Avoiding Local Optima with User Demonstrations and Low-level Control

Shane Celis
Dept. of Computer Science
University of Vermont
Email: shane.celis@uvm.edu

Gregory S. Hornby
UARC, U.C. Santa Cruz
NASA Ames Research Center
Email: gregory.s.hornby@nasa.gov

Josh Bongard
Dept. of Computer Science
University of Vermont
Email: josh.bongard@uvm.edu

***Abstract*—Interactive Evolutionary Algorithms (IEAs) use human input to help drive a search process. Traditionally, IEAs allow the user to exhibit preferences among some set of individuals. Here we present a system in which the user directly demonstrates what he or she prefers. Demonstration has an advantage over preferences because the user can provide the system with a solution that would never have been presented to a user who can only provide preferences. However, demonstration exacerbates the *user fatigue* problem because it is more taxing than exhibiting preferences. The system compensates for this by retaining and reusing the user demonstration, similar in spirit to user modeling. The system is exercised on a robot locomotion and obstacle avoidance task that has an obvious local optimum. The user demonstration is provided through low-level control. The system is compared against a high-level fitness function that is susceptible to becoming trapped by a local optimum and a mid-level fitness function designed to remove the local optimum. We show that our proposed system outperforms most variants of these completely automatic methods, providing further evidence that Evolutionary Robotics (ER) can benefit by combining the intuitions of inexpert human users with the search capabilities of computers.**

I. Introduction

Citizen science, wherein inexpert users contribute to scientific research, has demonstrated that non-expert users can assist researchers on tasks that appear to require a great deal of training, such as Foldit, the crowd-sourced protein-folding game [12]. Interactive evolutionary algorithms seem like a natural means of integrating user input and computational algorithms for certain problem domains [7, 10, 19, 20] because input from a user warps and, if done correctly, can smooth the fitness landscape for the evolutionary algorithm. However, it is not yet clear how, when, and how much the user should influence the evolutionary algorithm. How should one incorporate user input? Should an algorithm present complete solutions that the user ranks according to preference? Should an algorithm present solutions that the user may directly manipulate? This paper explores the latter idea whereby the user may take a solution and directly manipulate it to suit his or her preference. Some similar problems crop up in approaches that require the user to indicate preferences between solutions and approaches that require the user to provide a demonstration, so it is worth surveying work on user preferences.

Traditional IEAs rely on the user to effectively become the fitness function of an EA. The user is presented with a set of individuals. The user chooses one or more individuals according to their preference. The EA then selects the favored individuals to generate the next generation. This can be especially effective in domains that rely on human perception [9, 10, 18, 19].

However, driving search by user input does have a number of issues. One limitation is that the fitness function is costly, degrades over time, and is imprecise. The degradation over time is known as *user fatigue* [20]. Typical non-interactive evolutionary algorithms require thousands of fitness function evaluations to find useful solutions. This is particularly problematic if the user is required to demonstrate preferred solutions rather than simply judge the relative merits of automatically-generated solutions.

One thread of research seeking to alleviate the *user fatigue* problem is user modeling, originally presented in the evolutionary algorithms domain by Schmidt and Lipson [17], inspired by the Estimation-Exploration Algorithm [4, 6]. The basic scheme is to collect the user responses and automatically build a user model that may be used with impunity. Subsequent work by Hornby and Bongard has shown that user modeling can produce solutions of better quality to those produced by a traditional IEA and with far fewer user interactions [13, 14].

Although the user input differs in this paper from the work on user preferences in that it allows the user to demonstrate what they prefer, it needs to overcome the *user fatigue* problem to even be feasible. It must have a means of retaining and reusing the user demonstrations similar in spirit to the way user modeling retains, reuses, and learns the user preferences.

Demonstration is meant broadly, i.e., if the domain involves creating images as in Picbreeder [18], the user might demonstrate their preference by painting a portion of an image red, signaling to the search algorithm that it should favor images with red coloring. If the domain involves creating 3D shapes as in Endless Forms [9], the user might demonstrate by extruding or molding the form. If the problem is a robotics task, the user might demonstrate by moving the joints of the robot directly, an idea pursued in this paper. Precursors to this idea are known as behavioral cloning [2], programming by demonstration [8], also known as learning by imitation.

This paper uses a robotics task to evaluate the proposed system: a robot must locomote around a barrier to reach a target (see Fig. 1). This task was chosen in part to compare with concurrent work on user preferences [5]. In this paper we consider three levels of control: high-level, mid-level, and low-level. Consider the quadruped robot attempting the task as shown in Fig. 2. A high-level controller might command

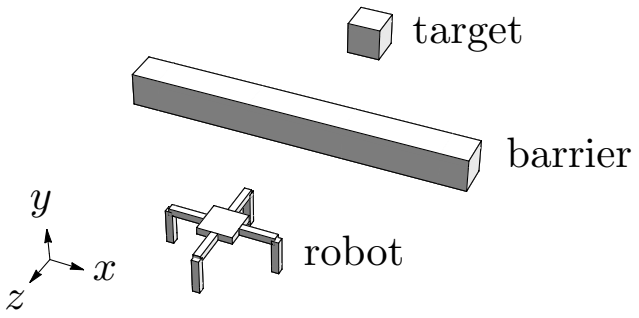

Figure 1. A quadruped robot is exercised on a task to locomote and approach the target. To successfully do this, it must navigate around the barrier.

the robot to go to the target location; it need not specify that there is, or how to avoid, an obstacle. A mid-level controller might command the robot to move rightward ($+x$), presumably to avoid the barrier, then go forward ($-z$), then go leftward ($-x$) towards the target; thus resembling commands a human might provide using a remote control. A low-level controller commands the angle of each joint directly; it is likely that simultaneous dictating all joint angles would be troublesome for a human controller. By analogy with high-level control, we consider a high-level fitness function to be one that captures the high-level objective: reach the target without trying to specify details like avoid obstacles. And like mid-level control, we consider a mid-level fitness function to be one that decomposes the problem somewhat, e.g., go toward this position, then go toward that object.

In accepting user input, it behooves us to consider at what level of detail we will accept that input, at a high-level, mid-level, or low-level? Traditionally, the dream of Evolutionary Robotics (ER) has concentrated on automatically producing robot controllers without any human input, but this usually translated into researchers iteratively producing mid- or high-level fitness functions, which in some cases seems like a herculean effort [21]. In contrast, this system accepts low-level inexpert user input coupled with a high-level objective and a means of retaining and reusing that input. The aim is to test the counter intuitive notion that human intuition, exercised through low-level control, may help steer search away from local optima.

Although this paper uses low-level control for user demonstration, that should not be regarded as its essential feature. A user demonstration could use mid-level control, e.g., a user might describe the path they wish for the robot to follow. A user demonstration could also use high-level control. For example, in work by Bongard, Beliveau, and Hornby a user may define the fitness function by placing the robot next to a target such that its target sensors are some small value [3]. The fitness function will select for robots that come closest to achieving those target sensor values. Although Bongard et al. do not couch this method in terms of user demonstration or high-, mid-, or low-level control, it is a good example of a user demonstration that uses high-level control.

Why focus on avoiding local optima? Because these are the places where common sense, high-level thinking and fitness functions get stuck. Other approaches to avoiding local optima are worth mentioning. Novelty search works well avoiding local optima on constrained behavior spaces with deceptive tasks [16]. Work by Karpov on human-assisted neuro-evolution through examples closely matches the user demonstration line of thought pursued in this paper [15]. The main contribution of this paper is to show that low-level user demonstration may help avoid local optima in ER.

II. Methods

This paper compares three distinct fitness functions with a couple of variants on a robot locomotion and obstacle avoidance task. This section will describe the task, the robot, the fitness functions, and the user surrogate. There are three principle fitness functions: High-level, mid-level, and our proposed hybrid IEA that combines a high-level fitness function with a low-level user demonstration. No low-level fitness function is defined but low-level control is discussed. The objective is to demonstrate that the hybrid IEA can allow an inexpert user to avoid the local optima by partly demonstrating how the robot should behave. To assess the performance of this system, a surrogate is used in place of the user.

A. Task

The task chosen to exercise these methods is a robot obstacle avoidance task as shown in Fig. 1. A barrier sits between a robot and a target object. The target object emits light which cannot be seen from behind the barrier. The robot succeeds at this task if it can get within 4.5 units of the target within 30 simulated seconds, which requires it to move around the barrier. The local optima depends on the fitness function; however, this task was selected specifically because it has an obvious local optima for the high-level fitness function.

B. Robot

A simulated quadruped robot has eight degrees of freedom denoted $\{\theta_1, \theta_2, \ldots, \theta_8\}$. The joint range is constrained to $[-\pi/4, \pi/4]$. The hinge joints are position controlled. The robot has two distance sensors s_1 and s_2 located at positions shown in Fig. 2. The sensors measure the distance between their position and the target object's position. However, the sensors only work if within the line-of-sight of the target. Behind the obstacle, the sensors report maximum distance.

The controller is a feed-forward artificial Neural Network (NN) with a sigmoidal activation function. It accepts four inputs and produces eight outputs. The first two inputs are measures of time. A fast time measure, intended for controlling the gait of the robot, oscillates between -1 and 1 every simulated second. A slow time measure oscillates between -1 and 1 over the entire evaluation duration, 30 simulated seconds. The last two inputs are sensors s_1 and s_2. The NN updates the desired angle θ_i ten times a simulated second. The NN has two hidden layers of twelve neurons each. Twelve was the lowest number of neurons which adequately captured the user demonstrated gait.

C. High-Level Control and Fitness

The task objective is to reach the target object. The most direct and naive way to write a fitness function is to minimize

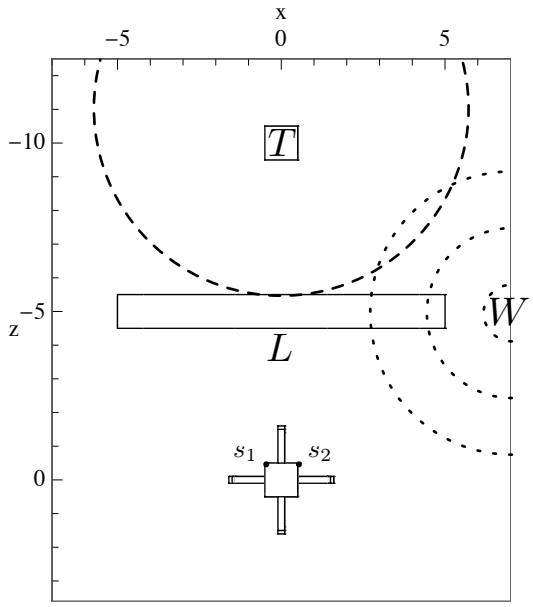

Figure 2. This top-down view shows the robot and task environment, which includes a barrier between the robot and the target. The target T is the global optimum, shown with a dashed bounding radius which is the threshold for success. The local optima for f_{high} is located at L. The waypoint W is shown with a series of successively larger radii. The radii are 10%, 30%, and 50% of the distance between the robot and waypoint, which relate to the parameter α with the value 0.1, 0.3, and 0.5 respectively. The robot has two sensors s_1 and s_2 located on the top-left and top-right side of its root body segment.

the distance between the robot and the target.[1] Let $\mathbf{r}_r(t_f)$ and $\mathbf{r}_t$ denote the position of the robot at the final time t_f and the target respectively.

$$f_{\text{high}} = ||\mathbf{r}_r(t_f) - \mathbf{r}_t|| \tag{1}$$

Many straight forward fitness functions like this have local optima that may trap a population in an evolutionary algorithm. Often this requires one define a fitness function in an iterative fashion, adding more terms to penalize against these local optima. For a greedy optimization method, this task has an obvious local optima (labeled L on Fig. 2). If the robot heads directly towards the barrier and stops, it can do no better on that side of the barrier. All small steps away from L offer worse fitness.

D. Mid-level Control and Fitness

Rather than trying to determine the appropriate high-level fitness function, the mid-level fitness function attempts to guide the solution. For a mobility task like the one presented in this paper, a natural solution is to use a waypoint, whereby the robot is encouraged to go to a waypoint before attempting to go towards the target. Essentially this parameterizes and composes the high-level fitness functions in sequence. Let $\mathbf{r}_r(t)$ denote the position of the robot at time t. Let $\mathbf{r}_w$, $\mathbf{r}_t$ denote the position of the waypoint and target respectively. The initial time is t_0. The robot reaches the waypoint at time t_1.

[1]All fitness functions in this paper are defined to be minimized—not maximized.

$$f_1(t) = \frac{||\mathbf{r}_r(t) - \mathbf{r}_w||}{||\mathbf{r}_r(t_0) - \mathbf{r}_w||} \tag{2}$$

$$f_2(t) = \frac{||\mathbf{r}_r(t) - \mathbf{r}_t||}{||\mathbf{r}_r(t_1) - \mathbf{r}_t||} \tag{3}$$

$$t_1 = \min_t f_1(t) < \alpha \tag{4}$$

$$f_{\text{mid}} = \frac{1}{t_f} \sum_{t=0}^{t_f} \begin{cases} f_1(t) & t < t_1 \\ \alpha f_2(t) & \text{otherwise} \end{cases} \tag{5}$$

The parameter α defines the waypoint radius, determining how close the robot must approach to reach the waypoint. The fitness can be broken into two parts: 1) While the time is between t_0 and t_1, the fitness is the average normalized distance to the waypoint. 2) While the time is between t_1 and the final time t_f, the fitness is the average normalized distance to the target scaled by the parameter α. The reason for this scaling is to avoid a local optima where a robot may more profitably come close to a waypoint but not cross its threshold.

The parameter α is free for the choosing as is the location of the waypoint. The parameter α determines when the waypoint has been reached. If it is too small, the robot may circle the waypoint for a long time, missing the waypoint. If α is too large, the robot may pursue the target before it has actually overcome the barrier leading back to the same local optima exhibited by the high-level fitness function. If both α and the waypoint position are set appropriately, this method should succeed at the task.

E. Low-level Control

The lowest-level of control is usually direct position-, velocity-, or force-control of the robot motors. Depending on the morphology and motors of the robot, the low- and mid-levels of control may blur. For instance, a remote control toy car may be easily maneuvered using velocity control of the wheels, in which case mid- and low-level control is essentially the same. However, maneuvering an articulated body such as the one used in this paper (see Fig. 1) by position-control of its joints is not easily done, so the low-level control of its joints is entirely distinct from a mid-level controller where one may push a button labeled "forward". Much research in ER attempts to automatically derive a low-level controller to make mid- or high-level controller available for human operators.

If low-level control can be done by humans at an expert level, Abbeel et al. demonstrate a technique that records the expert human demonstration, sensor data, and use a dynamics model to derive a mid-level controller [1].

F. Hybrid Low-level Demonstration and High-level Fitness

The method we propose is a hybrid of high-level fitness and low-level user demonstrations. Instead of having a user create a fitness function or choose between a set of preferences, the user demonstrates what he or she wants the solution to do. In case of a robot, this means moving the joints or setting their positions at a particular time. The user demonstration is not expected to be complete or expertly done, so it is not clear how much weight one should place on the user demonstration. Either extreme is unsatisfactory: Respect the user too much,

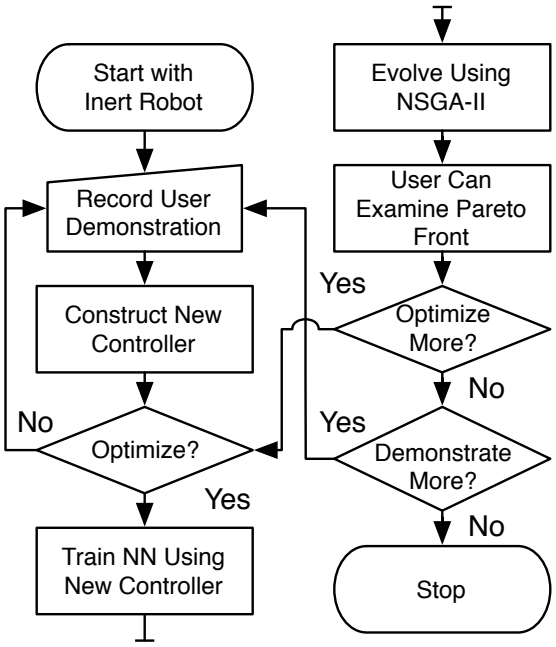

Figure 3. The interactive system works as follows. 1) The initial robot controller does not move ($\theta_i(t) = 0$). 2) A user may record a demonstration by moving the joint positions in time. Or a user may sweep over the time interval and set joint positions for a specific time t. 3) A new controller $\theta_i(t)'$ is constructed for each change the user makes in real-time. 4) Once a user is satisfied with the demonstration, he or she may elect to evolve the controller for a number of generations, 5) at which point the derived controller is used to train a new Neural Network (NN) (trained until 5000 epochs elapse or the MSE is less than 0.001). 6) The new NN is seeded into the initial generation used by NSGA-II. 7) The user is presented with the Pareto front, and may switch between the individuals. 8) The user may optimize further, or 9) the user may make further demonstrations with a particular individual, or 10) the user may stop.

and no solution better than what the user provided will be found. Respect the user too little, and the user demonstration will have no bearing on the solution nor any resemblance to the user demonstration.

To cast a wide net between respecting the user demonstration while allowing for exploration of effective solutions, this paper uses a multi-objective approach. The first objective is the high-level fitness function already defined in Eq. (1), which is the distance between the robot at the end of the evaluation and the target. The second objective is the User Demonstration Error (UDE), which captures how much a controller differs from what the user demonstrated.

$$[f_{\text{hybrid}}]_1 = f_{\text{high}} = ||\mathbf{r}_r(t_f) - \mathbf{r}_t|| \tag{6}$$

$$[f_{\text{hybrid}}]_2 = UDE \tag{7}$$

Before defining the UDE, let us take a step back and discuss how the system works with the user. Figure 3 shows a detailed flow of user interaction. Focusing on the initial user interaction, the system allows the user to manipulate the robot as though it were only a low-level remote control (because the initial controller is set to $\theta_i(t) = 0$ by making the NN weights all zero). In general though, the user can record his or her demonstration on top of the current controller and see

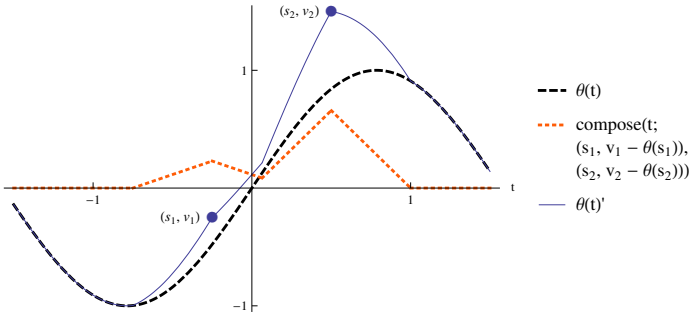

Figure 4. This is an example of augmented controller construction. Given an original controller $\theta(t)$, a user demonstration of two points (s_1, v_1) and (s_2, v_2), we wish to construct an augmented controller $\theta(t)'$ that satisfies the user demonstration. Broadly the operation takes the difference between the user demonstration points and the controller $v_i - \theta(s_i)$, smooths them out with $\text{compose}(t; \ldots)$, then adds that to the original controller $\theta(t)$ to produce the augmented controller $\theta(t)' = \theta(t) + \text{compose}(t; \ldots)$. In this paper $\theta(t)$ is implemented by a neural network.

it integrated immediately. This requires that we can take an existing controller $\theta_i(t)$ and construct an augmented controller $\theta_i(t)'$ that incorporates the user demonstration. The UDE error can be defined more easily once the augmented controller is defined. Thus two operations are performed with a user demonstration: 1) augmented controller construction and 2) user demonstration error calculation.

1) Constructing an Augmented Controller: Figure 4 shows graphically what the this section explains in detail: how to construct an augmented controller based off a user demonstration. First let us define what a user demonstration is. A user demonstration is represented by a set of tuples. Each tuple consists of the time, joint index, and joint value (s, i, v). This is interpreted to mean at time s the user has demonstrated that joint i should be at position v. Consider a user demonstration that consists of only one tuple (s, i, v) with no prior controller, i.e. $\theta_i(t) = 0$. Without loss of generality, one can omit the joint index i and corresponding subscript on the controller $\theta(t)$. Thus the tuple under consideration is only the time offset and joint value (s, v). The construction of the augmented controller $\theta(t)'$ will proceed incrementally, starting with the special case of one tuple and proceeding to the more general case.

First the user demonstration is "smoothed out" with a triangle element (Fig. 5) since the controller is expected to interact with a continuous system. The triangle base duration for the controller b_c is chosen to be 1 second.

$$\theta(t)' = \text{tri}(t; s, b_c, v) \tag{8}$$

In the case of a user demonstration of two tuples (s_1, v_1) and (s_2, v_2) where $v_1, v_2 \geq 0$, one must compose these triangle elements some how. Because the time offsets s_1 and s_2 may differ by an arbitrarily small amount, composition by addition is inadvisable because the triangle bases would overlap. Instead, the triangle elements are composed using the $\max$ function.

$$\theta(t)' = \max(\text{tri}(t; s_1, b_c, v_1),\ \text{tri}(t; s_2, b_c, v_2)) \tag{9}$$

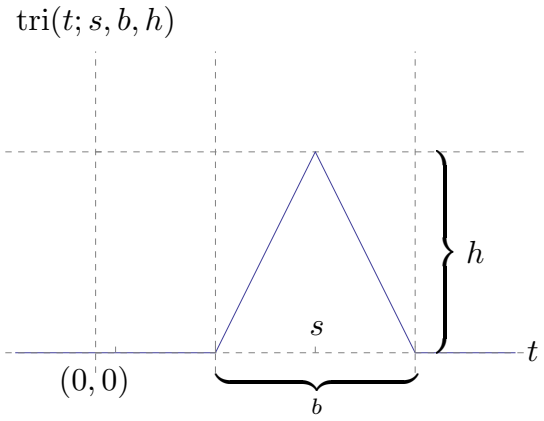

Figure 5. The triangle element "smooths out" a user demonstration. It is denoted $\text{tri}(t; s, b, h)$ and fully described by its parameters: time offset s, base duration b, and height h. It could be substituted by other smoothing functions, e.g., a Gaussian function.

For cases where $v_1, v_2 < 0$, the min function is used instead of max.

$$\theta(t)' = \min(\text{tri}(t; s_1, b_c, v_1),\ \text{tri}(t; s_2, b_c, v_2)) \tag{10}$$

The general case for a user demonstration of n tuples (s_1, v_1), $(s_2, v_2), \ldots, (s_n, v_n)$. We separate them in two sets: 1) the positive set is relabeled as (s_1, u_1), $(s_2, u_2), \ldots$ for $v_i \geq 0$, and 2) the negative set is relabeled as (s_1, w_1), $(s_2, w_2), \ldots$ for $v_i < 0$. Because this formulation will be used again, we will name the function compose which accepts the time t and a set of tuples as parameters.

$$\theta(t)' = \text{compose}(t; (s_1, v_1), \ldots) \tag{11}$$

$$\begin{aligned} &\text{compose}(t; (s_1, v_1), \ldots) \\ &\quad = \max(\text{tri}(t; s_1, b_c, u_1), \quad \text{tri}(t; s_2, b_c, u_2), \ldots) \\ &\quad + \min(\text{tri}(t; s_1, b_c, w_1), \quad \text{tri}(t; s_2, b_c, w_2), \ldots) \end{aligned} \tag{12}$$

However, when the user demonstrates an action, we do not assume that no prior controller exists. With a given prior controller $\theta(t)$, which in the case of this paper is a NN, we must amend Eq. (11) to only incorporate the difference between the prior controller and what the user demonstrated. That completes the construction of an augmented controller $\theta(t)'$.

$$\begin{aligned} \theta(t)' = \theta(t) + \text{compose}(t; \quad & (s_1, v_1 - \theta(s_1)) \\ & (s_2, v_2 - \theta(s_2)) \\ & \vdots \\ & (s_n, v_n - \theta(s_n))) \end{aligned} \tag{13}$$

2) Calculating the User Demonstration Error: Figure 6 shows graphically how the UDE is calculated for some controller in question $\phi(t)$. This section describes the operation in full detail.

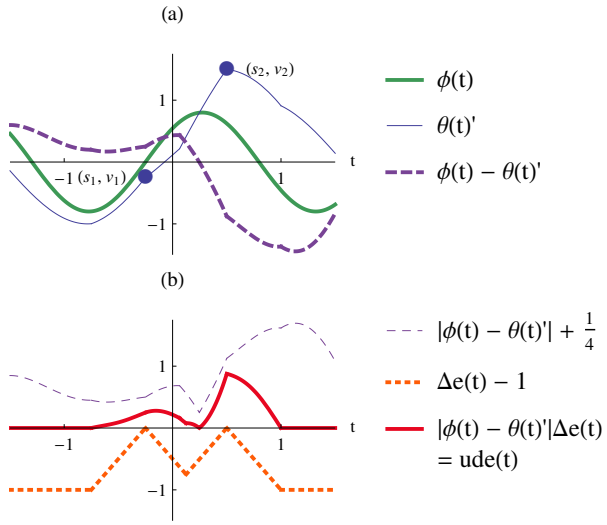

Figure 6. This is an example of calculating the User Demonstration Error (UDE) for some controller $\phi(t)$. Part (a) shows the user demonstration points, the controller in question $\phi(t)$, the augmented path $\theta(t)'$, and the difference between the two. Part (b) shows the intermediate steps to calculate the UDE at time t: We examine the absolute difference between the controller in question $\phi(t)$ and the augmented controller $\theta(t)$. (Note: the added constants are only to separate the graphs visually.) If the user has made no demonstration near some time s, then there can be no error which is captured by the $\Delta e(t)$ term. Multiplying these two terms together supplies the UDE at time t, and integrating that function over the time interval (not shown) is the UDE.

Given some controller $\phi(t)$, how different is it from what the user demonstrated? The answer to this question is captured quantitatively by the User Demonstration Error (UDE). In defining this error, there are three driving considerations: 1) When the user demonstrates v at time t, there should be a maximum error at time t. 2) When the user has performed no demonstration on or near time t, there should be no error at time t. 3) In between those extremes, the error should be at an intermediate value. Thus if the user has demonstrated nothing, the UDE is zero by definition. These considerations point toward to using the same triangle element and compose function to calculate the error, albeit with slight differences.

The definition is built up incrementally by considering the simplest case first. Let the user demonstration error at time t be denoted $\text{ude}(t)$. Given one user demonstration (s, v), the prior controller $\theta(t)$, and a new controller $\phi(t)$. What is the $\text{ude}(t)$ for $\phi(t)$? Using the above method (Section II-F1) one constructs augmented controller $\theta(t)'$ based off the prior controller $\theta(t)$ used when the demonstration was made.

$$\text{ude}(t) = |\phi(t) - \theta(t)'|\Delta e(t) \tag{14}$$

$$\Delta e(t) = \text{tri}(t; s, b_e, 1) \tag{15}$$

Equation (14) is similar to $\text{ude}(t) = |\phi(t) - v|\Delta e(t)$ except that Eq. (14) takes into account what the prior controller $\theta(t)$ does on the periphery of the user demonstration. The error delta Eq. (15) captures the idea that at time s, the user demonstrated the controller should be at value v; therefore, that should be the maximum error. Outside of the base duration b_e, the error should be zero and in between it is some intermediate error.

The base duration of the error b_e is set to 1 second, the same as the base duration for the augmented controller b_c.

The general case of the $\mathrm{ude}(t)$ for a user demonstration with n tuples is a simple extension.

$$\mathrm{ude}(t) = |\phi(t) - \theta(t)'|\Delta e(t) \tag{16}$$
$$\Delta e(t) = \mathrm{compose}(t; (s_1, 1), \ldots, (s_n, 1)) \tag{17}$$

Finally, the sum of time dependent $\mathrm{ude}(t)$ over an interval of time provides the user demonstration error.

$$\mathrm{UDE} = \int_0^{t_f} \mathrm{ude}(t)\,\mathrm{d}t \approx \sum_{i=0}^{m} \mathrm{ude}(i\,\Delta t) \tag{18}$$

G. Surrogate User

This paper uses a user surrogate to perform this comparison. This user surrogate sets the initial user demonstration only. It does not iteratively update with new user demonstrations, which would be expected from a human user. Iterative updates are omitted because it is not clear how to program a surrogate that reacts to the robot's current behavior. Initially, however, the robot does nothing by fiat, so that allows us to apply a demonstration that was made by a user, in this case one of the authors, without requiring it to be reactive.

When initially using this system, one essentially has low-level control of the quadruped. It is difficult initially to make the robot locomote. One of the authors determined that by oscillating any of the joints attached to the root body, one may effect motion in the direction of that leg. Whether people can in general do likewise is an open question.[2] With this small amount of control, one may set the robot off in any cardinal direction. For this task, the robot is set off towards the positive x axis (to the right). One might think this is cheating since the user knows this will help the robot get around the barrier; however, that is in fact the point of this experiment—to show that the user through demonstration can help avoid local optima.

The user demonstration used by the surrogate user is shown in Fig. 7 (a). It is described as $\{(0.35, 1, -\frac{3\pi}{16}), (0.75, 1, \frac{7\pi}{40})\}$. This user demonstration means at 0.35 seconds set joint 1 to $-\frac{3\pi}{16}$ radians and at 0.75 seconds set joint 1 to $\frac{7\pi}{40}$. These values were determined by making a gait that walked to the right interactively (see Fig. 3), then recording the user demonstration values to only two significant figures. Because this is a locomotive task, we chose for the time the user specified to be connected to the fast time measure which causes the demonstration to effectively repeat every second as shown in Fig. 7 (b).

[2]This is merely anecdotal but during a live demo, a person did accomplish this obstacle avoidance task using *only* low-level control of the quadruped. Although it was not clear that they discovered a means of repeatable mid-level control.

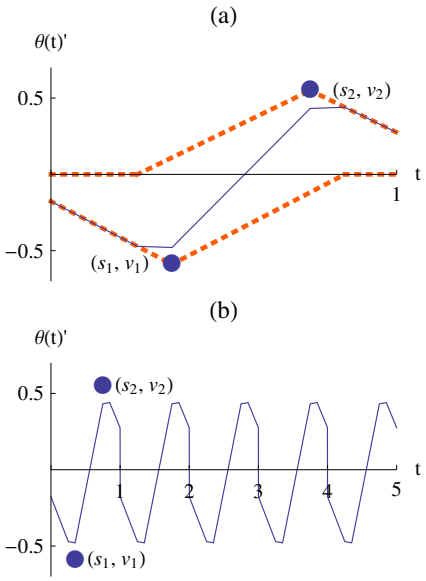

Figure 7. This figures shows the surrogate user demonstration. Part (a) shows the two user demonstrated points. The dashed lines represent the triangle elements that "smooth" the demonstration, and the solid line represents the augmented controller. Part (b) shows that the augmented controller is periodic due to the fast time measure in the neural network input. Thus it produces an oscillating pattern, which in turn oscillates one proximal hinge on the quadruped causing it to move to the right ($+x$). Note: although this is the only user demonstration used, it places a pressure throughout the entire evolutionary trial to select for controllers that preserve a similar motion for that joint.

III. Results

The optimization algorithm used was NSGA-II [11] with a population of 20 individuals and 100 generations. The physics simulator used time steps of 1/60 simulated second. Each robot was evaluated for 30 simulated seconds. Figure 8 shows the proportion of success trials from 30 independent trials of each fitness function and its variants. A successful trial is defined as any trial that contains one individual that comes within 4.5 units of the target object at any time during the evaluation. The same results are provided numerically in Table I. The hybrid fitness function outperforms the high-level fitness function and is competitive with the mid-level (waypoint) fitness function and outperforms it significantly in one case.

Some variations of the hybrid fitness function were also evaluated to try and pinpoint what parts of the hybrid fitness function are effective. Three variants were considered: (1) "dependent", (2) "no seed", and (3) "no error". The first variant "dependent" is probably best explained by considering that normally the user demonstration is independent for each joint, i.e., changing one joint has no effect on the other joints. However, one can imagine that when a user actively changes some joint, this choice also exhibits a passive preference for the current positions of the other joints. Operationally this means that when a user actively demonstrates (s_1, v_1), the system also records $(s_1, v_i = \theta_i(s_1))$ for all other joints. This has no effect on constructing the augmented controller but does act to preserve those passive preferences when computing the UDE. As Table I shows the "dependent" variant did significantly worse than the hybrid fitness function. However, because of the limitation of the initial robot being inert, it seems like this variation may be worth investigating further.

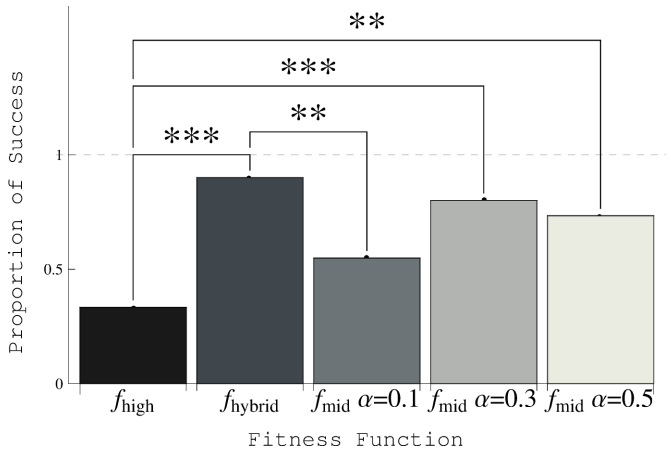

Figure 8. The results show the proportion of successful evolutionary trials that produced a robot that avoided the obstacle. Thirty trials were performed for each fitness function with a population of 20 for 100 generations using NSGA-II. The stars represent significantly different results determined by the Fischer Exact Test. The high-level fitness function minimizes distance to target. The mid-level fitness function attempts to first go to a waypoint then go toward the target. The hybrid fitness function minimizes the distance to the target and the user demonstration error provided by the user surrogate. The hybrid fitness function does significantly better than the high-level fitness and is better or competitive with the mid-level.

Table I. SUMMARY OF EXPERIMENTAL RESULTS. THE P-VALUES ARE OBTAINED BY COMPARING EACH EXPERIMENT WITH f_{HYBRID} USING THE FISCHER EXACT TEST.

Experiment	Percent Successful	p-value
f_{high}	33.3%	$p < 0.001$
f_{hybrid}	90.0%	$p = 1$
f_{mid} $\alpha = 0.1$	54.8%	$p < 0.01$
f_{mid} $\alpha = 0.3$	80.0%	$p = 0.5$
f_{mid} $\alpha = 0.5$	73.3%	$p = 0.2$
f_{hybrid} dependent	53.3%	$p < 0.01$
f_{hybrid} no seed	96.7%	$p = 0.6$
f_{hybrid} no error	40.0%	$p < 0.001$

The "no seed" variant differs in the following way. Normally before evolution begins, the initial population is seeded with a NN controller. This NN is found by back-propagation based off the augmented controller (described in Fig. 3), so that the NN produces a pattern very close to what is shown in Fig. 7. Thus it starts out walking to the right. The "no seed" variant does not seed the initial population with a walking NN. Instead it uses a random initial population. As Table I shows the performance of "no seed" variant is comparable to the hybrid fitness function. This is intriguing because it means that the UDE pressure is sufficient to select for controllers that presumably walk to the right.

Lastly, the "no error" variant does seed the population with a NN that walks to the right, but it does not use the user demonstration error (UDE) as its second objective. Instead it uses some constant value for the second objective. The "no error" variant does significantly worse than the hybrid fitness function probably because the seeded walker is eliminated from the population and the evolutionary trial degrades into a replication of the high-level fitness function. This result shows that the UDE is an integral element to this method's efficacy.

IV. DISCUSSION

In this paper we propose that allowing users to make low-level, inexpert demonstrations to control articulated bodies may help avoid local optima, without requiring the user to reformulate the fitness function. This might appear to contradict prior emphasis that humans are not likely particularly good at unassisted low-level control of articulate bodies. However, this method can be seen as an assisted low-level control that can optimize suggestions offered by the user.

The task environment presented in this paper is easily decomposable into discrete steps: (1) go to here then (2) go to there, which is what the mid-level (waypoint) fitness function relies on. However, low-level user demonstration might be more profitably applied to tasks that have no such obvious mid-level decomposition. For instance, consider the task of jumping over a large gap with a quadruped. Imagine trying to solve it by placing waypoints for the robot to move toward. If one places a waypoint in the middle of the gap, does that help the robot achieve the task in any way? Suppose one places a string of waypoints along the arc of an ideal jump. Does that help? It is not clear to us that it would help. Instead consider using a system like the one presented in this paper with the same task. We have bodies. We can jump. We have an intuition about how to do it. Perhaps the jump ought to be initiated from a crouched position somewhat like coiling a spring, then a burst of movement releasing the spring. Supplying that information solely through low-level control would probably not cause the robot to jump over the gap, but this system allows us to offer just such a suggestive solution that can be further refined by an automatic search procedure.

A. Limitations and Future Work

The surrogate user is a large limitation of this work. To support the claim that user demonstration is an effective means of combining the human intuition with automated search, a study must be done with actual users.

The obstacle avoidance task can be easily decomposed into a set of steps for a mid-level controller, which makes low-level control seem inappropriate. Investigating a number of different tasks which vary in terms of difficulty depending on the level of control—high, mid, or low—would be enlightening. For example, the task of jumping over a gap seems more appropriate for low-level control.

The construction of the augmented controller from the user demonstration was only dependent on time. It did not have to take into account the line-of-sight inputs since the robot begins behind the wall. However, in the more general case, the controller may have inputs such that the user demonstration is meant to suggest a relationship with these inputs, e.g., if the robot is headed toward an obstacle, turn right, which is input dependent—not at time s_1, turn right, which is time dependent.

This system only allows for the user to demonstrate through low-level control. However, as mentioned in the introduction, user demonstration may be provided at low-, mid-, or high-levels of control. User demonstration with high-level control, however, is still susceptible to becoming trapped in the same kind of local optima presented here. This suggests that perhaps the user ought be able to choose the level of control appropriate to the task.

V. Summary

This paper described an interactive system that accepts user demonstrations via low-level control. It compared the system with a user surrogate to a high-level fitness function and a mid-level fitness function with three different parameter settings. The system achieved a 90% success rate over 30 trials, which did significantly better than the high-level fitness function success rate of 33% ($p < 0.001$) and significantly better than the mid-level fitness function ($\alpha = 0.1$) rate of 55% ($p < 0.01$) and was comparable for the other parameter settings which achieved a success rate of 80% ($p = 0.5$) and 73% ($p = 0.2$). Some variants of the system were exercised to determine what features were strictly required to achieve its performance. Limitations of this system were discussed along with suggestions for future work. This paper submits that IEAs can accept more than user preferences, and user demonstrations may be acquired through low-, mid-, and high-level control. The ultimate hope of this line of research is to use user demonstrations to harness inexpert human intuition such that one may profitably crowd-source challenging tasks in evolutionary robotics.

VI. Acknowledgments

This work was supported by National Science Foundation Grant PECASE-0953837, DARPA M3 grant W911NF-1-11-0076, and the University of Vermont Complex Systems Graduate Research Assistantship. The authors would like to thank the reviewers for their feedback.

This paper made use of the following software packages: Bullet Physics Library, NSGA-II, Fast Artificial Neural Network Library (FANN), GNU Guile Scheme, and TouchOSC.

References

[1] P. Abbeel, A. Coates, and A. Y. Ng. Autonomous Helicopter Aerobatics through Apprenticeship Learning. *The International Journal of Robotics Research*, 29(13):1608–1639, Nov. 2010.

[2] M. Bain and C. Sammut. A Framework for Behavioural Cloning. *Machine Intelligence 15, Intelligent Agents*, pages 1–37, July 2001.

[3] J. Bongard, P. Beliveau, and G. Hornby. Avoiding local optima with interactive evolutionary robotics. pages 1405–1406, 2012.

[4] J. Bongard and H. Lipson. From the Cover: Automated reverse engineering of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 104(24):9943–9948, June 2007.

[5] J. C. Bongard. Multiobjective Preference-Based Policy Learning for Evolutionary Robotics. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2013.

[6] J. C. Bongard and H. Lipson. Nonlinear system identification using coevolution of models and tests. *Evolutionary Computation*, 2005.

[7] C. Caldwell and V. S. Johnston. Tracking a Criminal Suspect through "Face-Space" with a Genetic Algorithm. In *Proceedings of the Fourth International Conference on Genetic Algorithm*, pages 416–421. Morgan Kaufmann Publisher, 1991.

[8] S. Calinon, F. Guenter, and A. Billard. On learning, representing, and generalizing a task in a humanoid robot. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 37(2):286–298, 2007.

[9] J. Clune and H. Lipson. Evolving Three-Dimensional Objects with a Generative Encoding Inspired by Developmental Biology. *Proceedings of the European Conference on Artificial Life*, pages 1–8, June 2011.

[10] R. Dawkins. *The Blind Watchmaker*. Why the Evidence of Evolution Reveals a Universe Without Design. W. W. Norton & Company, 1986.

[11] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, Apr. 2002.

[12] C. B. Eiben, J. B. Siegel, J. B. Bale, S. Cooper, F. Khatib, B. W. Shen, F. Players, B. L. Stoddard, Z. Popovic, and D. Baker. Increased Diels-Alderase activity through backbone remodeling guided by Foldit players. *Nature Biotechnology*, 30(2):190–192, Jan. 2012.

[13] G. Hornby and J. Bongard. Learning Comparative User Models for Accelerating Human-Computer Collaborative Search. In P. Machado, J. Romero, and A. Carballal, editors, *Evolutionary and Biologically Inspired Music, Sound, Art and Design*, pages 117–128. Springer Berlin Heidelberg, 2012.

[14] G. Hornby and J. C. Bongard. Accelerating human-computer collaborative search through learning comparative and predictive user models. *Proceedings of the 14th international conference on Genetic and evolutionary computation conference*, pages 225–232, 2012.

[15] I. V. Karpov, V. K. Valsalam, and R. Miikkulainen. Human-assisted neuroevolution through shaping, advice and examples. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. ACM, July 2011.

[16] P. Krcah. Solving deceptive tasks in robot body-brain co-evolution by searching for behavioral novelty. In *Intelligent Systems Design and Applications (ISDA), 10th International Conference on*, pages 284–289, 2010.

[17] M. Schmidt. Actively probing and modeling users in interactive coevolution. *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, 2006.

[18] J. Secretan, N. Beato, and D. B. D'Ambrosio. Picbreeder: A case study in collaborative evolutionary exploration of design space. *Evolutionary Computation*, 2011.

[19] K. Sims. Artificial evolution for computer graphics. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*. ACM, July 1991.

[20] H. Takagi. Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation. *Proceedings of the IEEE*, 89(9):1275–1296, 2001.

[21] E. Vaughan, E. A. Di Paolo, and I. Harvey. The evolution of control and adaptation in a 3D powered passive dynamic walker. *Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems, Artificial Life IX*, pages 139–145, 2004.