maxwell, Nicholas Ward, and Frederic Heckel. Game-based design of human-robot interfaces for urban search and rescue. In *Computer-Human Interaction Fringe*, Vienna, Austria, April 2004.

[87] Andrew Kachites Mccallum. *Reinforcement learning with selective perception and hidden state*. PhD thesis, The University of Rochester, May 1996.

[88] R. Andrew McCallum. Overcoming incomplete perception with utile distinction memory. In *International Conference on Machine Learning*, pages 190–196, Amherst, MA, USA, June 1993.

[89] Edward Meeds and Simon Osindero. An alternative infinite mixture of Gaussian process experts. In *Neural Information Processing Systems*, pages 883–890, Vancouver, CAN, December 2005.

[90] Francisco S. Melo and M. Isabel Ribeiro. Reinforcement learning with function approximation for cooperative navigation tasks. In *International Conference on Robotics and Automation*, pages 3321–3327, Pasadena, CA, May 2008.

[91] M. Alejandra Menchaca-Brandan, Andrew M. Liu, Charles M. Oman, and Alan Natapoff. Influence of perspective-taking and mental rotation abilities in space teleoperation. In *International Conference on Human-Robot Interaction*, pages 271–278, Arlington, VA, March 2007.

[92] Heiko Müller, Martin Lauer, Roland Hafner, Sascha Lange, Artur Merke, and Martin Riedmiller. Making a robot learn to play soccer using reward and punishment. In *German Conference on Advances in Artificial Intelligence*, pages 220–234, Osnabrück, Germany, September 2007.

[93] Bilge Mutlu and Jodi Forlizzi. Robots in organizations: the role of workflow, social, and environmental factors in human-robot interaction. In *International Conference on Human-Robot Interaction*, pages 287–294, Amsterdam, The Netherlands, March 2008.

[94] Radford M. Neal. Markov chain sampling methods for Dirichlet process mixture models. Technical Report 9815, University of Toronto, 1998.

[95] Andrew Y. Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, pages 663–670, Stanford, CA, June 2000.

[96] Monica Nicolescu and Maja J. Matarić. Natural methods for robot task learning: Instructive demonstration, generalization and practice. In *International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 241–248, Melbourne, AUS, July 2003.

[97] Monica N. Nicolescu, Odest C. Jenkins, Adam Olenderski, and Eric Fritzinger. Learning behavior fusion from demonstration. *Interaction Studies*, 9(2):319–352, June 2008.

[98] Curtis W. Nielsen, Michael A. Goodrich, and Robert W. Ricks. Ecological interfaces for improving mobile robot teleoperation. *Transactions on Robotics*, 23(5):927–941, October 2007.

[99] Stefano Nolfi and Jun Tani. Extracting regularities in space and time through a cascade of prediction networks: The case of a mobile robot navigating in a structured environment. *Connection Science*, 11(2):129–152, June 1999.

[100] Donald A. Norman. *The Design of Everyday Things*. Basic Books, 1998.

[101] Sarah Osentoski and Sridhar Mahadevan. Learning state-action basis functions for hierarchical MDPs. In *International Conference on Machine Learning*, pages 705–712, Corvalis, Oregon, June 2007.

[102] Nuno Otero, Aris Alissandrakis, Kerstin Dautenhahn, Chrystopher Nehaniv, Dag Sverre Syrdal, and Kheng Lee Koay. Human to robot demonstrations of routine home tasks: exploring the role of the robot's feedback. In *International Conference on Human-Robot Interaction*, pages 177–184, Amsterdam, The Netherlands, March 2008.

[103] Christopher J. Paciorek and Mark J. Schervish. Nonstationary covariance functions for Gaussian process regression. In *Neural Information Processing Systems*, pages 273–280, Vancouver, CAN, December 2003.

[104] Angelika Peer, Sandra Hirche, Carolina Weber, Inga Krause, Martin Buss, Sylvain Miossec, Paul Evrard, Olivier Stasse, Ee Sian Neo, Abderrahmane Kheddar, and Kazuhito Yokoi. Intercontinental multimodal tele-cooperation using a humanoid robot. In *International Conference on Intelligent Robots and Systems*, pages 405–411, Nice, France, September 2008.

[105] Jim Pitman. Combinatorial stochastic processes. Notes for Saint Flour Summer School, 2002.

[106] Nancy S. Pollard and Victor Brian Zordan. Physically based grasping control from example. In *SIGGRAPH/Eurographics symposium on Computer animation*, pages 311–318, Los Angeles, California, July 2005.

[107] Josep M. Porta, Nikos Vlassis, Matthijs T.J. Spaan, and Pascal Poupart. Point-based value iteration for continuous pomdps. *Journal of Machine Learning Research*, 7(11):2329–2367, November 2006.

[108] Harsha Prahlad, Ron Pelrine, Scott Stanford, John Marlow, and Roy Kornbluh. Electroadhesive robots – wall climbing robots enabled by a novel, robust, and electrically controllable adhesion technology. In *International Conference on Robotics and Automation*, pages 3028–3033, Pasadena, CA, May 2008.

[109] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software in Robotics*, Kobe, Japan, May 2009.

[110] Joaquin Quiñonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6(12):1939–1959, December 2005.

[111] Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. In *International Joint Conference on Artificial Intelligence*, pages 2586 – 2591, Hyderabad, India, January 2007.

[112] Carl Rasmussen and Zoubin Ghahramani. Infinite mixtures of Gaussian process experts. In *Neural Information Processing Systems*, pages 881–888, Vancouver, CAN, December 2001.

[113] Carl Edward Rasmussen. The infinite Gaussian mixture model. In *Neural Information Processing Systems*, pages 554–560, Vancouver, CAN, December 2000.

[114] Martin Riedmiller, Artur Merke, Manuel Nickschas, Wilhem Nowak, and Daniel Withopf. Brainstormers 2003 - team description. In *Robocup*, Padua, Italy, July 2003.

[115] Paul E. Rybski, Kevin Yoon, Jeremy Stolarz, and Manuela Veloso. Interactive robot task training through dialog and demonstration. In *International Conference on Human-Robot Interaction*, pages 255–262, Arlington, VA, March 2007.

[116] Joe Saunders, Chrystopher L. Nehaniv, and Kerstin Dautenhahn. Teaching robots by moulding behavior and scaffolding the environment. In *International Conference on Human-Robot Interaction*, pages 142–150, Salt Lake City, Utah, USA, March 2006.

[117] Tetsuo Sawaragi and Yukio Horiguchi. Ecological interface enabling human-embodied cognition in mobile robot teleoperation. *Intelligence*, 11(3):20–32, September 2000.

[118] Stefan Schaal, Auke Ijspeert, and Aude Billard. Computational approaches to motor learning by imitation. *Philosophical Transaction of the Royal Society of London*, 358(1431):537–547, March 2003.

[119] Stefan Schaal, Marc Toussaint, Giorgos Petkos, and Narayanan Edakunni. LWPR code. http://homepages.inf.ed.ac.uk/svijayak/software/LWPR/, 2003.

[120] Stephen Se, David Lowe, and Jim Little. Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks. *International Journal of Robotics Research*, 21(8):735–758, August 2002.

[121] Pierre Sermanet, Raia Hadsell, Marco Scoffier, Matt Grimes, Jan Ben, Ayse Erkan, Chris Crudele, Urs Muller, and Yann LeCun. A multi-range architecture for collision-free off-road robot naviffion. *Journal of Field Robotics*, 26(1):58–87, January 2009.

[122] Jivko Sinapov, Mark Wiemer, and Alexander Stoytchev. Interactive learning of the acousitc properties of household objects. In *International Conference on Robotics and Automation*, pages 2518–2524, Kobe, Japan, May 2009.

[123] William D. Smart and Leslie Pack Kaelbling. Effective reinforcement learning for mobile robots. In *International Conference on Robotics and Automation*, pages 3404–3410, Washington, D.C., May 2002.

[124] Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. In *Neural Information Processing Systems*, pages 1257–1264, Vancouver, CAN, December 2006.

[125] Matthijs T. Spaan and Nikos Vlassis. Perseus: Randomized point-based value iteration for pomdps. *Journal of Artifical Intelligence Research*, 24(1):195–220, January 2005.

[126] Mike Stilman, Koichi Nishiwaki, and Satoshi Kagami. Humanoid teleoperation for whole body manipulation. In *International Conference on Robotics and Automation*, pages 3175–3180, Pasadena, CA, May 2008.

[127] Mervyn Stone and Rodney J. Brooks. Continuum regression: Cross-validated sequentially constructed prediction embracing ordinary least squares, partial least squares and principal components regression. *Journal of the Royal Statistical Society*, 52(2):237–269, March 1990.

[128] Peter Stone and Manuela Veloso. Beating a defender in robotic soccer: Memory-based learning of a continuous function. In *Neural Information Processing Systems*, pages 896–902, Vancouver, CAN, December 1996.

[129] Peter Stone and Manuela Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110(2):241–273, June 1999.

[130] Peter Stone and Manuela M. Veloso. Layered learning. In *European Conference on Machine Learning*, pages 369–381, Barcelona, Catalonia, Spain, May 2000.

[131] Daniel Stronger and Peter Stone. Maximum likelihood estimation of sensor and action model functions on a mobile robot. In *International Conference on Robotics and Automation*, pages 2104–2109, Pasadena, CA, May 2008.

[132] Simone Stumpf, Erin Sullivan, Erin Fitzhenry, Ian Oberst, Weng-Keen Wong, and Margaret Burnett. Integrating rich user feedback into intelligent user interfaces. In *International Conference on Intelligent User Interfaces*, pages 50–59, Gran Canaria, Spain, January 2008.

[133] Richard Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Neural Information Processing Systems*, pages 1057–1063, Vancouver, CAN, December 2000.

[134] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[135] Richard S. Sutton, Doina Precup, and Satinder P. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, June 1999.

[136] Dag Sverre Syrdal, Kheng Lee Koay, Mick L. Walters, and Kerstin Dautenhahn. A Personalized Robot Companion? - the Role of Individual Differences on Spatial Preferences in HRI Scenarios. In *International Symposium on Robot & Human Interaction*, pages pp. 1143–1148, Jeju Island, Korea, August 2007.

[137] Leila Takayama, Wendy Ju, and Clifford Nass. Beyond dirty, dangerous and dull: what everyday people think robots should do. In *International Conference on Human-Robot Interaction*, pages 25–32, Amsterdam, The Netherlands, March 2008.

[138] Jun Tani and Stefano Nolfi. Learning to perceive the world as articulated: An approach for hierarchical learning in sensory-motor systems. *Neural Networks*, 12(7–8):1131–1141, October 1999.

[139] Matthew E. Taylor, Shimon Whiteson, and Peter Stone. Temporal difference and policy search methods for reinforcement learning: An empirical comparison. In *National Conference on Artificial Intelligence*, pages 1675–1678, Vancouver, CAN, July 2007.

[140] Yee Whye Teh, Michael I. Jordan, Matthew J. Beal, and David M. Blei. Hierarchical dirichlet processes. *Journal of the American Statistical Association*, 101(476):1566–1581, December 2006.

[141] Andrea L Thomaz and Cynthia Breazeal. Transparency and socially guided machine learning. In *International Conference on Development and Learning*, pages 3475–3480, Bloomington, IN, May 2006.

[142] Sebastian Thrun. Learning maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, February 1998.

[143] Sebastian B. Thrun and Tom M. Mitchell. Lifelong robot learning. Technical Report IAI-TR-93-7, University of Bonn, Jan 1993.

[144] J. Gregory Trafton, Alan C. Schultz, Magdalena Bugajska, and Farilee Mintz. Perspective-taking with robots: experiments and models. In *International Symposium on Robot & Human Interaction*, pages 580–584, Nashville, TN, August 2005.

[145] Katherine Tsui. Design and evaluation of a visual control interface of a wheelchair mounted robotic arm for users with cognitive impairments. Master's thesis, UMass Lowell, May 2004.

[146] Douglas L. Vail and Manuela M. Veloso. Feature selection for activity recognition in multi-robot domains. In *National Conference on Artificial Intelligence*, pages 1415–1420, Chicao, IL, July 2008.

[147] Meel Velliste, Sagi Perel, M. Chance Spalding, Andrew S. Whitford, and Andrew B. Schwartz. Cortical control of a prosthetic arm for self-feeding. *Nature*, 453(7198):1098–1101, June 2009.

[148] Deepak Verma and Rajesh P.N. Rao. Planning and acting in uncertain environments using probabilistic inference. In *International Conference on Intelligent Robots and Systems*, pages 2382–2387, Beijing, China, October 2006.

[149] Sethu Vijayakumar, Aaron D'Souza, and Stefan Schaal. Incremental online learning in high dimensions. *Neural Computation*, 17(12):2602–2634, December 2005.

[150] Sethu Vijayakumar, Aaron D'Souza, and Stefan Schaal. Lwpr: A scalable method for incremental online learning in high dimensions. Technical Report EDI-INF-RR-0284, Edinburgh, 2005.

[151] Daniel Vlasic, Rolf Adelsberger, Giovanni Vannucci, John Barnwell, Markus Gross, Wojciech Matusik, and Jovan Popović. Practical motion capture in everyday surroundings. *ACM Trans. Graph.*, 26(3):35–43, August 2007.

[152] Steven D. Whitehead and Dana H. Ballard. Learning to perceive and act by trial and error. *Machine Learning*, 7(1):45–83, January 1991.

[153] Frank Wood, Daniel H. Grollman, Katherine A. Heller, Odest C. Jenkins, and Michael Black. Incremental Nonparametric Bayesian Regression. Technical Report CS-08-07, Brown University Department of Computer Science, 2008.

[154] Mark Alistair Wood. *An Agent-Independent Task Learning Framework*. PhD thesis, University of Bath, July 2008.

[155] Mark P. Woodward and Robert J. Wood. Using Bayesian inference to learn high-level tasks from a human teacher. To appear in *International Conference on Artificial Intelligence and Pattern Recognition*, Orlando, FL, July 2009.