Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Teacher feedback to scaffold and refine demonstrated motion primitives on a mobile robot

Brenna D. Argall[a,*], Brett Browning[a], Manuela M. Veloso[b]

[a] Robotics Institute, Carnegie Mellon University, Pittsburgh PA, 15224, USA
[b] Computer Science Department, Carnegie Mellon University, Pittsburgh PA, 15224, USA

ARTICLE INFO

Article history:
Received 2 April 2010
Received in revised form
12 November 2010
Accepted 19 November 2010
Available online 22 December 2010

Keywords:
Demonstration learning
Policy reuse
Teacher feedback
Robot motion control

ABSTRACT

Task demonstration is an effective technique for developing robot motion control policies. As tasks become more complex, however, demonstration can become more difficult. In this work, we introduce an algorithm that uses corrective human feedback to build a policy able to perform a novel task, by combining simpler policies learned from demonstration. While some demonstration-based learning approaches do adapt policies with execution experience, few provide corrections within low-level motion control domains or to enable the linking of multiple of demonstrated policies. Here we introduce *Feedback for Policy Scaffolding (FPS)* as an algorithm that first evaluates and corrects the execution of motion primitive policies learned from demonstration. The algorithm next corrects and enables the execution of a more complex task constructed from these primitives. Key advantages of building a policy from demonstrated primitives is the potential for primitive policy reuse within multiple complex policies and the faster development of these policies, in addition to the development of complex policies for which full demonstration is difficult. Policy reuse under our algorithm is assisted by human teacher feedback, which also contributes to the improvement of policy performance. Within a simulated robot motion control domain we validate that, using FPS, a policy for a novel task is successfully built from motion primitives learned from demonstration. We show feedback to both aid and *enable* policy development, improving policy performance in success, speed and efficiency.

1. Introduction

Appropriate and effective motion behaviors are fundamental to the successful operation of mobile robots. The development of robust *policies*, or mappings from world states to robot actions, is complicated however by noisy observations and action execution uncertainty. Furthermore, policy development is often specific to a particular robot platform and application, and policy reuse for other platforms or application tasks is rare.

A large field of effective development approaches have the robot *learn* a policy, from training data or execution experience. One advantage to policy learning is that the learning algorithm itself may be reused for the development of other policies, though a given policy typically is still platform or application specific. *Learning from Demonstration (LfD)* is a technique that derives a policy from example executions of a target behavior by a teacher. This approach has seen success in a variety of robotics applications, and has the attractive characteristics of being an intuitive means for human teacher to robot learner knowledge transfer, as well as an accessible policy development technique for those who are not robotics-experts.

As tasks become more complex, however, demonstration can become more difficult. One practical extension of the LfD approach is to incorporate simpler behaviors learned from demonstration into larger task behaviors, especially if such tasks are too complex to demonstrate in full. Though scale-up techniques of this nature have been explored within other policy development approaches, the topic remains largely unaddressed within the LfD paradigm. Moreover, the ability to reuse and incorporate existing policies is a practical feature for any policy development approach, considering the challenge of designing robust control policies. Here we introduce *Feedback for Policy Scaffolding (FPS)* as an algorithm that builds a complex policy from component behaviors learned from demonstration.

One crucial issue when building a policy from simpler primitives is how to effectively incorporate the existing behaviors into the new policy. Whether the behaviors are to be composed or sequenced, behaviors must be linked and how this linking occurs is a key design decision. The FPS algorithm takes the approach of driving this process with human *feedback*. Multiple types of feedback are provided within the algorithm, including

[*] Corresponding author. Tel.: +1 41 78 671 3902.
E-mail addresses: bargall@ri.cmu.edu (B.D. Argall), brettb@cs.cmu.edu (B. Browning), veloso@cs.cmu.edu (M.M. Veloso).

0921-8890/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.robot.2010.11.004

binary flags for good performance and policy *corrections*. Feedback provided under our formulation does not require that the teacher revisit those states in need of attention, which is particularly attractive for complex tasks that may be inconvenient or infeasible to demonstrate completely. By *completely*, we mean the expression or attainment of desired characteristics of the policy behavior, for example achieving a final task goal, executing all components of a sequence of motions or performing a task with a high level of accuracy.

The FPS algorithm operates in two phases. In the first phase, individual policies are developed for multiple motion primitives, from demonstrations of the target behaviors by a task expert. Each policy furthermore is refined using human teacher feedback. During the second phase, a single more complex policy is derived from these primitives, and executions with the complex policy on a novel task are evaluated. By providing corrections on this execution, the FPS algorithm develops a policy able to execute the more complex, undemonstrated task. Policy development is achieved without ever requiring the complete demonstration of the novel task: only the primitive behaviors have been demonstrated in full.

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

The following section provides background and related literature that motivates and supports this work. The FPS algorithm then is introduced Section 3, providing details of behavior building through teacher feedback and algorithm execution. Sections 4 and 5 present an empirical implementation of FPS, including descriptions of the task and domain, and experimental results from both feedback phases. The conclusions of this work are presented in Section 6, along with directions for future research.

2. Background and related work

We begin this section with a presentation of related work on policy development and improvement within demonstration learning, followed by motivations for building policies from behavior primitives and teacher feedback.

2.1. Learning from Demonstration

Learning from Demonstration (LfD) is a policy development technique in which teacher executions of a desired behavior are recorded and a policy is subsequently derived from the resulting dataset [1,2]. We formally define the world to consist of states $S \in \mathbb{R}^{\ell}$ and actions $A \in \mathbb{R}^{n}$. As we do not assume that state is fully observable, the learner has access to observed state Z through a mapping $S \to Z \in \mathbb{R}^{m}$. Typically the teacher provides multiple demonstrations, and the resulting sequences of observation–action pairs comprise a set D of behavior examples. We represent a teacher demonstration $d_j \in D$ as t_j pairs of observations and actions, such that $d_j = \{(\mathbf{z}^i_j, \mathbf{a}^i_j)\}_{i=0}^{t_j}$, $\mathbf{z}^i_j \in Z$, $\mathbf{a}^i_j \in A$. The robot generalizes from the dataset D to derive a policy $\pi : Z \to A$, thus learning a mapping from the set of world observations Z to robot actions A.

When recording and executing demonstrations the issue of *correspondence* is key, where teacher demonstrations do not directly map to the robot learner due to differences in sensing or motion [3]. Demonstration techniques that are best able to minimize the introduction of correspondence issues into an LfD system have the passive learner record from its own sensors while under the control of the teacher. One such technique is *teleoperation*, where the human remotely operates the learner platform during the demonstration execution. Demonstrations recorded through human teleoperation are used in a variety of applications, including flying a robotic helicopter [4], object grasping [5,6], as well as obstacle avoidance and navigation [7]. Teleoperation also is applied to a wide variety of simulated domains, for example robot soccer [8].

Policy derivation amounts to building a predictor that will reproduce the actions $\mathbf{a}^i \in D$ from the observations $\mathbf{z}^i \in D$. Many approaches exist within LfD to derive a policy from the demonstration data [1], the most popular of which either directly approximate the underlying function mapping observations to actions, or approximate a state transition model and then derive a policy using techniques such as *Reinforcement Learning* [9]. Our work derives a policy under the first approach, with function approximation being performed via regression techniques, since the prediction space of our function approximation – that is, the action space of our target application of low-level motion control – is continuous. A wealth of regression approaches exist [10], and we highlight that any are compatible with the FPS algorithm.

To have a robot learn from its execution performance, or *experience*, is a valuable policy improvement tool, and there are LfD approaches that incorporate learning from experience into their algorithms. For example, execution experience is used to update state transition models [11] and reward-determined state values [12]. Other approaches provide more demonstration data, driven by learner requests for more data [13,14] as well as more teacher-initiated demonstrations [15].

Our approach similarly provides new example state-action mappings, but the source for these mappings is *not* more teacher demonstration. There are some LfD limitations that more teacher demonstrations cannot address, for instance correspondence discrepancies between the teacher and learner. Moreover, the need to visit states in order to provide execution information is a drawback if certain world states are difficult to reach or dangerous to visit, for example that lead to a rover falling over a cliff. We employ a technique for policy improvement that *synthesizes* new example state-action mappings from teacher feedback and learner executions, without requiring state revisitation by the teacher to provide appropriate behavior information [16]. In particular, the behavior (specifically, an observation–action pair) exhibited by the learner during policy execution is *corrected*, and the feedback-corrected pair is treated as new training data (full details provided in Section 3.1).

Other LfD approaches that provide policy corrections do so within action spaces that are discrete and with actions of significant time duration; that is, not low-level motion control domains, where actions take on continuous values and are rapidly sampled. For example, the correct action from a discrete set is provided by a human teacher to update a high-level action classifier [17], and to update the structure of a hierarchical Neural Network of behaviors [18].

2.2. Behavior primitives and teacher feedback

Our approach builds a complex policy from the demonstration of simpler behavior primitives and teacher feedback, rather than demonstrates the complex task in full. One motivation for our approach is that the demonstrated motion primitives may provide a base for multiple complex behaviors. Through the reuse of these primitives, the effort required to develop policies for the complex behaviors reduces.

A second motivation is that as behaviors become more complex, demonstrating the complete behavior can become more difficult. In this case, the teacher may be able to demonstrate behavior primitives for a task but not the task in full, or provide higher quality demonstrations for subsets of the task behavior. Under our algorithm, feedback may be used to improve, possibly even to enable, the policy at transition points that link demonstrated behavior primitives. In doing so, it enables expression of the full task behavior, without requiring its full demonstration.

A final motivation is that reaching all states encountered during task execution can become increasingly difficult as domains become more complex. States may be infeasible, inconvenient or undesirable to reach for a variety of reasons that only compound as domain and task complexity increases. A drawback to demonstration is the need to visit such states in order to provide task execution information in that area. One of the notable benefits of providing feedback under the framework we employ (Section 3.1) is that states do *not* need to be revisited to provide execution feedback.

Within LfD, hand-coded behavior primitives are used to build larger tasks learned from demonstration [18], demonstrated tasks are decomposed into a library of primitives [12,19] and behavior primitives are demonstrated and then explicitly combined into a new policy by a human [20]. Closest to our work is that of [19], where a robotic marble maze and humanoid playing air hockey reuse learned primitives, and furthermore refine the policy with execution experience. In this work, the complex task is demonstrated in full and the behavior primitives are extracted using hand-written rules. Policy improvement is accomplished through an automated binary reward signal for task failure, which is used to adjust regression weights on the policy prediction. By contrast, our approach demonstrates the primitives individually, rather than the full task. Policy improvement is accomplished by generating new behavior examples, from human feedback that corrects practice executions of both the primitives and full task.

3. Algorithm: Feedback for Policy Scaffolding

This section presents our *Feedback for Policy Scaffolding (FPS)* algorithm [21]. The section begins with an overview of our techniques for providing teacher feedback. Following this, the FPS algorithm is presented in detail.

3.1. Teacher feedback and policy derivation

Here we outline[1] the framework through which we provide feedback (F3MRP), and the form taken by that feedback (advice-operators), as well as the employed policy derivation technique.

3.1.1. Teacher feedback

Our approach provides teacher feedback through the framework *Focused Feedback for Mobile Robot Policies (F3MRP)*. The F3MRP framework operates at the stage of low-level motion control, where actions are continuous-valued and sampled at high frequency, and is intended for mobile robot applications in particular.[2] A visual presentation of the 2-D ground path of the execution serves as an interface through which the teacher indicates areas of poor policy performance, by selecting segments of an execution path that are to receive feedback. The path displayed at each timestep consists of a point corresponding to the location of the robot on the ground, and a vector to indicate robot heading. The path sequence furthermore is played back in real-time, providing an indication of execution speed. This interface simplifies the challenge of providing feedback to policies sampled at a high frequency. Visual indications of data support for the policy predictions made during the execution furthermore assist the teacher in the selection of execution segments.

Execution *corrections* are offered through *advice-operators*, which function by performing numerical computations on the observations or actions of executed data points; an operator $f : (s, a) \rightarrow (s, a) \in \mathbb{R}^{n\times m}$ thus maps an observation–action pair to the augmented space of observations and actions. Advice-operators are commonly defined between the student and teacher, and provide continuous-valued corrections on a learner execution without requiring the teacher to provide the exact *value* for the corrections. Instead, the teacher needs only to select from a finite list of operators, and indicate (through the F3MRP framework) the portion of the execution to which the operator should be applied. The feedback-modified points are treated as new training data for the policy. In addition to advice-operators, the FPS algorithm also provides feedback in the form of positive binary credit, which results in learner-executed datapoints being added to the dataset unmodified, and thus may equivalently be viewed as an identity function advice-operator.

In summary, the FPS algorithm requires a mechanism by which to provide feedback on learner executions. The primary feedback type employed offers policy corrections, that are provided through advice-operators. The advice-operator technique modifies points produced during a learner execution, and thus requires the selection of individual execution points. Point selection is accomplished through the F3MRP framework, which provides an interface through which points may be selected, as well as an indication of data support for a policy prediction.

3.1.2. Policy derivation

Behavior primitive policies are initially built from teacher demonstrations of a target behavior. Using regression techniques, a policy mapping world observations to robot actions is derived. The specific regression technique used in our empirical implementation is a form of Locally Weighted Learning [22], where action $\boldsymbol{a}^t \in \mathbb{R}^n$ is predicted through an averaging of datapoints in D, weighted by their kernelized distance to $\boldsymbol{z}^t \in \mathbb{R}^m$. More specifically, the actions of the datapoints within D are weighted by a kernelized distance $\phi(\boldsymbol{z}^t, :)$ between their associated datapoint observations and the current observation $\boldsymbol{z}^t$. Thus,

$$\boldsymbol{a}^t = \sum_{(\boldsymbol{z}_i,\boldsymbol{a}_i)\in D} \phi(\boldsymbol{z}^t, \boldsymbol{z}_i) \cdot \boldsymbol{a}_i \tag{1}$$

$$\phi(\boldsymbol{z}^t, \boldsymbol{z}_i) = \frac{\mathrm{e}^{(\boldsymbol{z}_i-\boldsymbol{z}^t)^T \Sigma^{-1}(\boldsymbol{z}_i-\boldsymbol{z}^t)}}{\sum_{\boldsymbol{z}_j\in D} \mathrm{e}^{(\boldsymbol{z}_j-\boldsymbol{z}^t)^T \Sigma^{-1}(\boldsymbol{z}_j-\boldsymbol{z}^t)}},$$

$$\Sigma^{-1} = \begin{bmatrix} \sigma_0^2 & & & \\ & \sigma_1^2 & & \\ & & \ddots & \\ & & & \sigma_{m-1}^2 \end{bmatrix} \tag{2}$$

where Σ^{-1} is a constant diagonal matrix that scales each observation dimension (tuned through 10-fold Cross Validation to optimize the least-squared-error on action prediction). In our empirical implementation the distance computation is Euclidean

[1] For full details of F3MRP and advice-operators, we refer the reader to [16].

[2] For safety reasons, motion control is typically sampled at a rate of tens or hundreds of cycles per second on mobile robots (though of course this depends heavily on the speed capabilities of the mobile platform); our system for example runs at 30 Hz.

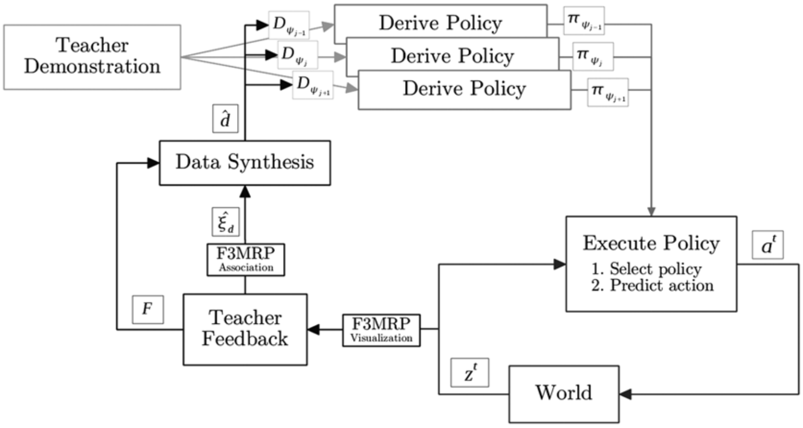


Fig. 1. Policy demonstration and adaptation under the FPS algorithm.

and the kernel Gaussian; the parameter Σ^{-1} furthermore embeds the bandwidth of the Gaussian kernel.[3]

3.2. Algorithm execution

Execution of the FPS algorithm occurs in two phases, presented respectively in Algorithms 1 and 2. Given a set Ψ of N primitive behaviors, the first phase develops a policy π_{ψ_j} for each primitive behavior $\psi_j \in \Psi, j = 1 \dots N$, producing a set of policies Π. The second phase develops a policy for a more complex behavior, building on the primitive policies in Π. A schematic of the overall algorithm is provided in Fig. 1.

3.2.1. Phase 1: primitive policy development

The development of each primitive policy begins with teacher demonstration. Example observation–action pairs recorded during demonstrations of primitive behavior ψ_j produce the initial dataset $D_{\psi_j} \in \mathbf{D}$ and policy $\pi_{\psi_j} \in \Pi$ (Algorithm 1, line 4). This initial policy is then refined during practice runs consisting of learner executions and teacher feedback.

Algorithm 1 *Feedback for Policy Scaffolding: Phase 1*

```
1:  Given $\mathbf{D}$
2:  initialize $\Pi \leftarrow \{\,\}$
3:  for all behavior primitives $\psi_j \in \Psi$ do
4:      initialize $\pi_{\psi_j} \leftarrow \texttt{policyDerivation}(D_{\psi_j})\,,\ D_{\psi_j} \in \mathbf{D}$
5:      while practicing do
6:          initialize $\xi_d \leftarrow \{\},\ \xi_p \leftarrow \{\}$
7:          repeat
8:              predict $\{\,\boldsymbol{a}^t, \tau^t\,\} \leftarrow \pi_{\psi_j}(\boldsymbol{z}^t)$
9:              execute $\boldsymbol{a}^t$
10:             record $\xi_d \leftarrow \xi_d \cup (\boldsymbol{z}^t, \boldsymbol{a}^t)\,,\ \xi_p \leftarrow \xi_p \cup (x^t, y^t, \theta^t, \tau^t)$
11:         until done
12:         advise $\{\mathcal{F}, \hat{\xi}_p\} \leftarrow \texttt{teacherFeedback}(\,\texttt{F3MRP}(\xi_p)\,)$
13:         associate $\hat{\xi}_d \leftarrow \texttt{F3MRP}(\,\xi_d, \hat{\xi}_p\,)$
14:         update $D_{\psi_j} \leftarrow \texttt{applyFeedback}(\,D_{\psi_j}, \mathcal{F}, \hat{\xi}_d\,)$
15:         rederive $\pi_{\psi_j} \leftarrow \texttt{policyDerivation}(\,D_{\psi_j}\,)$
16:     end while
17:     add $\Pi \leftarrow \Pi \cup \pi_{\psi_j}$
18: end for
19: return $\Pi$
```

During the learner *execution* portion (lines 7–11) of a practice run, the learner executes the task and records information from the execution into data traces ξ_d and ξ_p. At each timestep the learner observes the world, predicting action $\boldsymbol{a}^t$ with support τ^t (discussed in Section 3.3.2) according to policy π_{ψ_j} (line 8). Action $\boldsymbol{a}^t$ is executed and recorded, along with observation $\boldsymbol{z}^t$, into the prediction trace $\xi_d \in \mathbb{R}^{m \times n}$ (line 10). The information recorded in the trace ξ_d will be used for the policy update, when synthesizing data from feedback. The global position x^t, y^t and heading θ^t of the mobile robot, and data support τ^t of the regression prediction, are recorded into the position trace $\xi_p \in \mathbb{R}^4$. Information recorded into ξ_p will be used by the F3MRP framework, when visually presenting the path taken by the robot on the ground during execution.

During the teacher *feedback* and policy *update* portion of a practice run (lines 12–15), the teacher provides feedback in response to the learner execution performance. Using the visual presentation of ξ_p provided by the F3MRP interface, the teacher indicates a segment $\hat{\xi}_p \subseteq \xi_p$ of the learner execution, along with feedback $\mathcal{F}$ for that segment (line 12). F3MRP associates the segment $\hat{\xi}_p$ of the position trace with the appropriate segment $\hat{\xi}_d \subseteq \xi_d$ of the prediction trace (line 13). Feedback $\mathcal{F}$ may take two forms. The first is a binary credit, to indicate areas of good performance, and the second is an advice-operator, to correct the execution within this segment. Both feedback forms produce new data. In the case of binary credit, the selected segment $\hat{\xi}_d$ is added to D_{ψ_j} as executed, without any modifications. In the case of an advice-operator f, each point in $\hat{\xi}_d$ is mapped to a new observation–action pair, such that $f : (\mathbf{z}^i, \mathbf{a}^i) \to (\hat{\mathbf{z}}^i, \hat{\mathbf{a}}^i), \forall (\boldsymbol{z}^i, \boldsymbol{a}^i) \in \hat{\xi}_d$, producing a set $\hat{d}$ of feedback-modified data pairs. These data are added to the set $D_{\psi_j} \leftarrow D_{\psi_j} \cup \hat{d}$, constituting the dataset update.

3.2.2. Phase 2: policy scaffolding

The development of the complex policy builds on the primitive policies developed during Phase 1 of the FPS algorithm. Features that distinguish Phase 2 include (i) that development does *not* begin with teacher demonstration of the (complex) task, (ii) the automated selection between the action predictions of multiple policies and (iii) the automated selection of a dataset to receive any new data synthesized from teacher feedback on the learner executions.

Phase 2 begins with the initialization of more demonstration datasets. Specifically, N empty datasets are generated, each associated with one primitive policy. Notationally, let new dataset $D_{\psi_{i+N}}$ be associated with existing primitive dataset D_{ψ_i}, resulting in a total of $2N$ datasets $D_{\psi_j} \in \mathbf{D}, j = 1 \dots 2N$. Colloquially,

[3] While we acknowledge that a more sophisticated regression technique could perhaps improve policy performance, LWL regression has the advantage of transparency and clarity, which was important for this initial validation of the FPS algorithm given that our feedback techniques synthesize data. We furthermore emphasize that the focus of this work is on how to use teacher feedback to produce *new* data, that refines policy performance and builds new behavior, rather than on how to better utilize *existing* demonstration data.

Algorithm 2 *Feedback for Policy Scaffolding: Phase 2*

```
1:  Given $\mathit{\Pi}, \mathbf{D}$
2:  initialize $D_{\psi_{j=(n+1)\dots 2n}} \leftarrow \{\,\}$
3:  while practicing do
4:      initialize $\xi_d \leftarrow \{\},\ \xi_p \leftarrow \{\},\ \xi_s \leftarrow \{\}$
5:      repeat
6:          select $\pi_{\psi_j} \leftarrow$ policySelection$(\boldsymbol{z}^t)$, $\pi_{\psi_j} \in \mathit{\Pi}$
7:          predict $\{\boldsymbol{a}^t, \tau^t\} \leftarrow \pi_{\psi_j}(\boldsymbol{z}^t)$
8:          execute $\boldsymbol{a}^t$
9:          record $\xi_d \leftarrow \xi_d \cup (\boldsymbol{z}^t, \boldsymbol{a}^t)$, $\xi_p \leftarrow \xi_p \cup (x^t, y^t, \theta^t, \tau^t)$, $\xi_s \leftarrow \xi_s \cup$
            $(\tau^t, \psi^t = \psi_j)$
10:     until done
11:     advise $\{\digamma, \hat{\xi}_p\} \leftarrow$ teacherFeedback( F3MRP $(\xi_p)$ )
12:     associate $\{\hat{\xi}_d, \hat{\xi}_s\} \leftarrow$ F3MRP( $\xi_d, \xi_s, \hat{\xi}_p$ )
13:     for all $(\boldsymbol{z}^i, \boldsymbol{a}^i) \in \hat{\xi}_d,\ (\tau^i, \psi^i) \in \hat{\xi}_s$ do
14:         select $D_{\psi_j} \leftarrow$ datasetSelection$(\tau^t, \psi^t)$, $D_{\psi_j} \in \mathbf{D}$
15:         update $D_{\psi_j} \leftarrow$ applyFeedback$(D_{\psi_j}, \digamma, (\boldsymbol{z}^i, \boldsymbol{a}^i))$
16:         rederive $\pi_{\psi_j} \leftarrow$ policyDerivation$(D_{\psi_j})$, $\pi_{\psi_j} \in \mathit{\Pi}$
17:     end for
18: end while
19: return $\mathit{\Pi}$
```

call dataset $D_{\psi_{i+N}}$ the *feedback dataset* associated with primitive dataset D_{ψ_i}. Some of the new data synthesized during Phase 2 will be added to these feedback datasets; this process is described further within the details of feedback incorporation (Section 3.3.2). Both the primitive policies *and* the policies derived from the feedback datasets are considered for selection during execution of the more complex policy.

Refinement of the complex policy proceeds with learner *execution* (Algorithm 2, lines 5–10), teacher *feedback* (line 11) and policy *update* (lines 13–17), as in Phase 1 but with the following distinguishing characteristics.

- The learner now executes with the more complex policy, whose operation proceeds in two steps:
  1. Select between all contributing policies $\pi_{\psi_j} \in \mathit{\Pi}$ (line 6).
  2. Predict action $\boldsymbol{a}^t$ according to $\pi_{\psi_j}(\boldsymbol{z}^t)$ (line 7).

  The details of automated policy selection are provided in Section 3.3.1.
- After the application of teacher feedback, datasets are individually selected to receive each feedback-modified datapoint. For each recorded point $(\boldsymbol{z}^t, \boldsymbol{a}^t)$ in the indicated segment $\hat{\xi}_d \subseteq \xi_d$, dataset selection is determined by the support τ^t and tag ψ^t (line 14).

The tag ψ^t indicates which policy was selected for execution, and is set to ψ_j. At each step of the learner execution, both data support τ^t and tag ψ^t are recorded into a third data trace ξ_s (line 9). The details of automated dataset selection are provided in Section 3.3.2.

3.3. Scaffolding multiple policies

Two key factors when building a policy under FPS are how to select between the primitive behaviors, and how to incorporate teacher feedback into the built-up policy. The design of each of these factors within the FPS algorithm are discussed here, with a later section (6.3) detailing a few of the iterative stages from their development.

3.3.1. Selecting primitive policies

Primitive selection is treated as a classification problem, and we highlight that the algorithm does not place any restrictions on the type of classification technique that may be employed. Classification in our empirical implementation computes for each primitive ψ_j a kernelized distance $\phi(\boldsymbol{z}^t, \boldsymbol{z}_i)$ between query point $\boldsymbol{z}^t$ and each point $\boldsymbol{z}_i \in D_{\psi_j}$ (as in Eq. (2), but with Σ^{-1} in this case tuned to optimize classification error). A weight for policy $\psi_j \in \Psi$ is produced by summing the k largest kernel values $\phi(\boldsymbol{z}^t, \boldsymbol{z}_i)$, $\boldsymbol{z}_i \in D_{\psi_j}$; equivalent to selecting the k nearest points in D_{ψ_j} to query $\boldsymbol{z}^t$ ($k = 5$). The policy with the highest weight is then selected for execution.

Primitive selection thus assumes primitives to occupy nominally distinct areas of the observation space, and relies on a formulation for the observation features that captures aspects of the world that are unique to the execution of each primitive policy. We highlight however that similar assumptions are frequently tied to policy selection paradigms, and further argue that the assumptions are reasonable for many application domains. In particular, policy selection frequently is triggered by specific sensor readings or particular world states (e.g. [18,19]), and to incorporate the sensor reading (or state observation) into the feature computation effectively isolates policies in different areas of the observation space according to that feature dimension. For example, two primitives developed for our validation domain are *turn left* and *turn right*, which should be triggered by track curvature. The observation features accordingly incorporate a notion of track curvature, and demonstrations in left-versus right-curving areas of the track therefore do occupy distinct areas of the observation space.

3.3.2. Incorporating teacher feedback

From a technical standpoint, the question of how to incorporate teacher feedback into the policy boils down to into which dataset the new feedback-modified data should be added. Unlike during the refinement of the primitive policies – where a single dataset is associated with a given policy, and accordingly receives any new data produced as a result of its execution – when providing feedback on the scaffolded policy, multiple primitive policies and their associated datasets contribute to the behavior execution. Furthermore, complete behavior for the scaffolded policy was never demonstrated, and so no corresponding "scaffolded-behavior dataset" exists. A choice must be made, therefore, that determines into which underlying dataset a synthesized datapoint is added.

Before providing the details of this choice, let us establish two ideas. First: we assume that a primitive whose behavior does not match the intended behavior of the more complex policy is not being incorporated into the complex policy in the first place. Thus, in state-space areas covered by the dataset of any primitive under consideration, the behavior of the primitive matches the intended behavior of the more complex policy. Second: recall that every synthesized datapoint in $\hat{d}$ derives from an execution point, whose action was predicted by a single primitive policy (Algorithm 2, line 6). Two factors determine into which dataset a new datapoint is added: the policy (tagged by $\psi^t = \psi_j \in \Psi$) that predicted the execution point, and the measure of data support τ^t for that prediction.

- If the policy that made the prediction is a primitive policy ($j \in 1 \dots N$), the point is added to its dataset *if* the prediction had strong data support. Otherwise, the point is added to the *feedback* dataset (as defined in Section 3.2.2) associated with the primitive.
- If the policy that made the prediction is a feedback policy ($j \in (N + 1) \dots 2N$), the point is always added to its dataset, regardless of dataset support.

Prediction support is determined in the following manner. For a given dataset D_{ψ_j}, the 1-Nearest Neighbor Euclidean distances between all points in the set are modelled as a Poisson[4] distribution, parameterized by λ with mean $\mu = \lambda$ and standard

[4] A Poisson formulation was chosen since the distance calculations never fall below, and often cluster near, zero; behavior which is better modelled by a Poisson rather than Gaussian distribution.

deviation $\sigma = \sqrt{\lambda}$. Frequency counts are computed for k bins on the distance data when computing λ ($k = 50$). A threshold $\tau_{\psi_j} = \mu_{\psi_j} + \kappa\sigma_{\psi_j}$ on strong prediction support is defined for each policy $\pi_{\psi_j} \in \Pi$, $j = 1 \ldots 2N$, where κ is set by hand based on empirical evidence ($\kappa = 5$). Thus a prediction made by policy π_{ψ_j} for query point $\mathbf{z}^t$ with distance $\ell_{\mathbf{z}^t,\psi_j}$ to the nearest point in D_{ψ_j} is classified as strongly supported if $\ell_{\mathbf{z}^t,\psi_j} < \tau_{\psi_j}$ and weakly supported otherwise.[5] The motivation behind this approach is to avoid adding data to a primitive dataset that conflicts with the given primitive behavior. Only points that were close enough to a primitive dataset to be strongly supported during action prediction are presumed to exhibit behavior that is similar to the primitive policy.

Table 1
Advice-operators for the racetrack driving task.

	Operator	Parameter
0	Modify speed, static (*rot*)	(cw ccw)
1	Modify speed, static (*tr*)	(dec inc)
2	Modify speed, fractional (*rot*)	(dec inc zero)
3	Modify speed, fractional (*tr*)	(dec inc zero)
4	Modify speed, incremental fractional (*rot*)	(dec inc)
5	Modify speed, incremental fractional (*tr*)	(dec inc)
6	Adjust both, fractional	(dec inc)
7	Adjust turn, fractional	(loosen tighten)
8	Adjust turn, incremental fractional	(loosen tighten)

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

Table 2
Description of observation and action dimensions for the racetrack driving task.

Dim	Observation	Description
0	ν^t	Current translational speed
1	ω^t	Current rotational speed
2	a	Coefficients for polynomial approximation of track edge (third-order approximation, i.e. $y = ax^3 + bx^2 + cx + d$)
3	b	
4	c	
5	d	
Dim	**Action**	**Description**
0	$\hat{\nu}^t$	Predicted translational speed
1	$\hat{\omega}^t$	Predicted rotational speed

4. Experimental setup

This section presents the experimental details of the application of the FPS algorithm to a simulated motion control domain, as well as a description of the various policies developed for empirical evaluation. The ability of the algorithm to select primitive policies whose behavior is appropriate for a given area of the state space also is confirmed.

4.1. *Racetrack driving task and domain*

The domain chosen to validate the FPS policy building algorithm is a simulated differential drive robot within a racetrack driving environment. The task consists of the robot driving along the racetrack, with two failure criteria: the robot either stops moving or crosses a track boundary. The dimensions and parameters set in this simulated domain are based on real world execution with a Segway RMP robot [23]. Robot motion is propagated by simple differential drive simulation of the robot position

$$
\begin{aligned}
x^t &= x^{t-1} + (\nu^t + \epsilon^t_\nu) \cdot \cos(\theta^t) \cdot \mathrm{d}t \\
y^t &= y^{t-1} + (\nu^t + \epsilon^t_\nu) \cdot \sin(\theta^t) \cdot \mathrm{d}t \qquad (3) \\
\theta^t &= \theta^{t-1} + (\omega^t + \epsilon^t_\omega) \cdot \mathrm{d}t
\end{aligned}
$$

where (x^t, y^t) and θ^t are respectively the robot position and orientation within the global frame, and $\mathrm{d}t$ the length scale between timesteps ($\mathrm{d}t = 0.0\bar{3}3$ s). Robot rotational and translational speeds ν^t, ω^t are subject to Gaussian noise, $\epsilon^t_\nu \sim \mathcal{N}(0, 0.01)$ and $\epsilon^t_\omega \sim \mathcal{N}(0, 0.01)$. Dynamic limitations on the robot constrain its speeds $\left(\nu \in [0.0, 3.5]\ \frac{\mathrm{m}}{\mathrm{s}}, \omega \in [-5.0, 5.0]\ \frac{\mathrm{rad}}{\mathrm{s}}\right)$ and accelerations $\left(\dot{\nu} \in [-10.0, 10.0]\ \frac{\mathrm{m}}{\mathrm{s}^2}, \dot{\omega} \in [-10.0, 10.0]\ \frac{\mathrm{rad}}{\mathrm{s}^2}\right)$. The track borders are represented as sets of points along two parallel curved lines within a global frame. To simplify interactions between the robot and the track boundaries, the robot is represented as a point and the track width (1.5 m) is shrunk by moving each border towards the centerline by an amount equal to the radius of the robot (0.3 m).

The robot observes the world through a single monocular camera and wheel encoders, and state observations are sampled at 30 Hz. Data provided from both sensing modalities are assumed to be preprocessed. Specifically, processed data from the simulated wheel encoders provides the position estimates of Eq. (3). Processed data from the simulated camera provides a set of points corresponding to noisy ($\sim\mathcal{N}(0, 0.01)$) observations of the track border within the ground plane; that is, a simulation that projects image pixels identified as track borders into the ground plane (where the classification of a pixel as a track border point is assumed to be given). The simulated camera is forward facing and observes track borders within its field of view (130°, 5 m).

The robot is controlled by setting target translational and rotational speeds. Demonstrations are performed via teleoperation by a human,[6] who controls the robot motion by decreasing/increasing its translational and rotational speeds. The motion control advice-operators for this task were developed using the operator-scaffolding approach of Argall [16], and are presented in Table 1. Note that, according to the terminology of this approach, operators 0–5 are the baseline operators for each robot action (rotational and translational speed), and operators 6–8 were built through the operator-scaffolding technique.[7]

At each execution timestep, the robot computes a local track representation by fitting a third-order polynomial to track border points visually observed in the current and recent-past timesteps. Observation features and action dimensions are detailed in Table 2.

Lastly, we note that the aim of the developed policy is to reactively drive on a racetrack. The robot has no a priori map of the track, nor does it attempt to build a global map during the execution. A local map of sorts is estimated online, through the polynomial approximation of the track that is computed from the simulated sensor data at each timestep.

4.2. *Policy development and evaluation*

Here we describe the policies that result from the various steps taken during policy development under FPS, as well as the evaluation metrics that will be used in the performance analysis of Section 4.

[5] A similar measure is used by F3MRP when providing a visual indication of data support. Specifically, for each policy $\pi_{\psi_j} \in \Pi$ multiple thresholds $\tau^\kappa_{\psi_j} = \mu_{\psi_j} + \kappa\sigma_{\psi_j}$ are defined, by setting different values for κ (where μ_{ψ_j} and σ_{ψ_j} again derive from the Poisson model of 1-NN distances). A different color is associated with each threshold, and used in the graphical depiction of the 2-D execution path on the ground.

[6] The human operator, and feedback provider, in these experiments was one of the authors, and not a novice.

[7] We refer the reader to [16] for full details of this advice-operator development (operator-scaffolding) approach.

4.2.1. Development

The demonstrated motion primitives $\{\psi_j\}_{j=1}^{3}$ for this domain are: *turn right*, *go straight*, *turn left*. Demonstrations are performed via human teleoperation, by decreasing or increasing the target translational and rotational speeds by static amounts as the robot moves along the racetrack. Position updates are noisy, and computed according to Eq. (3). The following steps are taken during policy development:

1. *Demonstrate the motion primitives, and derive initial policies*. Teacher demonstration of each primitive is performed multiple (3) times on an appropriate track subset (Fig. 2, left). From each dataset a policy is derived, producing a set of policies PD_I.
2. *Provide feedback on the motion primitive policies' performance*. Learner execution with each policy in PD_I on its respective track subset is observed by the teacher, who provides feedback, and the resulting feedback-generated data is added to the executing policy's dataset. This observation–*feedback*–update cycle constitutes a single *practice run*, continues to the satisfaction of the teacher and results in final feedback versions of the primitive policies, referred to collectively as the set PF_F.
3. *Derive an initial scaffolded policy from the resultant primitive policies*. An initial *single* scaffolded policy SF_I, that selects between the primitive policies in PF_F, is built.
4. *Provide feedback on the scaffolded policy's performance*. Learner executions with SF_I on the full track are observed by the teacher, and feedback-generated data is added to either the executing policy's dataset *or* its associated feedback dataset (as described in Section 3.3.2). The observation–feedback–update cycle continues to the satisfaction of the teacher, and results in the final feedback-scaffolded policy SF_F.

For comparative purposes, we also evaluate the approach of providing more demonstrations in response to policy performance. The approach closely follows the policy development steps just outlined, but the teacher provides more teleoperation demonstrations instead of feedback. The result is a set of policies developed according to an observation–*demonstration*–update paradigm, specifically: the set PD_F of final more-demonstration versions of the primitive policies, the baseline policy SD_I scaffolded from them, and the final scaffolded more-demonstration policy SD_F.

4.2.2. Evaluation

Each of the developed policies are evaluated on racetrack executions. In particular,

- Each of the primitive policies (contained in sets PD_I, PF_F, PD_F) is evaluated on the track subset appropriate to their respective primitive behavior (Fig. 2, left).
- Each of the scaffolded policies (SF_I, SF_F, SD_I, SD_F) is evaluated on the full track (Fig. 2, right).

Executions begin from a random initial (forward facing) heading and position along the track start line (or beginning of the appropriate track subset, for the primitive policies). Executions proceed along the track until the robot runs off the track, stops prematurely or reaches the finish line. Policy performance is measured according to the following metrics:

- *Completion* is measured by the *percentage* of the track subset (for primitive policy executions) or full track (for scaffolded policy executions) driven by the robot before the execution ends.
- *Speed* is measured by the average translational speed during an execution.
- *Efficiency* is measured by execution time, which is governed jointly by speed and the execution ground path. This measure is computed exclusively for executions with 100% completion; for incomplete executions, that by definition aborted early, time is not a useful measure.

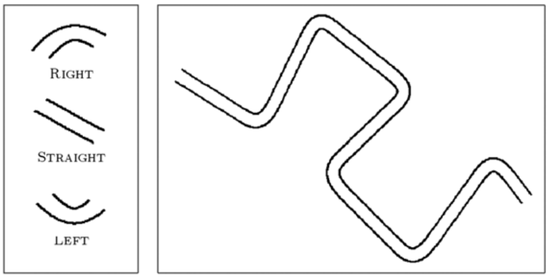


Fig. 2. Primitive subset regions (left) of the full racetrack (right).

4.3. Appropriate behavior selection

The FPS algorithm selects between behavior primitives based solely on the distribution of their data within the observation space; the demonstration teacher does not provide any additional information about when the behavior of a given primitive is appropriate for execution within the complex task. Before presenting performance results (Section 5), here we show that the FPS selection paradigm does in fact choose behavior that is appropriate for the various portions of the complex task execution.

4.3.1. Overview

Fig. 3 shows examples of primitive policy selection, during full track executions with the final FPS policy SF_F. Five example executions are shown as panels A–E. Across a panel, a single execution trace is displayed in triplicate, with each plot corresponding to the selection of a different policy primitive. Within a single plot, the full execution trace is displayed (cyan line) and those points for which the *primitive policy* (of the given column) was selected are reinforced as blue circles, while those for which the primitive behavior's *associated feedback policy* was selected are reinforced as red circles. Each execution begins at the track start line (green line, lower right corner) and ends at either the finish line (magenta line, upper left corner) or the point at which the robot drove off the track (e.g. panel C).

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

4.3.2. Discussion

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

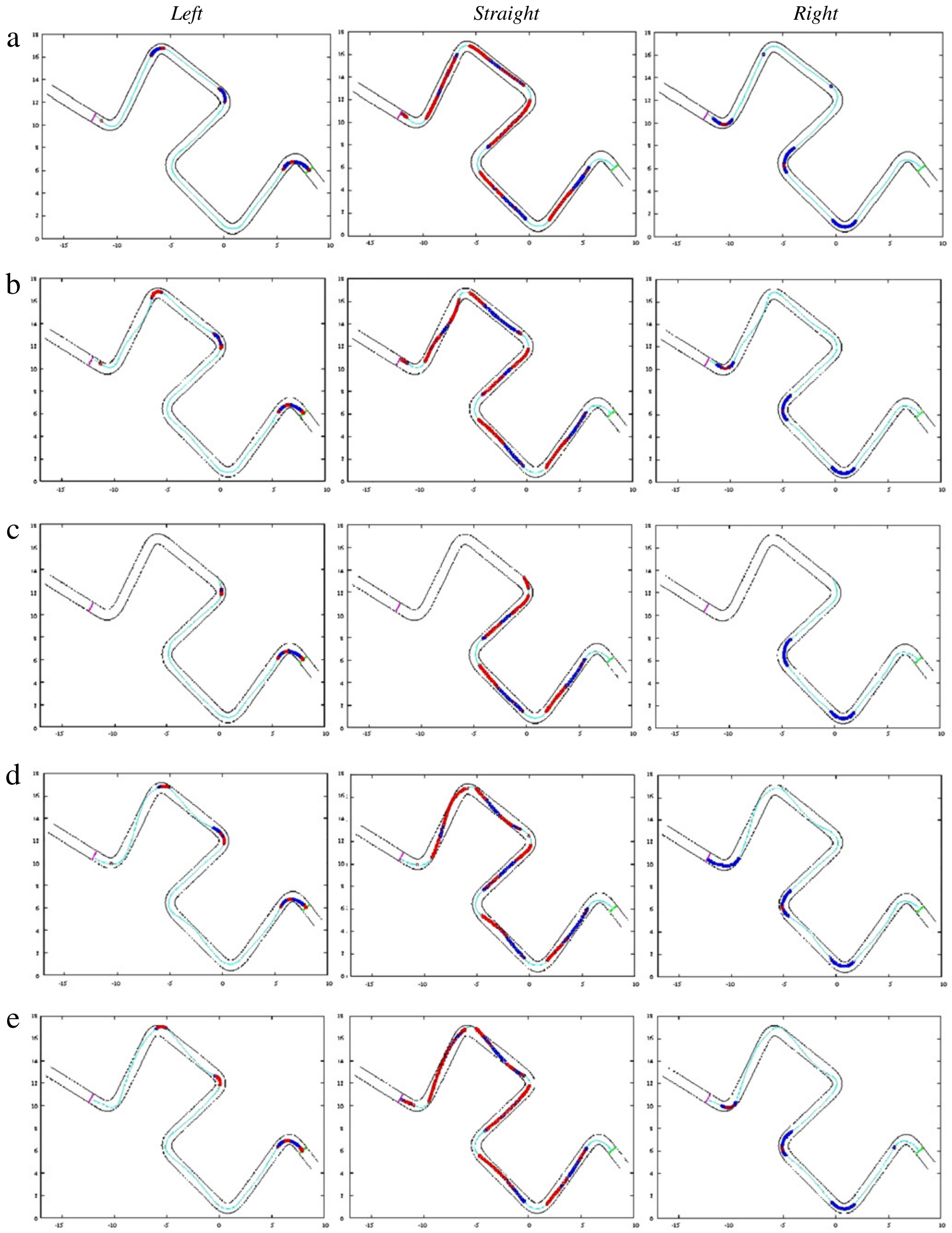


Fig. 3. Example complex driving task executions (rows), showing primitive behavior selection (columns); see text for details.

is determined by the distribution of datapoints within the set, and thus by the teacher's behavior when demonstrating (Section 3.3.2).

Other selection differences may be observed between panel executions. For example, between panels A and B, note the differences between selection of the *straight* primitive policy at the topmost track segment. The execution in panel A selects the *straight-feedback* policy almost exclusively (red circles), while the execution in panel B by comparison prefers the *straight* policy (blue circles). Panel A's execution runs close to the right-hand border, while B's runs close to the left-hand border, indicating that these are state-space areas supported respectively by the *straight-feedback* and *straight* datasets (further suggesting that the demonstration teacher tended to run near the right-hand track border, and consequently did not visit the area of the state space occupied by the left-hand border). Another example is panel C, which shows an execution that runs off the track before completion. Specifically, it runs off the track after a left turn. Comparisons between panel C's selection of behavior *left* at that turn and the selection in all other panels shows that the duration of the selection of the *left* policy (first column) is shorter in panel C, suggesting that the robot transitioned into its selection of the *straight* behavior too early, before completing the left turn. Not all selection transitions are this delicate however; for example compare the exit strategies of panels D and E on the final track turn. Panel D continues to select the *right* policy until reaching the finish line, while panel E transitions to select the *straight-feedback* policy, yet *both* exit the turn and complete the track successfully.

We draw two general conclusions from this qualitative analysis. The first is that our primitive selection paradigm is performing sensibly, if not perfectly (e.g. panel C). The second is that the effects of primitive policy selection are complex, and a policy may be alternately sensitive or insensitive to primitive selection at various points during task execution. We therefore highlight the investigation of alternate approaches to primitive behavior selection as an interesting area for future research (Section 6.3).

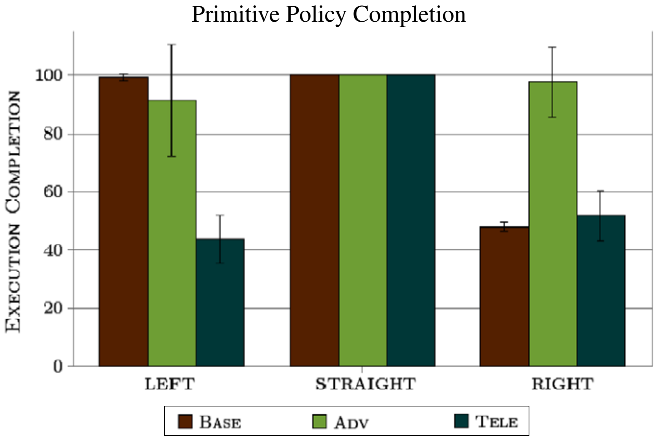


Fig. 4. Percent task completion with each of the primitive behavior policies (average of 50 executions, 1-standard deviation error bars). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 3
Execution performance of the primitive policies (average over practice runs, 1-standard deviation).

Policy	Completion (%)	Speed (mean) $\left(\frac{m}{s}\right)$	Efficiency (s)
Demo, left	**99.2 ± 1.3**	1.0 ± 0.0	2.8 ± 0.0
Feedback, left	91.3 ± 19.3	**1.5 ± 0.4**	**1.8 ± 0.4**
More Demo, left	43.8 ± 8.2	0.6 ± 0.0	–
Baseline, straight	100.0 ± 0.0	0.6 ± 0.0	5.7 ± 0.1
Feedback, straight	100.0 ± 0.0	**2.7 ± 0.1**	**1.3 ± 0.0**
More Demo, straight	100.0 ± 0.0	1.7 ± 0.3	3.1 ± 0.6
Baseline, right	48.0 ± 1.4	0.6 ± 0.0	–
Feedback, right	**97.6 ± 12.0**	**1.7 ± 0.0**	**1.9 ± 0.1**
More Demo, right	51.8 ± 8.5	0.7 ± 0.0	–

5. Performance results

In this section, the FPS algorithm is confirmed to successfully learn motion control primitives through a combination of demonstration and teacher feedback. Furthermore, a policy built on these primitives with feedback was able to successfully execute a more complex, novel behavior. In each phase of the FPS algorithm, teacher feedback was found to be critical to the development and performance improvement of policies. Additionally, the policies that resulted from FPS feedback far outperformed those that received only further teacher demonstrations.

5.1. Motion primitives learned from demonstration

Three motion primitives were successfully learned from demonstration and human feedback. Full details of the results from Phase 1 of the FPS algorithm are presented in Table 3 (the presented speed results are for the mean translational speed). For clarity of presentation, the figures and tables of this section (5.1) use the label *Baseline* in reference to the primitive policies in PD_I, *Feedback* to those in PF_F and *More Demonstration (More Demo)* to those in PD_F.

5.1.1. Policy development

Policy development under FPS terminates at the discretion of the feedback teacher. Here the teacher decided to terminate development once a policy displayed either satisfactory performance or no further performance improvement. The number of practice runs required to achieve the termination criterion varied for each behavior primitive, and for each policy improvement technique (Table 4). However, the more-demonstration approach consistently required more practice runs, which produced a significantly *larger* number of new datapoints. Since the larger number of datapoints did not improve policy performance over that of the feedback datasets (Table 3, *MoreDemo* vs. *Feedback*), the additional datapoints are likely redundant or unnecessary for successful policy behavior.

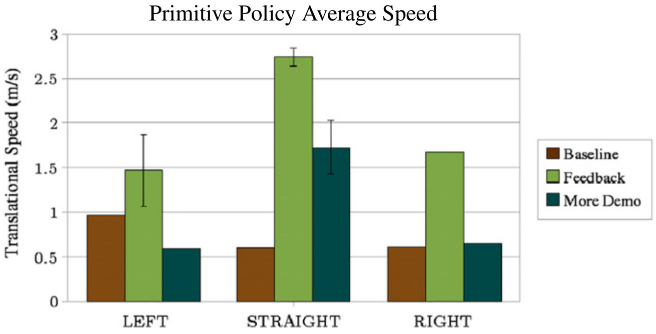


Fig. 5. Average translational execution speed with each of the primitive behavior policies (average of 50 executions, 1-standard deviation error bars). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

5.1.2. Performance results

The motion primitives were successfully learned by the FPS algorithm, as evidenced by the ability of each primitive policy in PF_F to complete executions on the corresponding track subsets (Table 3, *Feedback*). Fig. 4 shows the percent completed execution, for each primitive policy on its respective track subset (average of 50 executions, 1-standard deviation error bars).

For the *turn right* primitive behavior, the initial Baseline policy (in PD_I) was unable to complete the task (Fig. 4, category *right*, brown bar). The *turn right* policy (in PF_F) resulting after Phase 1 development under the FPS algorithm, however, was able to complete the task (Fig. 4, *right*, green bar). Furthermore, executions with this policy were much faster on average (Fig. 5, *right*, green bar). By contrast, the *turn right* policy (in PD_F) resulting from more teleoperation demonstrations was not able to complete the task, or even to improve upon the performance or speed of the baseline policy (Figs. 4–5, *right*, blue bar). Note that the teacher had a harder time demonstrating the right-hand turn; in particular, providing consistent demonstrations proved challenging, which resulted in a more spread dataset (as observed in Section 4.3.2).

In contrast to *turn right*, for the *go straight* primitive behavior the initial Baseline policy (in PD_I) was able to complete the task (Fig. 4, category *straight*, brown bar). The initial policy executed the task extremely slowly however (Fig. 5, *straight*, brown bar). By contrast, the *go straight* policy (in PF_F) resulting after Phase 1 development under the FPS algorithm was able to increase execution speeds (Fig. 5, *straight*, green bar), such that the *average* speed over the execution approached the maximum speed of the robot $\left(3.0\ \frac{m}{s}\right)$, all without compromising the completion success of the executions. The *go straight* policy (in PD_F) resulting from more teleoperation demonstrations also improved execution speeds over the baseline policy, but not as dramatically (Fig. 5, *straight*, blue bar).

For the *turn left* primitive behavior, the initial Baseline policy (in PD_I) was able to complete the task (Fig. 4, category *left*, brown bar). The *turn left* policy (in PF_F) resulting after Phase 1 development under the FPS algorithm somewhat degraded this performance (Fig. 4, *left*, green bar) by occasionally going off the track, a consequence of the policy being more aggressive with respect to speed than the baseline policy (Fig. 5, *left*, green bar). The advantage of this aggression was much more efficient executions (Table 3,

Table 4
Number of practice runs and datapoints produced in the development of each primitive behavior policy.

	Feedback				More Demo			
	Right	Straight	Left	Total	Right	Straight	Left	Total
Practice runs	23	27	8	58	36	36	12	84
New datapoints	561	426	252	1239	2846	1630	965	5441

Table 5
Execution performance of the complex policies (average of 50 executions, 1-standard deviation error bars, T = translational, R = rotational).

Policy	Completion (%)	Speed T (mean, max) $\left(\frac{\mathrm{m}}{\mathrm{s}}\right)$	Speed R (max) $\left(\frac{\mathrm{rad}}{\mathrm{s}}\right)$
Feedback Initial	6.3 ± 1.7	0.4 ± 0.2, 1.5 ± 0.4	0.9 ± 0.3
*Feedback**	63.3 ± 28.5	2.2 ± 0.2, 3.0 ± 0.2	2.1 ± 0.2
Feedback Final	**97.5 ± 7.9**	**2.3 ± 0.0**, **3.1 ± 0.0**	**2.6 ± 0.1**
Demo Initial	5.9 ± 0.2	0.6 ± 0.0, 0.7 ± 0.1	0.1 ± 0.1
Demo Final	13.7 ± 2.4	1.0 ± 0.1, 1.5 ± 0.6	1.0 ± 0.2

Feedback Left, Efficiency), but at the cost of occasional incomplete executions. The *turn left* policy (in PD_F) resulting from more teleoperation demonstrations decreased the performance of the initial policy in both completion and speed (Figs. 4–5, *left*, blue bar). Presumably more demonstrations in this case created ambiguous areas for the policy, a complication that would perhaps clear up with the presentation of more disambiguating demonstrations.

5.2. Undemonstrated task learned from primitives and feedback

A policy able to execute a more complex, novel behavior was successfully developed through the scaffolding of the learned primitive policies and the incorporation of teacher feedback. The complex task here was driving the full racetrack. Before any practice runs with teacher feedback, the complex policy, derived solely from selection between the developed feedback primitive policies PF_F, was unable to complete the task. Performance improvement over 160 practice runs is presented in Fig. 6. A new iterative policy results from each practice run, which consists of a single execution from the start line. Each plot point represents an average of 10 track executions with a given iterative policy, and a regularly sampled subset of the iterative policies were evaluated in this manner (sampled every 10 policies, 17 iterative policies evaluated in total, 1-standard deviation error bars). This constitutes Phase 2 of the FPS algorithm, after which the learner was able to consistently execute the complex task in full.

5.3. Improvement in complex task performance

Beyond the development of a policy *able* to perform the more complex task, Phase 2 of the FPS algorithm further enabled performance *improvement* such that task executions became faster and more efficient. Performance details are discussed in the following sections, and summarized within Table 5 (average of 50 executions).

5.3.1. Policy development

Development of the final complex *demonstration* policy SD_F was aborted early by the teacher due to the extremely slow rate of policy improvement which, while not entirely stagnant, was prohibitively slow. The final FPS policy SF_F therefore received feedback on more practice runs than the final more-demonstration policy SD_F (159 vs. 74). An additional comparison is therefore provided in the following sections, that considers an *iterative* FPS policy (*Feedback**). This policy is not the final FPS policy, but rather the result of development after 74 practice runs, the same number as the final demonstration policy (Table 6).

Table 6
Number of practice runs and datapoints produced in the development of each complex policy.

	Feedback Final	Feedback*	Demo Final
Practice runs	159	74	74
Datapoints	4503	2448	8520

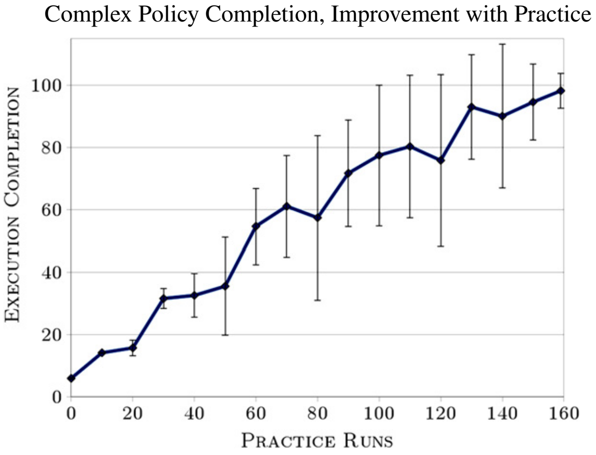


Fig. 6. Percent task completion during complex policy practice (average of 10 executions, 1-standard deviation error bars).

5.3.2. Performance results: completion

As noted above, the initial FPS policy SF_I, derived exclusively from the primitive feedback policies in PF_F, was not able to complete this task (Fig. 7, *Feedback Initial*, average of 50 executions, 1-standard deviation error bars). Neither was the initial policy SD_I derived exclusively from the primitives in PD_F that received more teacher demonstrations (*Demo Initial*). Both of these policies performed similarly poorly in the measure of execution completion.

The final policy SF_F that resulted after Phase 2 of the FPS algorithm, however, was able to consistently complete the task (*Feedback Final*). By contrast, the policy SD_F that resulted from more teleoperation demonstrations (*Demo Final*) was not able to complete the task, though it did nominally improve upon the performance the initial demonstration policy (*Demo Initial*).

Against the iterative feedback policy, the final demonstration policy similarly did not measure well. The iterative policy (*Feedback**) significantly outperformed the final demonstration policy (*Demo Final*) on the completion measure, though it does not yet perform as successfully or as consistently as the final FPS policy (*Feedback Final*).

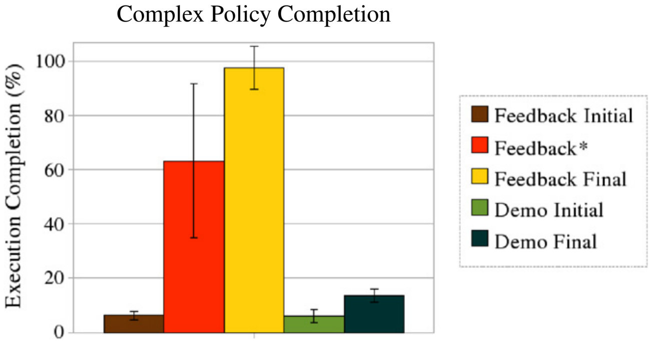


Fig. 7. Percent task completion with full track behavior policies.

5.3.3. *Performance results: speed*

Policies were additionally evaluated for speed. Since the full track is much longer than any of the primitive subsets, and thus offers more opportunities to reach higher speeds, the maximum speeds attained during executions are provided in addition to the average speed (Table 5, Fig. 8).

The speed performance results closely resemble those of the completion performance. Namely, the final FPS policy (*Feedback Final*) far outperformed both the initial FPS policy (*Feedback Initial*) as well as the final demonstration policy (*Demo Final*). The final demonstration policy did offer some improvement over the initial demonstration policy (*Demo Initial*), but not nearly as much as the iterative FPS policy provided for comparison (*Feedback**). Interesting to note is that this iterative FPS policy produced speeds similar to the final FPS policy in nearly all measures, though with slightly larger standard deviations (Table 5). This suggests that performance *consistency*, in addition to execution completion, was a motivation for continuing practice and development beyond this iterative policy.

6. Discussion

This section provides discussion on the development and performance of the FPS algorithm. Strengths of the algorithm are highlighted (Sections 6.1 and 6.2), and two key design decisions are detailed (Section 6.3).

6.1. *Improved performance with feedback*

Within our empirical validation, all FPS policies outperformed the comparative policies developed from teacher demonstration alone. While these exclusively demonstrated policies were able to nominally perform the primitive behavior tasks, albeit with varying degrees of success and always less adeptly than the FPS policies, they were never able to perform the more complex task behavior.

This work underlines the benefit, afforded through the F3MRP framework, of not needing to revisit world states in order to provide feedback. State formulations in this work were 6-dimensional and continuous-valued; to accurately revisit a state under such a formulation is effectively impossible. The segment selection technique of the F3MRP framework is more accurate at focusing feedback to the required states than is teacher demonstration, which must revisit the states to provide corrective feedback. The result here was smaller datasets paired with superior policy performance.

6.2. *Application to task hierarchies and real robots*

Our prior work (e.g. [24]) employed a hierarchical state machine architecture [25] for autonomous control of multiple Segway RMP robots. The state machine consisted of a set of control states, each of which encoded a policy, and transitions between different control states occurred as a function of observed state. We arranged these state machines into hierarchies, where a state machine could transition to other state machines. The power in a behavior hierarchy lies in its task decomposition; that is, by enabling a larger task to be easily decomposed into sub-tasks that can be solved as independent problems, and possibly reused for other similar problems.

A drawback to the hierarchical state machine approach is that for a real robot, the control architecture is often hand coded. Typically, it is not task decomposition which is difficult. Rather, most of designer effort focuses on developing the primitive policies and tuning the transitions between them. Moreover, the performance of such policies is highly dependent on robot hardware and the environment. In our experience, building such policies has often been tedious and time-consuming.[8] The FPS policy building algorithm offers an alternative to the drawback of hand-coding the primitive control policies. Here primitive control behaviors are *learned* from demonstration, and refined through teacher feedback. Furthermore, the *transitions* between primitive behaviors are also learned. These transitions are a function of teacher feedback and the world observation formulation (though relaxing the observation formulation requirement is a target area for future research).

Though the FPS algorithm to date has been validated in simulation only, we expect good results on a real robot, given our past success in using corrective feedback to refine behavior on the Segway RMP [16]. Our results additionally suggest that building complex behaviors with our scaffolding algorithm will be significantly more efficient and successful than providing teacher demonstrations alone, an advantage that becomes all the more relevant when demonstrations involve executing physical actions on a mobile robot platform within the real world. As tasks, domains and robot platforms become increasingly complex, policy development under FPS undoubtedly will become increasingly involved, likely requiring a larger number of primitive behaviors and more sophisticated interactions between them when scaffolding. Certainly the FPS algorithm will fare no worse in this regard however than other approaches, like state machines, that depend on task decomposition and thus also require defining a larger number of control behaviors and transitions between them.[9] In light of this, we expect the benefits afforded by the FPS algorithm, of efficiently learning control behaviors and transitions, to scale well with domain complexity.

6.3. *Primitive behavior selection and feedback incorporation*

One key design decision in the development of the FPS algorithm was how to select between primitive behaviors. The initial version of the algorithm performed no explicit primitive selection, and instead relied exclusively on the local nature of the employed regression technique and the assumption that primitives occupy distinct areas of the state space. The performance of this implicit primitive selection, however, was found to be substandard, as there did exist some areas of

[8] We acknowledge of course that in other domains (e.g. with many free degrees of freedom to control) providing sufficient demonstrations might also be tedious.

[9] Domains for which task decomposition is not an effective approach for the development of a complex behavior are beyond the scope of this article.

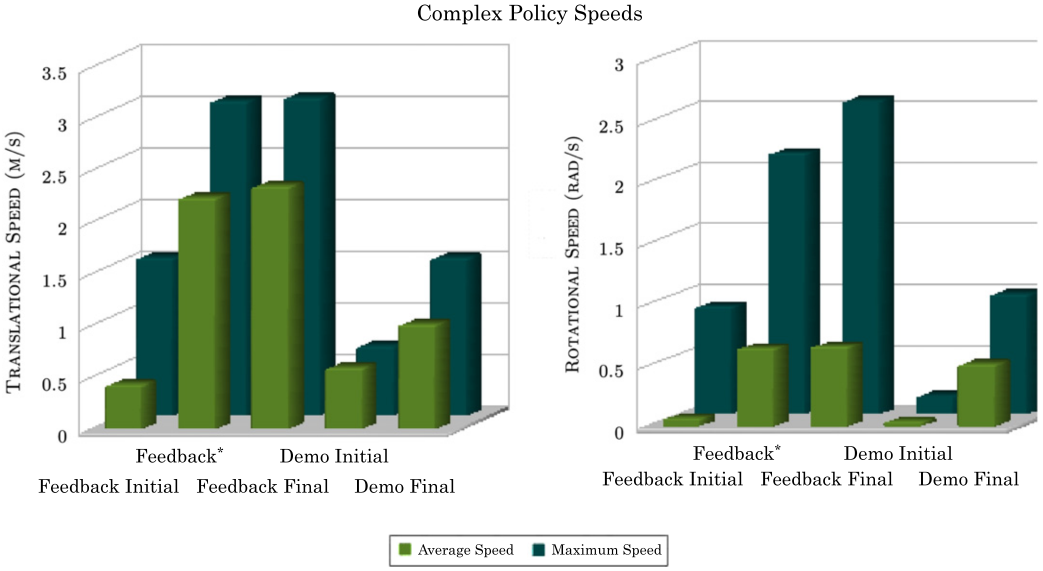


Fig. 8. Average and maximum translational (left) and rotational (right) speeds with full track behavior policies.

overlap between primitive datasets. Violations of the no-overlap assumption resulted in the mixing of data from different primitives into a single regression prediction, and consequently poor performance if their actions were in conflict.[10]

Later versions of the algorithm therefore relaxed the assumption of non-overlapping state-spaces between primitives. Datapoints were restricted to make predictions only with other points of the same primitive type; effectively isolating demonstrations of a particular primitive type into separate datasets, and treating each primitive as its own policy. In the final algorithm version, selection at execution time took the primitive with the highest weight, according to a kernelized weighting over the k-nearest points within each primitive dataset. A variety of formulations for how to select between primitive behaviors are possible and, as the effects of selection on policy performance can be complex (Section 4.3), we highlight the investigation of alternate paradigms as a rich area for future work.

Like so many policy development approaches, the success of a policy derived under the FPS algorithm thus depended on the appropriateness of the feature space to the task and domain; that is, on the ability of the computed features to uniquely describe the distinct actions necessary for task completion. The only difference within the FPS algorithm in comparison to more typical policy development paradigms is that the feature space was used for both higher-level classification, to select between the primitive policies, and lower-level regression, to predict an action for motion control.[11]

Another key design decision in the development of the FPS algorithm was how to incorporate teacher feedback into the complex policy. Early versions of the algorithm added a new point to the dataset of the primitive policy that predicted the execution point receiving feedback. For query points close to the dataset providing the prediction, and whose target behavior thus was close to the behavior of the dataset's primitive, this approach was reasonable and on par with correcting a single policy. However, if the more complex policy visited areas of the state space that were unvisited during the development of the primitive policies, the prediction of the nearest primitive might conflict with the intended behavior of the more complex policy (especially if there were aspects of the target behavior that were not captured by any of the policy primitives). In this case, corrections that took a step in the direction of the final policy behavior stepped *away* from the target primitive behavior. Adding such points to the primitive dataset could compromise its intended behavior, due to overgeneralization.

Later versions of the algorithm therefore placed new (weakly supported, as defined in Section 3.3.2) data into separate feedback datasets. The policies derived from these sets were selected between, as the primitive policies, during execution. Initially all new data was placed into a single feedback dataset, but the resulting feedback policy was found at times to over-generalize since its dataset contained the weakly data-supported predictions of *all* primitive policies, which varied markedly in their respective intended behaviors. To mitigate over-generalization, the final algorithm therefore associated a feedback dataset with *each* policy primitive. The decision to associate one feedback dataset with each primitive policy was somewhat arbitrary however, having been made based on empirical support of good policy performance (Fig. 3). One could envision many other valid approaches to determining the number of feedback datasets, which is an open area for future research.

7. Conclusion

We have introduced the Feedback for Policy Scaffolding (FPS) algorithm as an approach that uses teacher feedback and demonstration to build complex policy behaviors. More specifically, the algorithm operates in two phases. The first phase develops primitive motion behaviors from teacher demonstration and feedback provided through the F3MRP framework. The second phase scaffolds these primitive behaviors into a policy able to perform a more complex task. Feedback is again employed, in this case to assist scaffolding and thus enable the development of a sophisticated motion behavior. There are many potential gains to using primitives learned from demonstration to build more complex policies; for example, the potential for primitive policy reuse, and the development policies for which demonstration of the complete task is difficult.

[10] For example, if the rotational speed of 1.0 $\frac{\text{rad}}{\text{s}}$ for primitive *turn left* vs. 1.0 $\frac{\text{rad}}{\text{s}}$ for *turn right* become a mixed action prediction that does not turn at all, 0.0 $\frac{\text{rad}}{\text{s}} = \frac{1}{2}\left(1.0\ \frac{\text{rad}}{\text{s}}\right) + \frac{1}{2}\left(-1.0\ \frac{\text{rad}}{\text{s}}\right)$.

[11] An interesting side note is that if the design of the feature space is only moderately poor, for example intuitive for a human policy designer but suboptimal for the task execution, it is possible that providing policy corrections will take a step towards addressing the design deficiency. In this manner, the FPS algorithm has an advantage over other policy development paradigms.

The FPS algorithm was implemented within a simulated motion control domain, that tasked a differential drive robot with driving along a racetrack. Primitive behavior policies, which represent simple motion components of this task, were successfully learned through demonstration and teacher feedback. A policy able to accomplish a more complex behavior, i.e. to drive the full track, was successfully developed from the primitive behaviors and teacher feedback. In both development phases, empirical results showed policy performance to improve with teacher feedback. Furthermore, all FPS policies outperformed the comparative policies developed from teacher demonstrations alone.

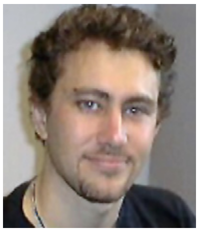
Acknowledgements

The research is partly sponsored by the Boeing Corporation under Grant No. CMU-BA-GTA-1, BBNT Solutions under subcontract No. 950008572, via prime Air Force contract No. SA-8650-06-C-7606, the United States Department of the Interior under Grant No. NBCH-1040007 and the Qatar Foundation for Education, Science and Community Development. The views and conclusions contained in this document are solely those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the US government or any other entity.

References

[1] B. Argall, S. Chernova, M. Veloso, B. Browning, A survey of robot learning from demonstration, Robotics and Autonomous Systems 57 (2009) 469–483.
[2] A. Billard, S. Callinon, R. Dillmann, S. Schaal, Robot programming by demonstration, in: B. Siciliano, O. Khatib (Eds.), Handbook of Robotics, Springer, New York, NY, USA, 2008 (Chapter 59).
[3] C.L. Nehaniv, K. Dautenhahn, The correspondence problem, in: K. Dautenhahn, C.L. Nehaniv (Eds.), Imitation in Animals and Artifacts, MIT Press, Cambridge, MA, USA, 2002 (Chapter 2).
[4] A. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, E. Liang, Inverted autonomous helicopter flight via reinforcement learning, in: International Symposium on Experimental Robotics, Singapore, 2004.
[5] P.K. Pook, D.H. Ballard, Recognizing teleoperated manipulations, in: Proceedings of the IEEE International Conference on Robotics and Automation, ICRA'93, Atlanta, Georgia, USA, 1993.
[6] J.D. Sweeney, R.A. Grupen, A model of shared grasp affordances from demonstration, in: Proceedings of the IEEE-RAS International Conference on Humanoids Robots, Humanoids'07, Tokyo, Japan, 2007.
[7] W.D. Smart, Making reinforcement learning work on real robots, Ph.D. Thesis, Department of Computer Science, Brown University, Providence, RI, 2002.
[8] R. Aler, O. Garcia, J.M. Valls, Correcting and improving imitation models of humans for robosoccer agents, Evolutionary Computation 3 (2–5) (2005) 2402–2409.
[9] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, The MIT Press, Cambridge, MA, London, England, 1998.
[10] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Springer, New York, NY, USA, 2001.
[11] P. Abbeel, A.Y. Ng, Exploration and apprenticeship learning in reinforcement learning, in: Proceedings of the 22nd International Conference on Machine Learning, ICML'05, 2005.
[12] M. Stolle, C.G. Atkeson, Knowledge transfer using local features, in: Proceedings of IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, ADPRL'07, Honolulu, Hawaii, USA, 2007.
[13] S. Chernova, M. Veloso, Multi-thresholded approach to demonstration selection for interactive robot learning, in: Proceedings of the 3rd ACM/IEEE International Conference on Human–Robot Interaction, HRI'08, Amsterdam, The Netherlands, 2008.
[14] D.H. Grollman, O.C. Jenkins, Dogged learning for robots, in: Proceedings of the IEEE International Conference on Robotics and Automation, ICRA'07, Rome, Italy, 2007.
[15] S. Calinon, A. Billard, Incremental learning of gestures by imitation in a humanoid robot, in: Proceedings of the 2nd ACM/IEEE International Conference on Human–Robot Interaction, HRI'07, Arlington, Virginia, USA, 2007.
[16] B.D. Argall, Learning mobile robot motion control from demonstration and corrective feedback, Ph.D. Thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 2009.
[17] S. Chernova, M. Veloso, Learning equivalent action choices from demonstration, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS'08, Nice, France, 2008.
[18] M.N. Nicolescu, M.J. Mataric, Methods for robot task learning: demonstrations, generalization and practice, in: Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems, AAMAS'03, Melbourne, Victoria, Australia, 2003.
[19] D.C. Bentivegna, Learning from observation using primitives, Ph.D. Thesis, College of Computing, Georgia Institute of Technology, Atlanta, GA, July 2004.
[20] J. Saunders, C.L. Nehaniv, K. Dautenhahn, Teaching robots by moulding behavior and scaffolding the environment, in: First Annual Conference on Human–Robot Interactions, HRI'06, Salt Lake City, Utah, USA, 2006.
[21] B. Argall, B. Browning, M. Veloso, Learning mobile robot motion control from demonstrated primitives and human feedback, in: Proceedings of the 14th International Symposium on Robotics Research, ISRR'09, Luzern, Switzerland, 2009.
[22] C.G. Atkeson, A.W. Moore, S. Schaal, Locally weighted learning for control, Artificial Intelligence Review 11 (1997) 75–113.
[23] H.G. Nguyen, J. Morrell, K. Mullens, A. Burmeister, S. Miles, K. Thomas, D.W. Gage, Segway robotic mobility platform, in: SPIE Mobile Robots XVII, Philadelphia, Pennsylvania, USA, 2004.
[24] B. Argall, Y. Gu, B. Browning, M. Veloso, The first segway soccer experience: towards peer-to-peer human–robot teams, in: First Annual Conference on Human–Robot Interactions, HRI'06, Salt Lake City, Utah, USA, 2006.
[25] R. Simmons, D. Apfelbaum, A task description language for robot control, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS'98, Victoria, Canada, 1998.

Brenna D. Argall was a Ph.D. candidate in the Robotics Institute at Carnegie Mellon University at the time of this work (degree granted in 2009). She received in 2006 an M.S. in Robotics and in 2002 a B.S. in Mathematics, both from Carnegie Mellon. Prior to graduate school, she held a Computational Biology position in the Laboratory of Brain and Cognition, at the National Institutes of Health, while investigating visualization techniques for neural fMRI data. She is now a postdoctoral fellow in the Learning Algorithms and Systems Laboratory (LASA) at the Swiss Federal Institute of Technology in Lausanne (EPFL). Her research interests focus upon machine learning techniques to develop and improve robot control systems, under the guidance of a human teacher.



Brett Browning is a Senior Systems Scientist in Carnegie Mellon University's School of Computer Science, where he has been a faculty member of the Robotics Institute since 2002. Prior to that, he was a postdoctoral fellow at Carnegie Mellon working with Manuela Veloso. Browning received his Ph.D. from the University of Queensland in 2000, and a B.Electrical Engineer and B.Sc. (Math.) from the same institution in 1996. His research interests are on robot autonomy and in particular real-time robot perception, applied machine learning, and teamwork.

Manuela M. Veloso is Herbert A. Simon Professor of Computer Science at Carnegie Mellon University. She received a licenciatura in Electrical Engineering in 1980, and an M.Sc. in Electrical and Computer Engineering in 1984 from the Instituto Superior Tecnico in Lisbon. She earned her Ph.D. in Computer Science from Carnegie Mellon in 1992. Veloso researches in planning, control learning, and execution algorithms, in particular for multirobot teams. With her students, Veloso has developed teams of robot soccer agents, which have been RoboCup world champions several times. She is a Fellow of AAAI, the Association for the Advancement of Artificial Intelligence, an IEEE Senior member, and the President Elect (2008) of the International RoboCup Federation.