Learning Mobile Robot Motion Control from Demonstrated Primitives and Human Feedback

Brenna Argall and Brett Browning and Manuela Veloso

Abstract Task demonstration is one effective technique for developing robot motion control policies. As tasks become more complex, however, demonstration can become more difficult. In this work we introduce a technique that uses corrective human feedback to build a policy able to perform an undemonstrated task from simpler policies learned from demonstration. Our algorithm first evaluates and corrects the execution of motion primitive policies learned from demonstration. The algorithm next corrects and enables the execution of a larger task built from these primitives. Within a simulated robot motion control domain, we validate that a policy for an undemonstrated task is successfully built from motion primitives learned from demonstration under our approach. We show feedback to both aid and *enable* policy development, improving policy performance in success, speed and efficiency.

1 Introduction

The appropriate selection of actions is a fundamental challenge within mobile robotics. The development of a robust *policy*, or mapping from world states to robot actions, is complicated by noisy observations and action execution uncertainty. Policy development furthermore is often specific to a particular robot platform and application, and policy reuse for other platforms or application tasks is rare.

One field of effective development approaches has the robot *learn* a policy, from training data or execution experience. Unlike traditional techniques that model world dynamics by hand, an implemented policy learning algorithm may be reused

Brenna Argall
Robotics Institute, Carnegie Mellon University, USA e-mail: bargall@ri.cmu.edu

Brett Browning
Robotics Institute, Carnegie Mellon University, USA e-mail: brettb@cs.cmu.edu

Manuela Veloso
Computer Science Department, Carnegie Mellon University, USA e-mail: mmv@cs.cmu.edu

to learn another policy, though the policy itself 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 on 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 being an accessible policy development technique for non-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 tasks, 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 approach given the challenge of developing robust control policies. In this work, we contribute an algorithm that builds a more complex policy from existing behaviors learned from demonstration.

How to effectively incorporate existing behaviors into a new policy is a key design decision for this work. We take the approach of aiding this process with human feedback offered in multiple forms, the most notable of which is continuous-valued corrections on student executions. Our framework for providing feedback does not require revisiting states in need of improvement, and thus offers an alternative to the more typical LfD policy improvement approaches that provide further teacher demonstrations. This feature is particularly attractive given our consideration of complex tasks for which full demonstration may be inconvenient or infeasible.

We introduce *Feedback for Policy Scaffolding (FPS)* as an algorithm that builds and refines a complex policy from component behaviors learned from demonstration and teacher feedback. The FPS algorithm operates by first refining multiple policies learned from demonstrated motion primitives. A single complex policy is derived from these primitives, and execution with the complex policy on a novel, undemonstrated behavior is then evaluated. By providing corrections on this execution, the FPS algorithm develops a policy able to execute the more complex behavior, without ever requiring a full demonstration of the novel behavior.

We validate our algorithm within a simulated motion control domain, where a robot learns to drive on a racetrack. A policy built from demonstrated motion primitives and human feedback is developed and able to successfully execute a more complex, undemonstrated task. Feedback is shown to improve policy performance when offered in response both to motion primitive executions as well as novel behavior executions, and moreover the policy developed under this technique is found to perform well within the novel domain. 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 more complex behavior.

The following section overviews the related work that supports this approach. In Section 3 the scaffolding algorithm is presented. Section 4 details the experimental implementation, including results and discussion. In the final section we conclude.

2 Background and Related Work

In this section we first present related work on policy development and improvement within demonstration learning, followed by the details of building policies from behavior primitives and teacher feedback.

We formally define the world to consist of states S and actions A, with a probabilistic transition function $T(s'|s,a) : S \times A \times S \rightarrow [0,1]$ describing the mapping between states by way of actions. As we do not assume that state is fully observable, the learner has access to observed state Z through the mapping $M : S \rightarrow Z$. A policy $\pi : Z \rightarrow A$ selects actions based on observations of the world state.

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. Formally, we represent a teacher demonstration $d_j \in D$ as t_j pairs of observations and actions such that $d_j = \{(\mathbf{z}_j^i, \mathbf{a}_j^i)\} \in D, \mathbf{z}_j^i \in Z, \mathbf{a}_j^i \in A, i = 0 \cdots t_j$. The set D of these demonstrations are provided to the learner.

When recording and executing demonstrations the issue of *correspondence*, where teacher demonstrations do not directly map to the robot learner due to differences in sensing or motion [9], is key. *Teleoperation* is a demonstration technique whereby the passive learner platform records from its own sensors while being controlled by the teacher during execution. Since the recorded data maps directly to the learner platform, this demonstration technique best minimizes the introduction of correspondence issues into an LfD system. Examples of successful teleoperated LfD systems include both real [10] and simulated [7] robot applications.

Policy derivation amounts to building a predictor that will reproduce the actions $\mathbf{a}^t \in D$ from the observations $\mathbf{z}^t \in D$. Many approaches exist within LfD to derive a policy from the demonstration data [3], the most popular of which either directly approximate the underlying function mapping observations to actions or approximate a state transition model and then derives a policy using techniques such as *Reinforcement Learning*. Our work derives a policy under the first approach, with function approximation being performed via regression since our target application of low-level motion control has a continuous action space.

A wealth of regression approaches exist, and any are compatible with the FPS algorithm. The specific technique used in our implementation is a form of Locally Weighted Learning [4]. In particular, given observation $\mathbf{z}^t$, action $\mathbf{a}^t$ is predicted through an averaging of datapoints in D, weighted by their kernelized distance to $\mathbf{z}^t$. While a more sophisticated regression technique would likely improve the performance of our implementation, the focus of this work is not how to better use *existing* demonstration data, but rather how to use teacher feedback to produce *new* data and thus refine policy performance and build new behavior.

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 [1] and reward-determined state values [13]. Other approaches provide more demonstration data, driven by learner requests for more data [7, 8] as well as more teacher-initiated demonstrations [6].

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. Our technique for policy improvement *synthesizes* new example state-action mappings from teacher feedback and learner executions [2], without requiring state re-visitation by the teacher to provide appropriate behavior information.

2.2 Behavior Primitives and Teacher Feedback

Our approach builds a policy from the demonstration of simpler behavior primitives and teacher feedback, rather than demonstrate a complex task in full. One motivation is that as behaviors become more complex, demonstrating the behavior in full can become more difficult. In this case, the teacher may be able to demonstrate behavior primitives for a task but not the task in full, or provide higher quality demonstrations for subsets of the behavior. A second motivation is that reaching all states encountered during task execution can become increasingly difficult as tasks become more complex. States may be infeasible, inconvenient or undesirable to reach for a variety of reasons that only compound as task complexity increases. A final motivation 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.

Within LfD, hand-coded behavior primitives are used to build larger tasks learned from demonstration [11], demonstrated tasks are decomposed into a library of primitives [5, 13] and behavior primitives are demonstrated and then explicitly combined into a new policy by a human [12]. Closest to our work is that of Bentivegna [5], where a robotic marble maze and humanoid playing air hockey reuse learned primitives, and furthermore refine the policy with execution experience. The work demonstrates the full task and then extractes behavior primitives using hand-written rules. Policy improvement is accomplished through an automatic binary reward signal for task failure, used to adjust regression weights on the policy prediction. By contrast, our approach does not demonstrate the full task, and instead demonstrates the primitives individually. Policy improvement is accomplished by generating new examples, from human feedback on practice executions of the primitives and full task.

3 Algorithm

This section presents our *Feedback for Policy Scaffolding (FPS)* algorithm. Under FPS, teacher feedback is used to enable and improve policy behavior at transition points that link demonstrated primitives. In doing so, it enables expression of the full task behavior, without requiring its full demonstration.

Feedback is provided through the framework *Focused Feedback for Mobile Robot Policies (F3MRP)* [3]. The F3MRP framework operates at the stage of low-level motion control, where actions are continuous-valued and sampled at high frequency. A visual presentation of the 2-D ground path of the mobile robot execution serves as an interface through which the teacher selects segments of an execution to receive feedback, which simplifies the challenge of providing feedback to policies sampled at a high frequency. Visual indications of data support during an execution futhermore assist the teacher in the selection of execution segments and feedback type.

Execution *corrections* are offered through a language termed *advice-operators*, first introduced with the *Advice-Operator Policy Improvement (A-OPI)* algorithm [2]. Advice-operators are commonly defined between the student and teacher, and function by performing mathematical computations on the observations or actions of executed data points. In this manner, they provide continuous-valued corrections on a learner execution, without requiring the teacher to provide the exact *value* for the corrections. Instead, the teacher need only select from a finite list of operators, and indicate the portion of the execution requiring improvement.

3.1 Algorithm Execution

Execution of the FPS algorithm occurs in two phases, presented respectively in Algorithms 1 and 2. The first phase develops a policy π_{ξ_j} for each primitive behavior $\xi_j \in \Xi, j = 1 \ldots n$, producing a set of policies Π. The second phase develops a policy for a more complex behavior, building on the primitive policies in Π.

3.1.1 Phase 1: Primitive Policy Development

The development of each primitive policy begins with teacher demonstration. Example observation-action pairs recorded during demonstration of primitive behavior ξ_j produce the initial dataset $D_{\xi_j} \in \mathbf{D}$ and policy $\pi_{\xi_j} \in \Pi$. This initial policy is then refined during practice runs consisting of learner executions and teacher feedback.

During the learner execution portion of a practice run (Alg. 1, lines 7-11), the learner first executes the task. At each timestep the learner observes the world, predicting action $\mathbf{a}^t$ according to policy π_{ξ_j} (line 8). Action $\mathbf{a}^t$ is executed and recorded in the prediction trace d, with observation $\mathbf{z}^t$ (line 10). The information recorded in the trace d will be incorporated into the policy update. The global position x^t, y^t

Algorithm 1 *Feedback for Policy Scaffolding: Phase 1*

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

and heading θ^t of the mobile robot, and data support τ^t (discussed in Section 3.2.2) of the regression prediction, are recorded into the execution trace tr, for use by the F3MRP framework when visually presenting the ground path taken by the robot.

During the teacher feedback portion of the practice run, the teacher first indicates a segment Φ of the learner execution trace requiring improvement (line 12). The teacher further indicates feedback F, which takes the form either of a binary credit to indicate areas of good performance, or an advice-operator to correct the execution within this segment. The application of F across all points recorded in d and within the indicated subset Φ generates new data, $\hat{d}_\Phi$, which is added to dataset D_{ξ_j} (lines 13,14). Policy π_{ξ_j} for primitive ξ_j is then rederived from this set (line 15).

3.1.2 Phase 2: Policy Scaffolding

The development of the complex policy builds on the primitive policies developed during Phase 1 of the algorithm. Complex policy development therefore does *not* begin with teacher demonstration of the complex task. Two distinguishing features of the second phase of the FPS algorithm are the (i) selection between the action predictions of multiple policies and (ii) selection of a dataset to receive any new synthesized data. Figure 1 presents a schematic of our scaffolding approach, where dashed lines indicate execution cycles that are performed multiple times.

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 data set $D_{\xi_{i+n}}$ be associated with existing primitive dataset D_{ξ_i}, resulting in a total of $2n$ datasets $D_{\xi_j} \in \mathbf{D}$, $j = 1 \cdots 2n$. Colloquially, call dataset $D_{\xi_{i+n}}$

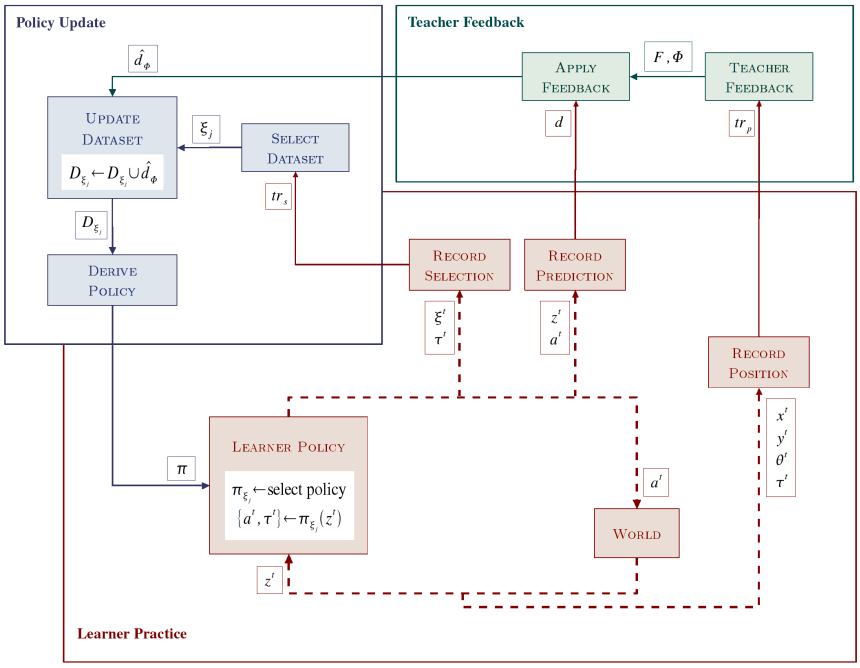

Fig. 1 Policy derivation and execution under the Feedback for Policy Scaffolding algorithm.

the *feedback dataset* associated with primitive dataset D_{ξ_i}. Some of the new data generated during learner practice will be added to these feedback datasets (further details are provided in Sec. 3.2.2). The policies derived from the feedback datasets are considered, along with the primitive policies, for selection during execution of the more complex policy.

Refinement of the complex policy proceeds with learner execution (Alg. 2, lines 5-10) and teacher feedback (lines 11-17) 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. The first step is to select between all contributing policies π_{ξ_j} based on observation $\mathbf{z}^t$ (line 6); the details of this selection are provided in Section 3.2.1. The second step is to predict action $\mathbf{a}^t$ according to $\pi_{\xi_j}(\mathbf{z}^t)$, with prediction support τ^t (line 7). After the application of teacher feedback, datasets are individually selected to receive each feedback-modified datapoint. For each point, indexed as $\varphi \in \Phi$, dataset selection (line 14) is determined by the data support τ^φ of $\mathbf{z}^\varphi$ when predicted by policy ξ^φ (recorded in tr_s, line 9); the details of this selection are provided in Section 3.2.2.

Algorithm 2 *Feedback for Policy Scaffolding: Phase 2*

Given $\Pi, \mathbf{D}$
initialize $D_{\xi_i=(n+1)\ldots 2n} \leftarrow \{\,\}$
while *practicing* **do**
  initialize $d \leftarrow \{\}$, $tr_s \leftarrow \{\}$, $tr_p \leftarrow \{\}$
  repeat
    select $\pi_{\xi_j} \leftarrow$ `policySelection`$(\,\mathbf{z}^t\,)$, $\pi_{\xi_j} \in \Pi$
    predict $\{\,\mathbf{a}^t, \tau^t\,\} \leftarrow \pi_{\xi_j}(\mathbf{z}^t)$
    execute $\mathbf{a}^t$
    record $d \leftarrow d \cup (\mathbf{z}^t, \mathbf{a}^t)$, $tr_s \leftarrow tr_s \cup (\tau^t, \xi^t = \xi_j)$, $tr_p \leftarrow tr_p \cup (x^t, y^t, \theta^t, \tau^t)$
  until done
  advise $\{\,F, \Phi\,\} \leftarrow$ `teacherFeedback`$(\,tr_p\,)$
  for all $\varphi \in \Phi$, $(\mathbf{z}^\varphi, \mathbf{a}^\varphi) \in d, (\tau^\varphi, \xi^\varphi) \in tr_s$ **do**
    apply $\hat{d}_\varphi \leftarrow$ `applyFeedback`$(\,F, \mathbf{z}^\varphi, \mathbf{a}^\varphi\,)$
    select $D_{\xi_t} \leftarrow$ `datasetSelection`$(\,\tau^\varphi, \xi^\varphi\,)$, $D_{\xi_t} \in \mathbf{D}$
    update $D_{\xi_t} \leftarrow D_{\xi_t} \cup \hat{d}_\varphi$
    rederive $\pi_{\xi_t} \leftarrow$ `policyDerivation`(D_{ξ_t}), $\pi_{\xi_t} \in \Pi$
  end for
end while
return Π

3.2 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 algorithm is discussed here.

3.2.1 Selecting Primitive Policies

Primitive selection under FPS assumes that primitives occupy nominally distinct areas of the observation-space. This assumption relies on a state observation formulation that captures aspects of the world that are unique to the demonstrations of each primitive policy. For example, two primitives developed for our validation domain are *turn left* and *turn right*. Observations are formulated to incorporate a notion of track curvature, and so demonstrations in left- versus right-curving areas of the track occupy distinct areas of the observation space.

Primitive selection then is treated as a classification problem. For each primitive ξ_j, a kernelized distance $\phi(\mathbf{z}^t, \mathbf{z}_i)$ between query point $\mathbf{z}^t$ and each point $\mathbf{z}_i \in D_{\xi_j}$ is computed.[1] A weight for policy ξ_j is produced by summing the k largest kernel values $\phi(\mathbf{z}^t, :)$; equivalent to selecting the k nearest points in D_{ξ_j} to query $\mathbf{z}^t$ ($k = 5$). The policy with the highest weight is then selected for execution.

[1] In our implementation the distance computation is Euclidean and the kernel Gaussian, and so $\phi(\mathbf{z}^t, \mathbf{z}_i) = e^{|\mathbf{z}_i - \mathbf{z}^t| \Sigma^{-1} |\mathbf{z}_i - \mathbf{z}^t|}$. The parameter Σ^{-1} is a constant diagonal matrix that scales each observation dimension and embeds the bandwidth of the Gaussian kernel, and is tuned through 10-folds Cross Validation to optimize the least-squared-error on primitive label classification.

3.2.2 Incorporating Teacher Feedback

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

Two factors determine into which dataset a new datapoint $\hat{d}_\varphi$ is added: the policy ξ^φ that predicted the execution point $(\mathbf{z}^\varphi, \mathbf{a}^\varphi)$, and the measure of data support τ^φ for that prediction. In particular, if the policy that made the prediction is a primitive policy, the point is added to its dataset *if* the prediction had strong data support. Otherwise, the point is added to the *feedback* dataset associated with primitive ξ^φ (Sec. 3.1.2). By contrast, data generated from execution points predicted by a feedback policy are automatically added to its dataset, regardless of dataset support.

Prediction support is determined in the following manner. For a given dataset, the 1-Nearest Neighbor Euclidean distance between all points in the set are modelled as a Poisson distribution, parameterized by λ, with mean $\mu = \lambda$ and standard deviation $\sigma = \sqrt{\lambda}$. The threshold on strong prediction support is set by hand, based on empirical evidence ($\tau_{\xi_j} = \mu_{\xi_j} + 5\sigma_{\xi_j}$). Thus a prediction made by policy π_{ξ_j} for query point $\mathbf{z}^t$ with distance $\ell_{\mathbf{z}^t}$ to the nearest point in D_{ξ_j} is classified as strongly supported if $\ell_{\mathbf{z}^t} < \tau_{\xi_j}$ and weakly supported otherwise.

The motivation behind this approach is to avoid adding data to a primitive dataset that conflicts with the behavior of that primitive. Given our assumption that the associated actions of nearby observations express similar behaviors, points that were close enough to the dataset to be strongly supported during prediction therefore are assumed to express behavior similar to that of the primitive.

4 Empirical Implementation

This section presents the experimental details, results and discussion of the application of algorithm FPS to a simulated motion control domain.

4.1 Experimental Setup

The validation domain consists of a simulated differential drive robot performing a racetrack driving task. Robot motion is propagated by simple differential drive sim-

ulation of the robot position (1% Gaussian noise), limited in speed and acceleration. The robot observes the world through a monocular camera and wheel encoders; the camera is forward facing and observes track borders (1% Gaussian noise) within its field of view $(130^{\circ}, 5m)$ as a set of points, each of which corresponds to a single image pixel projected into the ground plane. The robot computes a local track representation at each time step $(30Hz)$ by fitting a 3-degree polynomial to recently observed track border points. Policy observations are 6-dimensional: current rotational and translational speeds, and the 4 coefficients of the local track polynomial. The actions are 2-dimensional: target rotational and translational speeds.

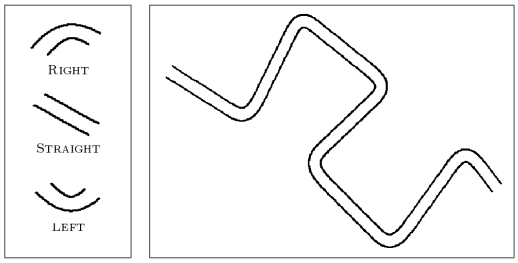

Fig. 2 Primitive subset regions (left) of the full racetrack (right).

The demonstrated motion primitives are: *turn right* (ξ_R), *go straight* (ξ_S) and *turn left* (ξ_L). Demonstrations are performed via human teleoperation, by decreasing or increasing the translational and rotational speeds as the robot moves along the racetrack. The robot has no a priori map of the track, nor does it attempt to build up a map during execution; the aim of the developed policy is to reactively drive on a racetrack. The following steps are taken during policy development:

Demonstrate the motion primitives and derive initial policies. Teacher demonstration of each primitive is performed 3 times on an appropriate track subset (Fig. 2, left). From each dataset a policy is derived, referred to collectively as the set PD_I.

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, and 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.

Derive an initial scaffolded policy from the resultant primitive policies. An initial scaffolded policy SF_I, that selects between the primitive policies in PF_F, is built.

Provide feedback on the scaffolded policy 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 per Sec. 3.2.2). 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 providing more demonstrations. The approach closely follows the policy development steps just outlined, but the teacher provides more teleoperation demonstrations instead of feedback. The result is *demonstration+* versions of a final set of primitives policies (PD_F), initial baseline scaffolded policy (SD_I) and final scaffolded policy (SD_F).

Each of the primitive policies (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 end when the robot either runs off the track or reaches the finish line.

Policy performance is measured according to the following metrics. *Success* indicates the ability of the robot to stay on the track, and is measured by the percentage of the track subset (for primitive policy executions) or full track (for scaffolded policy executions) completed. *Speed* is measured by the average translational execution speed. *Efficiency* is measured as the execution time, and is governed jointly by speed and the execution ground path.[2]

4.2 Results

The FPS algorithm successfully learned motion control primitives through a combination of demonstration and teacher feedback, as well as a policy built from these primitives to execute a more complex, *undemonstrated*, behavior. Teacher feedback was found to be critical to the development and performance improvement of all policies, which far outperformed those that received more teacher demonstrations.

4.2.1 Motion Primitives Learned from Demonstration

The three motion primitives were successfully learned in the first phase of the FPS algorithm (Tbl. 1, average of 50 executions, 1-standard deviation).[3]

The initial policies in PD_I were unable to complete either of the *turn right* or *turn left* behaviors. The initial *go straight* primitive behavior was able to complete the task, however execution proceeded extremely slowly.

All policies resulting after Phase 1 development of the FPS algorithm (in PF_F) were able to complete their respective primitive behaviors. Furthermore, executions with these policies were much faster on average than those in PD_I, as summarized in Figure 3 (green bars; also in Tbl. 1). Of particular note is the *go straight* primitive policy, whose *average* speed over the executions approaches the maximum speed of the robot ($3.0\frac{m}{s}$), all without compromising the success of the executions. Aggressive speeds at times negatively impacted the *turn left* policy however, whose more efficient executions came at the cost of occasional incomplete executions.

[2] Efficiency is computed only for successful executions, that by definition do not abort early.

[3] The figures and tables of this section label the primitive policy sets intuitively: *Baseline* refers to PD_I, *Feedback* refers to PF_F and *More Demonstration (More Demo)* refers to PD_F.

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

Table 1 Execution performance of the primitive policies.

In contrast to the FPS policy, the *turn right* policy resulting from more teleoperation demonstrations (in PD_F) was not able to complete the task, or even to improve upon the performance or speed of the initial policy. Furthermore, the policy was developed with significantly more practice runs and training data (36 *vs*. 23 practice runs, $2,846$ *vs*. 561 new datapoints). The *go straight* demonstration policy (Fig. 3, blue bar) was able to improve execution speed over the baseline policy, but not as dramatically as the FPS policy, and again with more training data and practice runs (36 *vs*. 27 practice runs, $1,630$ *vs*. 426 new datapoints). The *turn left* policy actually *decreased* the performance of the initial policy, both in success and speed (and with 12 *vs*. 8 practice runs, 965 *vs*. 252 new datapoints). More demonstrations in this case likely created ambiguous areas for the policy, a complication that would perhaps clear up with the presentation of more disambiguating demonstrations.

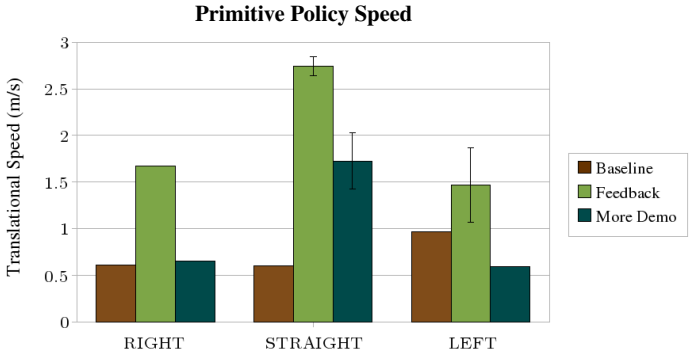

Fig. 3 Average translational execution speed with each of the primitive behavior policies.

4.2.2 Undemonstrated Task Learned from Primitives and Feedback

A policy able to execute a more complex, *undemonstrated*, behavior was successfully developed through the scaffolding of the learned primitive policies, plus the incorporation of teacher feedback. 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 execute this task in full. Performance improvement over 160 practice runs is presented in Figure 4 (left). Each practice run produces a new iterative policy. Each plot point represents an average of 10 track executions with a given iterative policy, and a regularly sampled subset of the iterative policies were evaluated in this manner (sampled every 10 policies, 17 iterative policies evaluated in total). This constitutes Phase 2 of the FPS algorithm, after which the learner was able to consistently execute the complex task in full.

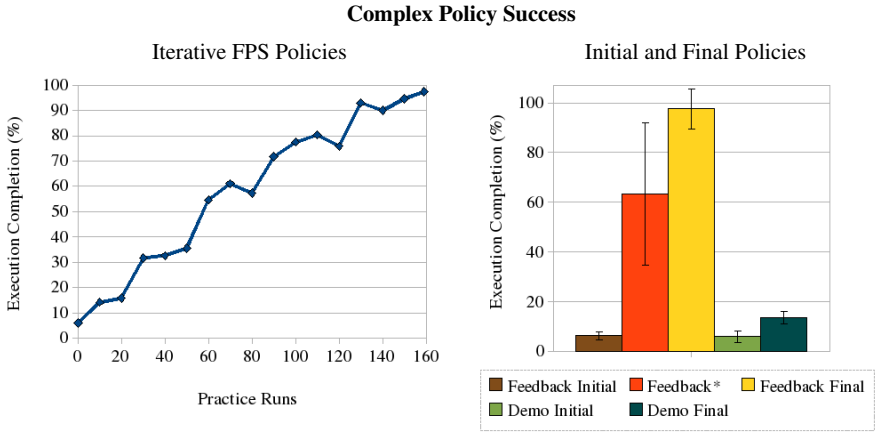

Fig. 4 Percent task completion during (left) and after (right) complex policy practice.

4.2.3 Improvement in Complex Task Performance

Beyond the development of a policy *able* to perform the more complex task, FPS furthermore enabled performance *improvement* such that executions became faster and more efficient (Tbl. 2, average of 50 executions, 1-standard deviation error bars).

Figure 4 (right) summarizes the percent completed execution of multiple policies on the full track task (also in Tbl. 2). The final policy SF_F that resulted after Phase 2 of the FPS algorithm was able to consistently execute the task successfully (*Feedback Final*); as noted above, the initial scaffolded FPS policy SF_I was not able to complete this task (*Feedback Initial*). By contrast, the policy SD_F that resulted from more teleoperation demonstrations (*Demo Final*) was not able to complete the task, though it did improve upon the performance its initial policy (*Demo Initial*).

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

Table 2 Execution performance of the scaffolded policies.

The final FPS policy SF_F however was more extensively developed than the demonstration policy SD_F, whose extremely slow rate of policy improvement prompted the teacher to abort policy development (159 *vs*. 74 practice runs). The above comparison between the final FPS and demonstration policies therefore is not a fair one, and so the results from an *iterative* FPS policy are also provided (*Feedback**). This policy is not the final FPS policy, but rather the result of development after only 74 practice runs, the same number of practice runs as the final demonstration policy. These runs produced $2,448$ new datapoints; far fewer than the 74 runs of the demonstration policy, which produced $8,520$ new points. Even so, against this iterative policy the final demonstration policy also did not measure well. The iterative policy (*Feedback**) significantly outperformed the final demonstration policy (*Demo Final*) on the success measure, though it does not yet perform as successfully or as consistently as the final FPS policy (*Feedback Final*).

The speed performance results closely resemble those of success performance (Tbl. 2). Namely, the final FPS policy 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 its initial policy (*Demo Initial*), but not nearly as much as the iterative FPS policy provided for comparison (*Feedback**). Interesting to note is that the iterative FPS policy (*Feedback**) produced similar speeds to the final FPS policy, but with larger standard deviations (Tbl. 2), suggesting that performance *consistency*, in addition to execution success, also motivated the teacher to continue development beyond this iterative policy.

4.3 Discussion

This section highlights some noteworthy gains provided by the FPS algorithm, including policy reuse and more focused datasets.

4.3.1 Reuse of Primitives Learned from Demonstration

These empirical results confirm that FPS was able to successfully build a policy for an undemonstrated task, from existing primitive policies learned from demonstration. We identify two crucial gains to such an approach.

The first gain is that the multiple motion primitive policies were developed from *demonstration*. Demonstration has many attractive features as a medium for knowledge transfer from human teacher to robot learner [3]. Moreover, this demonstration technique was aided with teacher *feedback*, provided under the F3MRP framework. Without this feedback, the learned primitive policies are less successful, less efficient, slower, and in some cases even unable to complete the target behavior. This is true not just of the initial primitive policies derived from demonstration, but also of the policies provided with more demonstrations in response to learner execution performance. The FPS algorithm therefore provides a more efficient and effective LfD technique for the development of these motion primitive policies.

Even with the advantages secured through demonstration and teacher feedback however, policy development typically is still a non-trivial task. The second gain of the FPS approach therefore is the ability to *reuse* the primitives within another policy. The full track task was shown to be sufficiently complex that the improvements afforded by demonstrations of the full task behavior were significantly smaller than those gained through teacher feedback. Moreover, this performance difference between the feedback and more-demonstration techniques was much larger for the complex task than for the simpler primitive policies. These results suggest that the complex task cannot be learned through demonstration exclusively, unless perhaps provided with a very large quantity of demonstration data, again underlining the advantage of simple policy reuse within this complex domain.

4.3.2 More Focused Datasets

One result from the experimental validation of A-OPI in [2] was the development of much smaller datasets with corrective feedback in comparison to more demonstration. The smaller datasets furthermore produced similar or superior performance, prompting the conclusion that the datasets were less redundant and more focused.

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

Moreover, these teleoperation policies never perform similarly to their FPS counterparts and, in contrast to the A-OPI results, instead usually display significantly *inferior* performance. This observation suggests that not only is the added data less redundant, but furthermore includes *relevant* data that is *not* being produced by the demonstration. One issue is that to provide a correction through demonstration can be difficult with motion control tasks, and this difficulty scales with task complexity. Further detrimental is the reality that the teacher often must reproduce suboptimal behavior in order to *reach* the state intended to receive a corrective demonstration. We propose that the value in policy refinement alternatives to state revisitation only grows as tasks and domains become more complex.

5 Conclusions

We have introduced *Feedback for Policy Scaffolding (FPS)* as an algorithm that builds, or scaffolds, a policy from demonstrated component behaviors and corrective human feedback. The complete behavior of the scaffolded policy itself need not be demonstrated. We have validated the FPS algorithm within a simulated robot motion control domain. A policy built from demonstrated motion primitives and human feedback was able to execute a more complex, undemonstrated task, thus confirming successful policy reuse. Moreover, we found that successful execution of the complex behavior was in fact *enabled* by teacher feedback. When compared to providing more teacher demonstrations, FPS was shown to produce better performing policies, from more focused datasets. Policy performance was found to *improve* with feedback, in the measures of success, speed and efficiency, and for the complex as well as primitive behaviors.

Acknowledgements This 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, and the Qatar Foundation for Education, Science and Community Development. The views and conclusions in this document are solely those of the authors.

References

1. P. Abbeel and A. Y. Ng. Exploration and apprenticeship learning in reinforcement learning. In *Proceedings of ICML*, 2005.
2. B. Argall, B. Browning, and M. Veloso. Learning robot motion control with demonstration and advice-operators. In *Proceedings of IROS*, 2008.
3. B. D. Argall. *Learning Mobile Robot Motion Control from Demonstration and Corrective Feedback*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 2009.
4. C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning for control. *Artificial Intelligence Review*, 11:75–113, 1997.
5. D. C. Bentivegna. *Learning from Observation Using Primitives*. PhD thesis, College of Computing, Georgia Institute of Technology, Atlanta, GA, July 2004.
6. S. Calinon and A. Billard. Incremental learning of gestures by imitation in a humanoid robot. In *Proceedings of HRI*, 2007.
7. S. Chernova and M. Veloso. Multi-thresholded approach to demonstration selection for interactive robot learning. In *Proceedings of HRI*, 2008.
8. D. H. Grollman and O. C. Jenkins. Dogged learning for robots. In *Proceedings of ICRA*, 2007.
9. C. L. Nehaniv and K. Dautenhahn. The correspondence problem. chapter 2. MIT Press, Cambridge, MA, USA, 2002.
10. A. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang. Inverted autonomous helicopter flight via reinforcement learning. In *International Symposium on Experimental Robotics*, 2004.
11. M. N. Nicolescu and M. J. Mataric. Methods for robot task learning: Demonstrations, generalization and practice. In *Proceedings of AAMAS*, 2003.
12. J. Saunders, C. L. Nehaniv, and K. Dautenhahn. Teaching robots by moulding behavior and scaffolding the environment. In *Proceedings of HRI*, 2006.
13. M. Stolle and C. G. Atkeson. Knowledge transfer using local features. In *Proceedings of ADPRL*, 2007.