Nonlinear Inverse Reinforcement Learning for Human Performance Enhancing Feedback

A Thesis

Presented to

the Faculty of the School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment

of the requirements for the Degree

Master of Science (Systems and Information Engineering)

by

Mark Rucker

December 2018

Abstract

Many of the today's most wicked problems are rooted in human behavior: whether it is distributing professional skills, managing chronic health conditions or responsibly sharing a common resource. Previous generations have dealt with such challenges through methods such as market forces, natural consequences and governmental policies. In this paper we propose a new, potential approach. We hypothesize that Inverse Reinforcement Learning can learn multifaceted, nonlinear reward functions which can drive predictable behavior change. To test this theory we create a stochastic, online control task and modify the task rewards according to IRL. A quasi-experimental design with a pre and post test, 4 treatment levels (plus a control group) and 400 participants per group (n≈2000) suggests that IRL rewards can cause behavior changes in a predictable manner.

Contents

Contents **iv**

1 Introduction **1**

2 Literature **2**

2.1 Human Behavior Change and Learning 2

2.2 Inverse Reinforcement Learning 3

3 Preliminaries **4**

4 Algorithms **5**

4.1 Inverse Reinforcement Learning 5

4.1.1 Kernel-Projection Algorithm 5

4.1.2 KPIRL Performance Comparisons 7

4.2 Reinforcement Learning 10

4.2.1 Kernel-Lookup Approximation Algorithm 10

4.2.2 KLA Performance Comparisons 16

5 Experiment **18**

5.1 Hypothesis 18

5.2 Design 19

5.3 Dependent Variable 19

5.4 Game . . . 20

5.4.1 Stochasticity . . . 20

5.4.2 Framing . . . 21

5.4.3 Directions . . . 21

5.5 Treatment . . . 22

5.5.1 Experts . . . 22

5.5.2 Rewards . . . 23

5.5.3 Delivery . . . 24

5.6 Analysis . . . 25

5.6.1 Blocking Variables . . . 25

5.6.2 Hypothesis Tests . . . 26

5.6.3 Alternative Explanations . . . 29

6 Application 30

6.1 MDP Model . . . 30

6.1.1 Raw Observations . . . 31

6.1.2 State Space . . . 31

6.1.3 State Features . . . 31

6.1.4 Expert Trajectories . . . 32

6.1.5 Action Space . . . 32

6.1.6 Transition Probabilities . . . 32

6.1.7 Reward and Value Basii Functions . . . 33

6.1.8 Value Function Structure . . . 33

6.2 Communicating the Reward Function . . . 34

6.2.1 Transform to Increase Understanding . . . 34

6.2.2 Smooth to Improve Understanding . . . 36

6.2.3 Forecast to improve understanding 37

7 Conclusion **38**

Appendices **40**

A Full Data Tables **41**

Bibliography **43**

Introduction

Spare the rod, spoil the child. The impact of environmental feedback on human behavior has long been understood at an intuitive level. However, it wasn't until the pioneering work of Thorndike and Pavolov in the late 19th century, that such intuition was formulated as a scientific system of knowledge. Fast forward more than a century, and much of the current state of the art in machine learning and robotics draws inspiration from Thorndike's and Pavlov's theories in an area of research known as Reinforcement Learning (RL).

In the classic research of Thorndike and Pavlov, environmental signals were known. The question then was how did these signals effect behavior. Similarly, within the machine learning community, RL also assumes the environmental feedback signal is known (referred to as the reward signal). The challenge is to learn optimal behavior via repeated interactions with an environment and its reward signal.

Recently, researchers have started to turn the field of RL on its head. Instead of assuming a reward signal is known, and the behavior is unknown, these researchers assume the behavior is known and its reward signal is unknown. This approach is known as Inverse Reinforcement Learning (IRL), and a number of novel applications have been found for it (listed in more detail below).

This paper seeks to show the efficacy of one more novel IRL application. We hypothesize that if one uses IRL to learn the reward function of an individual who is good at performing a task, then one could communicate this reward to a new individual and that individual's performance would increase. Likewise, if the reward of a poor performer is communicated, then performance should decrease.

To test this hypothesis a quasi-experiment is conducted with pre- and post-obesrvations of performance, four treatments and a control. Performance is measured with a stochastic pointing game. The results of the experiment show that an IRL reward can in fact improve human performance.

Section 2 provides a very cursory literature review of research related to this experiment. Section 3 covers a number of technical terms and formulations for individuals unfamiliar with RL or IRL. Section 4 describes the IRL and RL algorithm developed for the experiment. Section 5 outlines the experiment and its results. Section 6 details the specific models and techniques used to interact with participants. And Section 7 gives a brief discussion of the implications of this research along with future research questions.

Literature

2.1 Human Behavior Change and Learning

The literature concerned with reward driven human behavior change is large. In the course of this research a number of inspirations are found in disparate areas such as motor learning, health behavior change, and behavioral economics.

Research in the motor learning community has found that concurrent visual feedback can improve performance for complex tasks[1]. In this community, a complex task can be defined as one which is "ecologically valid" outside of a lab [2]. Other work suggests that an important component of motor skill learning is model based operant conditioning – which is similar to the IRL paradigm [3]. This suggests that environmental signals can aid human learning (though this community does not refer to these signals as rewards).

Similarly, experiments in the field of behavioral health have shown that properly structured financial incentives can improve health outcomes [4, 5]. Though the ability of these incentives to create sustained change has yet to be decided [6]. Once again, this field shows that environmental feedback can alter human behavior, but the feedback is often quite simple, and usually financial.

Finally, rational choice theory has long been used as an assumption in economic models. The idea of rational choice is fairly simple, an individual, when given a choice between a benefit or a cost, will always choose the benefit [7]. Unfortunately, this elegant theory, while not believed to be entirely wrong, has been shown lacking many times [8–10].

Despite the clear differences between motor learning, behavioral health and economics, we feel these fields share a common thread. That is, they all provide evidence that an appropriately motivated individual, with good information, will in general, perform better. Perhaps the most pure example of this is, that we found, was [11]. In this study the researchers were able to use human designed feedback, learned from human experts, to improve the performance of future participants.

2.2 Inverse Reinforcement Learning

Apart from the experimental design for this study, and its place within the larger research community, we also relied on considerable work from the IRL community. IRL originally grew out of the robotics community, where researchers were interested in teaching robots to perform tasks at a human level of proficiency [12].

One of the first major success for this technique was autonomous aerobatics [13]. And since then a number of pioneering articles have been published: predict human vehicle routes [14], measure cultural preferences in negotiations [15], support cooperation between humans and robots [16], construct an agent based model from data [17] and infer an airplane pilot's intent based on flight history [18].

Preliminaries

There are an number of technical definitions used within the RL and IRL communities. Perhaps most important is the Markov decision process (MDP). An MDP is a mathematical model of a stochastic process involving an agent that is able to make choices to alter the outcomes.

Formally, an MDP is defined as a five-tuple $<S, A, P, R, \gamma>$ where S is a set of states, A is a set of actions, P is a set of PMF's indexed by (S, A), R is a reward function $R : S \mapsto \mathbb{R}$ and $\gamma \in [0, 1]$ is a discount factor.

Using the MDP formalism, RL algorithms assume the reward, R, is known and that a policy of behavior, $\pi : S \mapsto A$, needs to be solved for. An optimal policy, π^*, is defined as one that maximizes the expression $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi^*, P] \; \forall \; s_0 \in S$.

The expression that an optimal policy maximizes is called the value function and is often written recursively in algebraic form as $V^*(s) = R(s) + \gamma\mathbb{E}[V^*(s')|\pi^*, P]$. This form is known as the Bellman equation.

Using the MDP formalism For IRL, one assume the policy of behavior is known, π_E and the reward function that makes this optimal is learned (i.e., $V(s|\pi_E, R) \geq V(s|\pi, R) \; \forall \; s \in S, \pi \in \Pi$).

Perfect knowledge of π_E is often not possible. Therefore, most IRL algorithms rely on what is referred to as expert trajectories. An expert trajectory is a sequence of states created by an individual following π_E. These trajectories are denoted as S_E and are indexed by $\{m, t\}$, where m is the trajectory index and t is the time-step within trajectory m.

Algorithms

4.1 Inverse Reinforcement Learning

4.1.1 Kernel-Projection Algorithm

The Kernel-Projection IRL algorithm (KPIRL) learns an R from S_E. KPIRL is a non-linear extension to the linear algorithm, Projection IRL (PIRL) in [19]. The PIRL algorithm has strong theoretical guarantees for convergence rate and arbitrary closeness around the expert feature expectation (4.6). KPIRL maintains these guarantees.

KPIRL begins by assuming access to a basii map for each state defined as any function where

$$\phi : S \mapsto \mathbb{R}^k \tag{4.1}$$

This mapping is arbitrary and is chosen to fit the problem (See section 6.1.7 for ours). Using the mapping ϕ, a value known as "feature expectation" is defined for every possible π as:

$$\mu(\pi) = \mathbb{E}_{s_0 \sim D}[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) | \pi] \in \mathbb{R}^k \tag{4.2}$$

An estimate of the expert's "feature expectation", $\hat{\mu}_E$, is calculated from trajectories, S_E.

$$\hat{\mu}_E = \frac{1}{M} \sum_{m=1}^{M} \sum_{t=0}^{T} \gamma^t \phi\left(S_{E\{m,t\}}\right) \approx \mu(\pi_E) \tag{4.3}$$

This value, $\hat{\mu}_E$, is KPIRL's convergence point. That is, KPIRL finds a set of reward functions, $\mathcal{R}$, whose optimal policies, $\Pi^*_{\mathcal{R}}$, can be combined to produce a policy whose feature expectation is arbitrarily close to $\hat{\mu}_E$.

To extend PIRL to non-linear reward functions equation 4.1 is factored in the following way:

$$\phi(s) = \Phi^T \mathrm{e}_{j(s)} \tag{4.4}$$

Here, Φ is an $\mathbb{R}^{|S| \times |k|}$ matrix whose rows correspond to $[\phi(s_1)^T,\ \phi(s_2)^T,\ ...,\ \phi(s_n)^T]$, j maps each state to its index (e.g., $j(s_1) = 1$) and e_j is the j^{th} column vector in the $I_{|S|}$ identity matrix. With this factorization equations 4.2 and 4.3 become:

$$\mu(\pi) = \Phi^T \mathbb{E}_{s_0 \sim D}[\sum_{t=0}^{\infty} \gamma^t \mathrm{e}_{j(s_t)} | \pi] \in \mathbb{R}^k \tag{4.5}$$

$$\hat{\mu}_E = \Phi^T \frac{1}{M} \sum_{m=1}^{M} \sum_{t=0}^{\infty} \gamma^t \mathrm{e}_{j\left(s_{E\{m,t\}}\right)} \tag{4.6}$$

For notational convenience the rightmost expression in 4.5 and 4.6 is broken out as:

$$F(\pi) = \mathbb{E}_{s_0 \sim D}[\sum_{t=0}^{\infty} \gamma^t \mathrm{e}_{j(s_t)} | \pi] \in \mathbb{R}^k \tag{4.7}$$

$$\hat{F_E} = \frac{1}{M} \sum_{m=1}^{M} \sum_{t=0}^{T} \gamma^t \mathrm{e}_{j\left(s_{E\{m,t\}}\right)} \tag{4.8}$$

An intuitive interpretation of F is that it as represents the discounted expected visitation rate for each state in the MDP when following π. Without discounting, this would simply be the steady state of the Markov chain induced by the policy.

Using the factorization in 4.4 the equations in section 3.1 of [19] are rewritten as:

$$
\begin{aligned}
\bar{\mu}^{(i-1)} &= \Phi * \bar{F}^{(i-1)} \\
\bar{F}^{(i-1)} &= \bar{F}^{(i-2)} + \frac{(\mu^{(i-1)} - \bar{\mu}^{(i-2)})^T(\mu_E - \bar{\mu}^{(i-2)})}{(\mu^{(i-1)} - \bar{\mu}^{(i-2)})^T(\mu^{(i-1)} - \bar{\mu}^{(i-2)})}(F^{(i-1)} - \bar{F}^{(i-2)}) \\
&= \bar{F}^{(i-2)} + \frac{(\Phi * F^{(i-1)} - \Phi * \bar{F}^{(i-2)})^T(\Phi * F_E - \Phi * \bar{F}^{(i-2)})}{(\Phi * F^{(i-1)} - \Phi * \bar{F}^{(i-2)})^T(\Phi * F^{(i-1)} - \Phi * \bar{F}^{(i-2)})}(F^{(i-1)} - \bar{F}^{(i-2)}) \\
&= \bar{F}^{(i-2)} + \frac{(F^{(i-1)} - \bar{F}^{(i-2)})^T(\Phi^T\Phi)(F_E - \bar{F}^{(i-2)})}{(F^{(i-1)} - \bar{F}^{(i-2)})^T(\Phi^T\Phi)(F^{(i-1)} - \bar{F}^{(i-2)})}(F^{(i-1)} - \bar{F}^{(i-2)}) \\
w^{(i)} &= \mu_E - \bar{\mu}^{(i-1)} \\
&= \Phi * (F_E - \bar{F}^{(i-1)}) \\
R^{(i)}(s) &= (w^{(i)})^T\phi(s) \qquad (4.9) \\
&= (F_E - \bar{F}^{(i-1)})^T * (\Phi^T\Phi) * \mathrm{e}_{j(s)} \\
t^{(i)} &= ||\mu_E - \bar{\mu}^(i-1)|| \qquad (4.10) \\
&= \sqrt{(F_E)^T(\Phi^T\Phi)(F_E) + (\bar{F}^{(i-1)})^T(\Phi^T\Phi)(\bar{F}^{(i-1)}) - 2 * (F_E)^T(\Phi^T\Phi)(\bar{F}^{(i-1)})}
\end{aligned}
$$

In this rewrite Φ only occurs as a transpose of itself. This makes it possible to implicitly map the reward basii into a high-dimension space with the kernel trick. The kernel trick replaces the inner product in $(\Phi^T\Phi)_{ij} = \langle\Phi_i, \Phi_j\rangle$ with an alternative function. For the experiment below the Gaussian kernel is used: $(\Phi^T\Phi)_{ij} = exp(\frac{-||\Phi_i - \Phi_j||^2}{2\sigma^2})$.

Apart from these mathematical transformations the original PIRL algorithm remains unchanged. For convenience, the remaining steps of KPIRL algorithm are provided in algorithm 1.

4.1.2 KPIRL Performance Comparisons

Among existing non-linear IRL algorithms there are three notable standouts [20], [21] and [22]. Many consider GPIRL ([20]) to be the current state of the art and we benchmark

Algorithm 1 Kernel-Projection IRL (KPIRL)

```
1: initialize $\epsilon \leftarrow$arbitrary, $\pi^{(0)} \leftarrow$arbitrary, $\mu^{(0)} \leftarrow \mu(\pi^{(0)})$, $i = 1$     ▷ defined in (4.5)
2: loop
3:     compute $t^{(i)}$                                   ▷ defined in (4.10)
4:     compute $R^{(i)}(\cdot)$                            ▷ defined in (4.9)
5:     if $t^{(i)} < \epsilon$ then
6:         terminate
7:     Solve for $\pi^{(i)}$ using $R^{(i)}$               ▷ see Algorithm 2
8:     $\mu^{(i)} \leftarrow \mu(\pi^{(i)})$               ▷ defined in (4.5)
9:     $i \leftarrow i + 1$
```

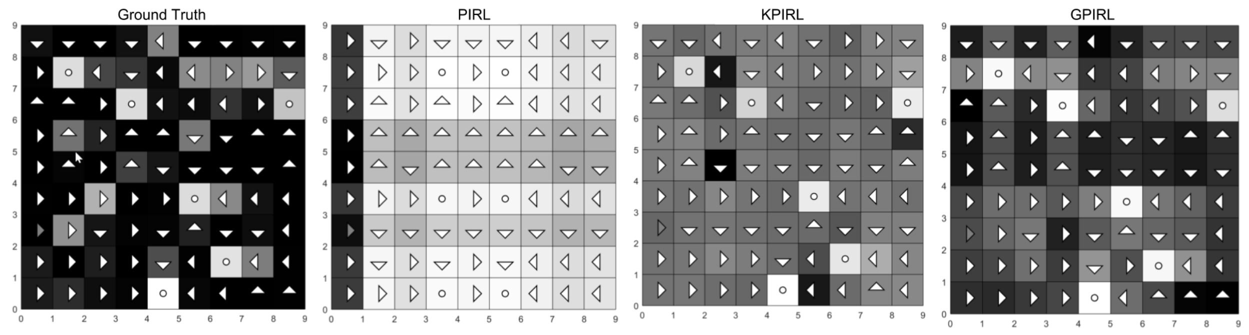


Figure 4.1: Reward for states are indicated by shading (i.e. white indicates high reward and black indicates low reward) while the arrows indicate the optimal policy when following the reward.

against it (as well as PIRL for reference).

To benchmark KPIRL a gridworld environment from the IRL-Toolkit is used (the IRL-Toolkit is a MATLAB environment created by Levine, which he has made freely available). The gridworld has $|S| = n^2$ states with 5 actions in each state (up, down, left, right and stay). All action transitions are deterministic. (see Figure 4.1 for an 81 state example of a gridlworld along with the results of several IRL algorithms).

States in this gridworld are assigned reward basii with non-linear relationships to the states' reward. Using this environment ten random random rewards are generated, along with expert trajectories. The performance of each algorithm is calculated in terms of speed (i.e., total runtime) and accuracy (i.e., what percent of the optimal value did the learned reward capture).

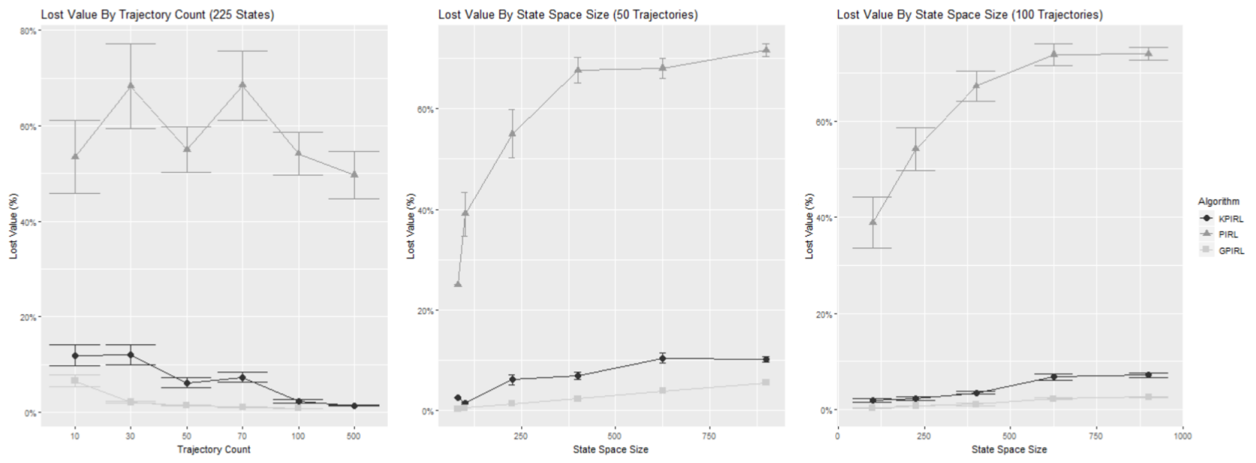


Figure 4.2: The percent of value lost when following the learned reward instead of the true reward.

In terms of accuracy, no matter the size of the state space or the number of expert trajectories, GPIRL is always the most accurate (see figure 4.2). KPIRL is a close second, typically about 5% behind GPIRL. And the linear algorithm, PIRL, is the least accurate (which is expected since the reward was a non-linear function of the basii).

In terms of speed, KPIRL and PIRL are the clear winners (figure 4.3). The runtime for both remained near zero for all state space sizes and number of expert trajectories. For GPIRL on the other hand, runtime increased nearly exponentially. For example, GPIRL took ten seconds to complete for one hundred states with one hundred trajectories and nine minutes to complete (a 54 times increase) for nine-hundred states with one hundred trajectories (a 9 times increase).

In the end, when dealing with large amounts of data and large state spaces, KPIRL seemes to be the better algorithm. It sacrifices a small amount accuracy for exponential gains in speed. On the other hand, when working with small amounts of data where accuracy is very important GPIRL seems to be the better algorithm.

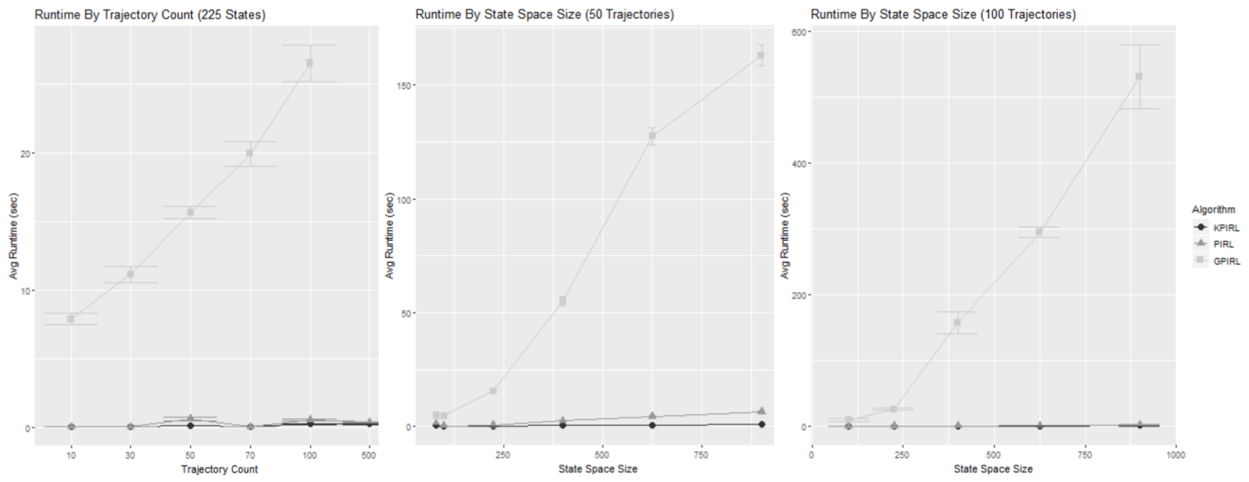


Figure 4.3: Runtimes for PIRL[19], KPIRL and GPIRL[20].

4.2 Reinforcement Learning

4.2.1 Kernel-Lookup Approximation Algorithm

The Kernel-Lookup Approximation algorithm (KLA) is an approximate RL algorithm that learns π from R. For the experiment, KLA is used as a part of KPIRL (c.f. line 7 in KPIRL). The KLA algorithm borrows heavily from the work of Powell (most notably, Approximate Dynamic Programming: Solving the Curse of Dimensionality [23]), and readers who would like to learn more are encouraged to read his text.

There are a number of existing approximate RL algorithms. Four of the more well known are Dyna-Q [24], Least-Squares Policy Iteration (LSPI) [25], Kernel-based Least Squares Policy Iteration (KLSPI) [26] and Kernel Smoothing [27].

In testing, none of these algorithms solved the experimental MDP well enough for KPIRL to work. This was due, in large part, to KPIRL's iterative maximum margin search through the reward space. This search means that, on each iteration, a new reward must be solved for, and that this reward is often arbitrary and considerably different from the reward function of the previous iteration.

To deal with the reward search challenge of KPIRL, KLA is designed to be:

- **Efficient** - to iterate through rewards in acceptable time
- **Approximate** - to handle a large transition and state space
- **Low-Bias** - to handle reward functions with few assumptions

The KLA algorithm most closely follows in the footsteps of Dyna-Q. KLA is more of an architecture than an algorithm. No rigorous performance guarantees are provided with KLA. And yet it performs very well in the comparison to LSPI and KLSPI shown in section 4.2.2.

KLA combines best practices from research in model-free learning (post-decision state), function approximations (Unbiased observations), exploration/exploitation and optimal learning (Value Function Updates). Each of these elements, along with the algorithm itself in 2 are explained in more detail below.

Post-Decision State

In traditional model-based RL techniques an optimal value function works in tandem with transition probabilities to determine the best action. This can be seen in the Bellman optimality equation where max a is determined by $P_{s,a}$ and V^*:

$$V^*(s) = R(s) + \gamma \max_{a \in A} \{\mathbb{E}[V^*(s')|P_{s,a}]\}$$

A well known solution to this dependency in the RL literature is to learn a Q-function rather than a value function. With a Q-function, choosing max a becomes a deterministic optimization problem:

$$Q^*(s,a) = \mathbb{E}[V^*(s')|P_{s,a}]$$
$$V^*(s) = R(s) + \gamma \max_{a \in A} \{Q^*(s,a)\}$$

Algorithm 2 Kernel-Lookup Approximation (KLA)

```
    initialize $InitStates$
    initialize $N, M, T, W$
    initialize $n \leftarrow 1$
    initialize $V_a^{\{1\}}(s_a) \leftarrow$arbitrary
 5: **while** $n \leq N$ **do**
        **if** $n \mod 4 = 1$ **then**
            $OnPolicyDist \leftarrow InitStates$
        $m \leftarrow 1$
        **while** $m \leq M$ **do**
10:         $t \leftarrow 1$
            $s^{\{t\}} \leftarrow \text{random}\{OnPolicyDist\}$
            **while** $t \leq (T + W - 1)$ **do**
                **if** $t = 1$ **then**
                    $a \leftarrow \arg\max\{V_a^{\{n\}}(\text{postDecision}(s^{\{t\}}, a)) + 2 * SE(\text{postDecision}(s^{\{t\}}, a))\}$
15:             **else**
                    $a \leftarrow \arg\max\{V_a^{\{n\}}(\text{postDecision}(s^{\{t\}}, a))\}$
                $s_a^{\{t\}} \leftarrow \text{postDecision}(s, a)$
                $s^{\{t\}} \leftarrow \text{preDecision}(s_a^{\{t\}})$
                $r^{\{t\}} \leftarrow R(s^{\{t\}})$
20:             $t \leftarrow t + 1$
            $m \leftarrow m + 1$
        $OnPolicyDist \leftarrow OnPolicyDist$ plus all $s^{\{1\ldots(T+W-1)\}}$
        set $v^{\{1\ldots W\}} \leftarrow 0$
        set $t \leftarrow (T + W - 1)$
25:     **while** $t \geq 1$ **do**
            $w \leftarrow W$
            **while** $w \geq 1$ **do**
                $v^{(w)} \leftarrow v^{\{w\}} + \gamma^{(t-1)} r^{\{t-w\}}$
                $w \leftarrow w - 1$
30:         $t \leftarrow t - 1$
        $w \leftarrow 1$
        **while** $w \leq W$ **do**
            $OBS \leftarrow OBS$ smoothed with $(s_a^{\{w\}}, v^{\{w\}})$ using BAKF
            $SE \leftarrow SE$ from BAKF
35:         $w \leftarrow w + 1$
        $V_a^{\{n+1\}} \leftarrow$ low-bias function approximation using $OBS$
        $n \leftarrow n + 1$
```

Unfortunately, this change can also reduce the learning efficiency since $|S| \times |A|$ function values need to be learned instead of just $|S|$. With this in mind Powell recommends the post-decision state in [23]. The idea of the post-decision state is simple, assume there is some state that exists after an action is selected, but before transitioning to the new state, and learn the value of this state.

To think in terms of Q-learning, the post-decision state is the "pseudo-state" of (s, a). That is, the state where one is in state s and has selected to take action a, but nothing has happened yet. In some MDP models this is the best one can do. In others though, the structure of the problem allows for a post-decision state with a lower dimension than the Q-learning approach, leading to a much more efficient use of information while still having the benefits of Q-learning.

In algorithm 2 the post-decision state (and its associated value) is differentiated from the traditional pre-decision state with subscripts (i.e., s_a and V_a). With this notation the Optimality equation becomes:

$$\begin{aligned} V_a^*(s_a) &= E\left[V^*(s')|P_{s,a}\right] \\ V^*(s) &= R(s) + \gamma \max_{a' \in A}\{V_a^*({s'}_{a'})\} \end{aligned}$$

This approach has one further utility worth noting. By isolating the expectation it becomes possible to estimate the value of V_a via repeated sampling. This opens the door to simulation based approaches (e.g., line 18) with smoothing of observations (e.g. line 33) instead of fully realizing the MDP's P, an impossibility for the experiment's MDP.

Finally, it should be noted, the same efficiency gain can still be achieved with the Q-function approach so long as one is using basis functions for approximation. In this case one simply needs to define the basis functions for a state action pair as the post-decision state basii. When one isn't using basis functions, (e.g., a lookup table algorithm), then only the post-decision state formulation still achieves greater efficiency in data.

Unbiased Value Observations

A common practice in the RL literature is to use previous estimates of a value function to update a new estimate of that value function. Sutton and Barto refer to this practice as bootstrapping [28]. In its simplest form this can be seen with the following iterative update:

$$V_i(s_t) = R(s_t) + \gamma \max_{a \in A} \left\{ \mathbb{E}\left[V_{i-1}(s_{t+1}) | P_{s,a} \right] \right\} \tag{4.11}$$

Using this update form, the bootstrap estimates (i.e., V_{i-1}) introduce bias into the new estimates (i.e., V_i). This occurs because the value of V is non-stationary. Under certain conditions it can be proven that V will still converge to its true value despite this bias (c.f., value iteration [23, 28]).

In general though, this cannot be proven, especially with approximate methods. Even so, bootstrapping still seems to be quite common, and some experimentalists report that it often works better than alternative methods [28].

One indisputable advantage that bootstrapping offers is speed. The above formulation (4.11) would only require a single time step into the future, s_{t+1} before an update can occur for a five-step value function. An unbiased estimate of the same function would require all five steps:

$$V_i(s_t) = R(s_t) + \gamma R(s_{t+1}) + \gamma^2 R(s_{t+2}) + ... + \gamma^5 R(s_{t+5})$$

In KLA, despite the speed gains and anecdotal evidence, bootstrap methods seem to always perform worse than non-bootstrapped methods in the tests done as a part of this research.

We hypothesize that this is due to the low-bias value function having too much sensitivity to observational bias.

A small change the KLA introduces to counteract the extra processing time of unbiased observations is parameter W. This parameter determines how many steps after T to take. Each step after T can be used to calculate an additional T step unbiased estimate.

Exploration vs Exploitation

One area where KLA doesn't deviate substantially from existing work is in exploration vs. exploitation. KLA uses an off-policy approach to select which actions to evaluate. This can be seen on line 13. To choose the first action, an estimate of the standard error (SE) produces an upper confidence bound. (The standard error is naturally calculated as part of BAKF smoothing on line 33 and can thus be re-used).

In addition, evaluation states are sampled according to an approximation of the on-policy distribution (lines 11 and 22). Because updates to the value function can alter the on-policy distribution over time, this approximation is re-initialized every 4 iterations (line 6).

Value Function Updates

A final challenge, common to approximate RL algorithms, is how much to update the value function approximation each iteration. This decision is often more clear when using a lookup table algorithm (e.g., α in $V(s) = (1-\alpha)V(s) + \alpha\hat{v}$), but it is still present in function approximation methods as well.

KLA uses Powell's Bias Adjust Kalman Filter (BAKF) [23] to determine α. BAKF tracks the standard error and bias for all current estimates to determine a confidence level. When there is high confidence in the current estimate α is reduced so the current estimate has more weight than the new observation. When there is low confidence α is increased to favor the new observation.

BAKF (and by extension KLA) maintains separate estimates for each distinct representation of basii functions. In this way, the basii functions form a pseudo state space and their estimates are updated using methods similar to table lookup algorithms.

On each KLA iteration, BAKF smoothing results in a single estimate for every unique set of basii functions that has been observed thus far. These pairs of basii function representations and their estimates are then used to fit the value function approximation on each iteration.

4.2.2 KLA Performance Comparisons

To benchmark the KLA algorithm's performance, it is compared to LSPI and KLSPI. The MDP model used in these tests is the experimental game model detailed in 6.1. This model has $|S| \approx 10^{89}$.

In tuning LSPI, a number of feature transformations are explored with LSPI. The best performance for LSPI is achieved with a full second order polynomial. An RBF transform, an RBF transform plus full second order polynomial and a full third order polynomial did not perform as well as the second order polynomial.

In order to tune KLSPI a number of performance enhancements were made to the original algorithm to improve run time. The ALD threshold parameter is set optimally via a number of tests as well as setting optimal sizes for the sample observations.

For the first test (figure 4.4) 230 random rewards are generated. The initial state distribution is set to three randomly selected states. With only three states in the initial state distribution, these algorithms perform in some ways like a priority sweep.

For the second test (figure 4.4) 172 random rewards are generated. The initial state distribution is set to 20 randomly selected states. Notice that in this second test LSPI outperforms KLSPI. With a larger initial state distribution, KLSPI seems to have too much variance to fit its approximate value function well.

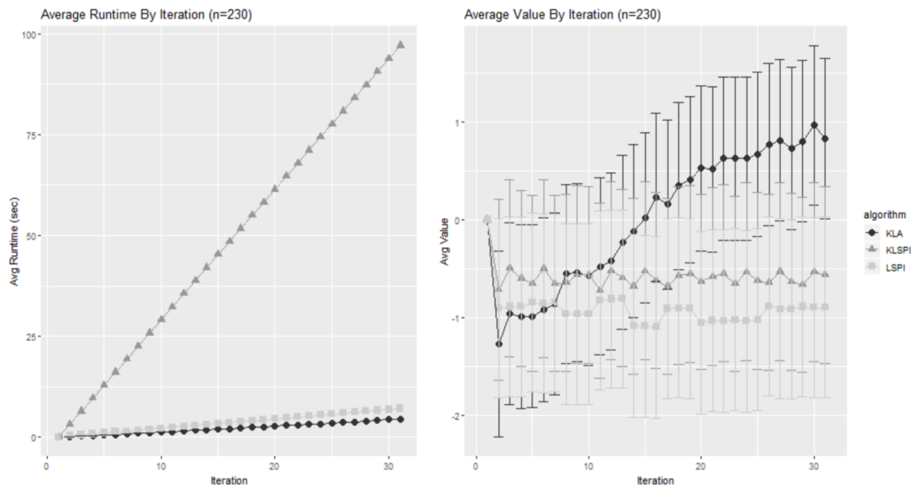


Figure 4.4: Performance comparison of KLSPI, LSPI and KLA with $|D| = 3$.

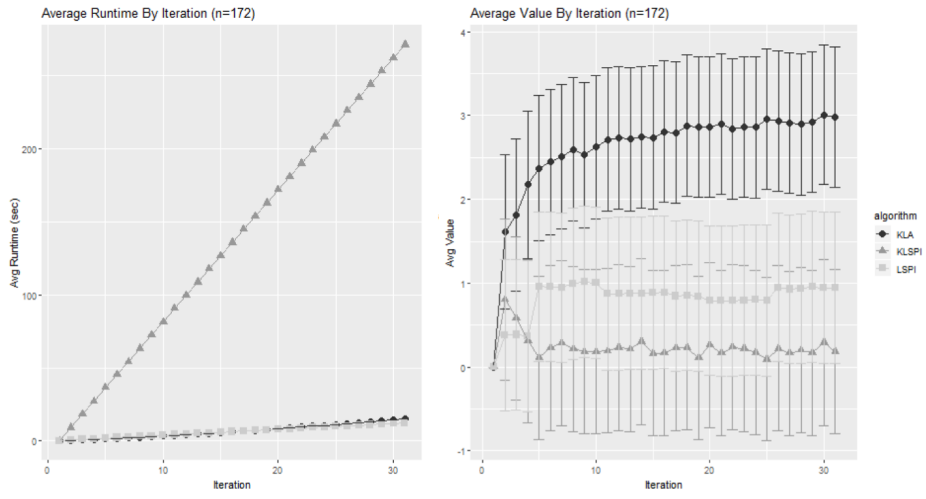


Figure 4.5: Performance comparison of KLSPI, LSPI and KLA with $|D| = 20$.

Finally, the last test (figure 4.6) is the same as the second test, but the KLA algorithm iterates 100 times instead of 30. This allows a more in-depth exploration of KLA's convergence point. Unfortunately, without ground truth there is no way to know if KLA has found a global or local optimal.

What makes these three tests most interesting is that all three are using the exact same MDP and value basii. Under these conditions KLA clearly outperforms. However, the original use case for LSPI and KLSPI was one where the reward is known. Knowing the reward makes it possible to design better value basii, and in these use cases they may outperform KLA.

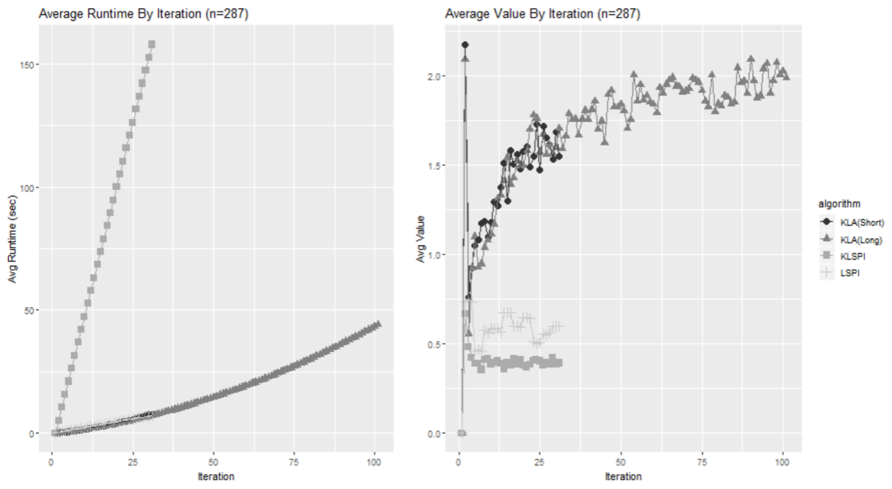


Figure 4.6: Performance comparison of KLSPI, LSPI, KLA for 30 and 100 iterations with $|D| = 20$.

Finally, because it seemes like LSPI and KLSPI get stuck in a local optimal fairly early in their approximations, efforts were made to encourage them to search more. None of these efforts improved their performance in any way. In fact, many times it made it worse.

Experiment

5.1 Hypothesis

The experiment tests two hypothesis are tested using a one-tailed Wilcoxon rank-sum test. The Wilcoxon test is selected over Welch's t-test due to deviations from normality in the dependent variable.

H1 Study participants who are shown a reward function learned from a high performer will perform better than a similar control group with a significance of $\alpha = .05$

H2 Study participants who are shown a reward function learned from a low performer will perform worse than a similar control group with a significance of $\alpha = .05$

$$\begin{array}{llll} N & O & & O \\ N & O & X_1 & O \\ N & O & X_2 & O \\ N & O & X_3 & O \\ N & O & X_4 & O \end{array}$$

Figure 5.1: The experimental design for the control and treatment groups

5.2 Design

A quasi-experimental design with eight blocking variables, four treatments and a control is used. For each group, two observations are made of the dependent variable ("performance"), once before and once after treatment (design shown in figure 5.1).

5.3 Dependent Variable

To measure "performance" (the dependent variable) participants play a game where they are awarded points for accomplishing goals. Participant's are told to earn as many points as they can within a time limit.

In the final analysis, performance is measured by the number of goals accomplished rather than the number of points earned.

Participants play the game twice to measure changes in performance. If the treatment is effective, participants in the treatment group should accomplish more goals (or fewer respective of the treatment) than the control group on the second play.

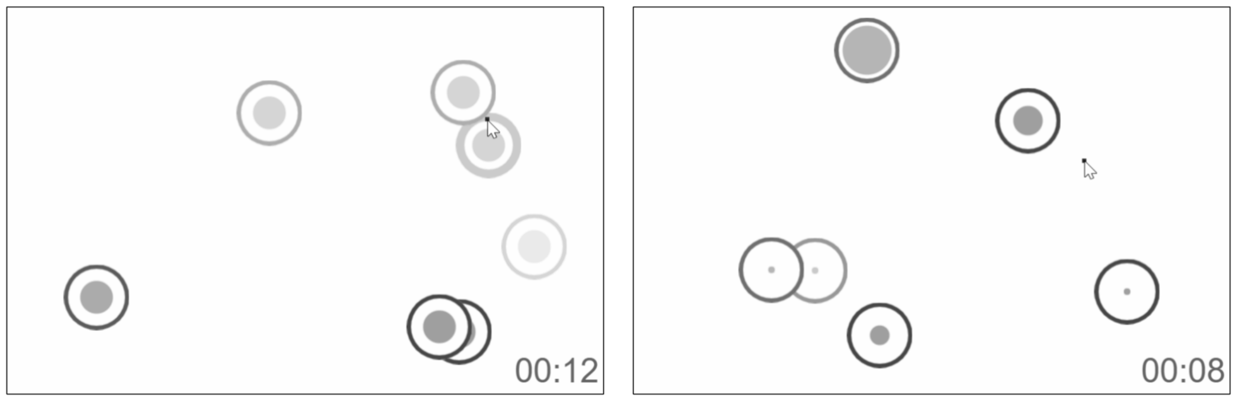


Figure 5.2: The left image shows a pre-treatment game in progress. The right image shows a post-treatment game. The colors you see here are accurate. The game was entirely in grayscale.

5.4 Game

The game is a continuous, stochastic pointing game (see figure 5.2). During the game targets appear randomly within the field of play. To earn points a player must simply pass their cursor over the top of a target. For all groups, targets are always the same size and always remain on screen for exactly one second before disappearing.

5.4.1 Stochasticity

Targets in the game appear according to a Poisson process with $\lambda = \frac{1}{200}$ targets per millisecond. This means that at any given time (after one second has passed) there will be an average of 5 targets on the screen (figure 6.3 left) with an average of 75 appearing over the course of the entire game (figure 6.3 right).

The stochastic engine that drives the game is the Math.random JavaScript API. For all major web browsers this is a pseudo-random number generator. This means there are no restrictions on the distribution of the random numbers — so long as they are uniform between 0 and 1. Most importantly, this means that tight clusters of targets can occur, increasing variance in the dependent variable.

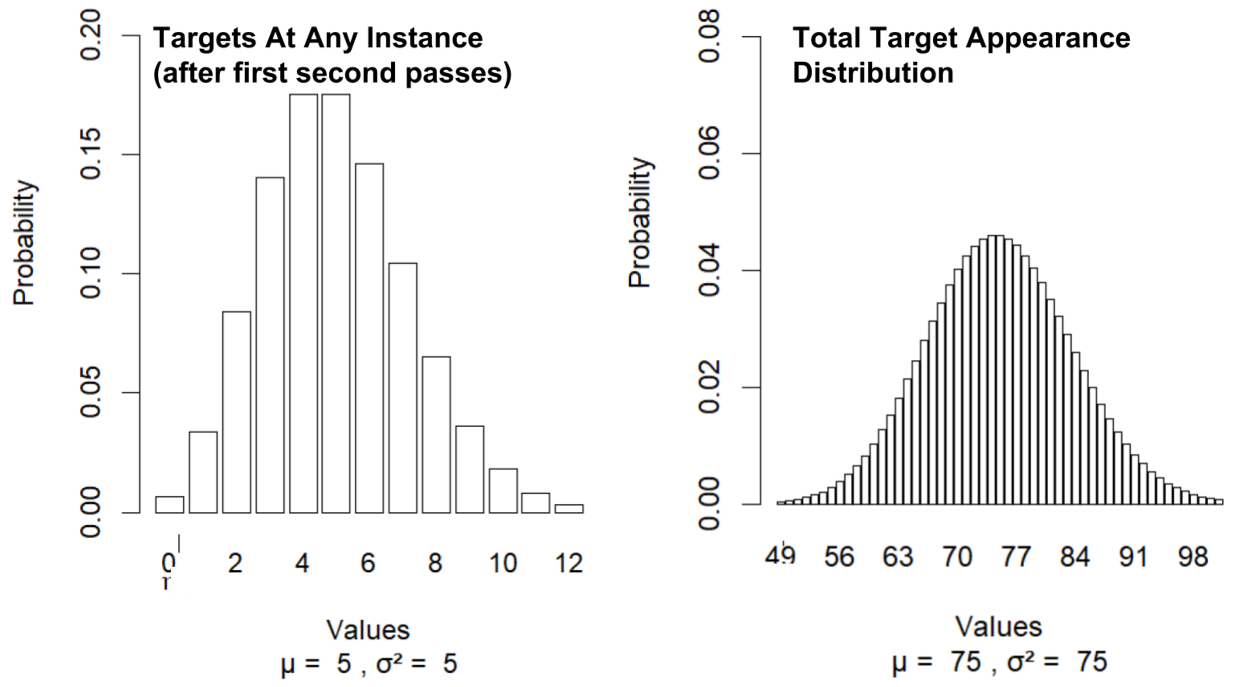


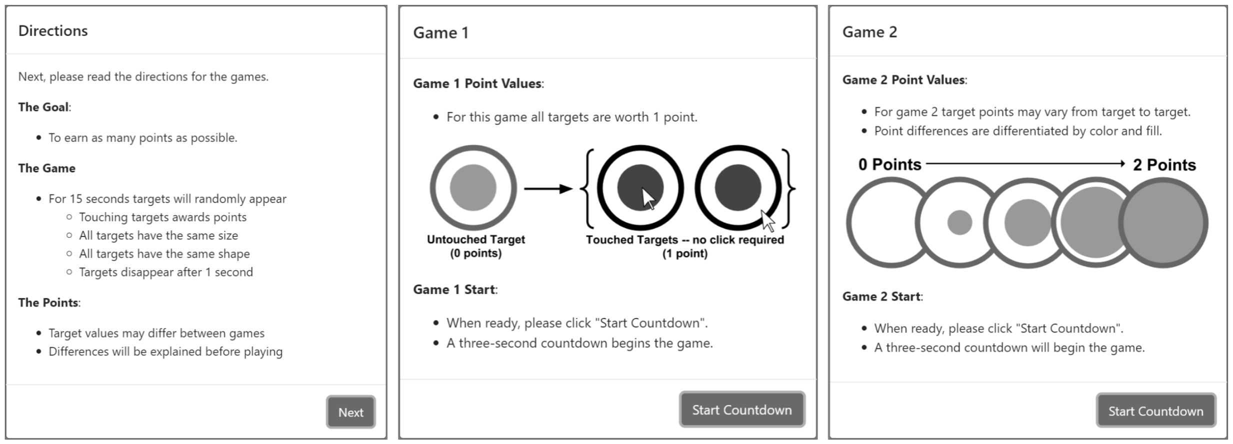
Figure 5.3: The distribution of targets at an instant in time (left) and over the course of the entire game (right)

5.4.2 Framing

When describing the task to participants the language of "games" and "points" is used (rather than task and goals) in an effort to encourage participant engagement and align mental models with the idea of rewards.

5.4.3 Directions

All participants are given identical directions: regardless of their group. These directions are shown in figure 5.4. The direction screens are displayed to participants one at a time from left to right (with each point screen showing just before its respective game begins).


Directions

Next, please read the directions for the games.

The Goal:

- To earn as many points as possible.

The Game

- For 15 seconds targets will randomly appear
  - Touching targets awards points
  - All targets have the same size
  - All targets have the same shape
  - Targets disappear after 1 second

The Points:

- Target values may differ between games
- Differences will be explained before playing

Next

Game 1

Game 1 Point Values:

- For this game all targets are worth 1 point.

Untouched Target (0 points)
Touched Targets -- no click required (1 point)

Game 1 Start:

- When ready, please click "Start Countdown".
- A three-second countdown begins the game.

Start Countdown

Game 2

Game 2 Point Values:

- For game 2 target points may vary from target to target.
- Point differences are differentiated by color and fill.

Game 2 Start:

- When ready, please click "Start Countdown".
- A three-second countdown will begin the game.

Start Countdown

Figure 5.4: The three direction screens. These screens are identical for all participants regardless if their control/treatment group.

5.5 Treatment

Participants within treatment groups are shown a reward function learned from two selected participants in the control group (i.e., “experts” in IRL terms). The experts were selected based on their performance. One expert to represent a high performer (H1) and one to represent a low performer (H2).

5.5.1 Experts

The summary statistics for the two selected experts can be seen in table 5.1. Expert “EH” is the high performer with 64 and 58 touches, and expert “EL” is the low performer with 20 and 21 touches.

Table 5.1: Expert Statistics

ID	Gender	Age	Input	First	Game 1 Touches	Game 2 Touches
EH	Male	25-34	Mouse	Yes	64	58
EL	Male	25-34	Mouse	Yes	20	21

5.5.2 Rewards

For each selected expert two reward functions are learned. Two functions are learned because the IRL algorithm produces slightly different rewards each time it runs, and there is no dominant way to select a best reward. The feature expectations for the experts (c.f., equation 4.3) and the two learned rewards associated with each expert are given in table 5.2.

The learned rewards can be differentiated according to two criteria: behavior match and goal match. Rewards with a strong behavior match result in policies that move like the human expert (e.g. similar velocity or location preferences), but may or may not achieve equal goal performance (i.e., touching as many targets as the expert in a single play). Similarly, rewards with a high goal match may touch the same number of targets as the expert, but in a very different way.

Table 5.2: Feature expectations calculated from expert trajectories and their IRL generated reward functions. (R_{CT}: expectation learned by KLA for "true" reward; S_{EH}:"good" expert-trajectories; R_{HH}:reward one from μ_{HE}; R_{HL}:reward two from μ_{HE}; S_{EL}:"bad" expert-trajectories; R_{LH}:Reward one from μ_{LE}; R_{LL}:Reward two from μ_{LE}.)

	Behavior Features					Goal Features
Source	**X-Loc**	**Y-Loc**	**Velocity**	**Acceleration**	**Direction**	**Not Touching**
R_{CT}	0.0132	1.5261	1.3745	1.3078	-2.8490	16.9471
S_{EH}	2.0093	1.4241	1.1993	0.6614	-0.6871	22.4824
R_{HH}	2.6119	2.0059	1.3469	1.1690	1.9244	18.9064
R_{HL}	2.4042	1.8073	1.2613	0.6936	-1.1199	25.2415
S_{EL}	1.7452	1.1804	0.7979	0.4558	2.2178	24.8904
R_{LH}	2.6265	1.7592	1.3417	1.2178	-1.4283	19.4081
R_{LL}	1.7136	2.5636	1.1956	0.6215	-1.1634	25.5276

According to this differentiation we assign the following labels to our learned rewards:

- R_{HH} - reward generates behavior matching high performer with high goal achievement
- R_{HL} - reward generates behavior matching high performer with low goal achievement
- R_{LH} - reward generates behavior matching low performer with high goal achievement
- R_{LL} - reward generates behavior matching low performer with high goal achievement

In addition to these rewards the “true reward” (or “control reward”) is defined as R_{CT} (i.e., each target is worth one point, the same way performance is actually measured):

- R_{CT} - reward for control/true performance measurement (all targets worth 1 point)

5.5.3 Delivery

When a treatment is applied to a group the treatment reward function alters target point values in the second game. This alteration is displayed as a variable amount of fill for each target. The more filled in a target is the more points it is worth. An image of this alteration can be seen on the right side of figure 5.2 as well as a complete description of all game variations and their rewards in table 5.3.

Table 5.3: The five game variations, their rewards and the hypothesis they are used to test

ID	**Points**	**Description**	**Hypothesis**
G_{CT}	R_{CT}	Targets are worth 1 point each	Control Group
G_{HH}	R_{HH}	Targets are worth 0 - 2 points	(H1) Increased Performance
G_{HL}	R_{HL}	Targets are worth 0 - 2 points	(H1) Increased Performance
G_{LH}	R_{LH}	Targets are worth 0 - 2 points	(H2) Decreased Performance
G_{LL}	R_{LL}	Targets are worth 0 - 2 points	(H2) Decreased Performance

Using the notation in table 5.3, figure 5.1 is updated to the more meaningful figure 5.5.

$$\begin{array}{lll} N & G_{CT} & G_{CT} \\ N & G_{CT} & G_{HH} \\ N & G_{CT} & G_{HL} \\ N & G_{CT} & G_{LH} \\ N & G_{CT} & G_{LL} \end{array}$$

Figure 5.5: The experimental design in terms of game variations

5.6 Analysis

Analysis begins with an abbreviated summary of important blocking variables. (The complete list can be found in the appendix in table A.1 along with the summary statistics).

Table 5.4: Statistics for all trials by treatment

Variable	**Level**	**CT**	**HH**	**HL**	**LH**	**LL**	**Total**
First	Y	324	307	344	359	432	1766
	N	66	77	53	82	117	395
Gender	M	213	180	212	224	257	1086
	F	177	204	184	216	291	1072
Age	18-24	44	57	55	50	85	291
	25-34	198	168	173	238	259	1036
	35-44	92	81	92	89	118	472
	45-54	33	48	42	36	56	215
	55+	23	30	35	28	31	147
Machine	Desktop	138	104	128	174	174	718
	Laptop	222	222	240	235	317	1236
	Tablet	7	16	7	6	12	48
	Smartphone	23	42	22	26	46	159
Input	Mouse	268	196	263	302	344	1373
	Touchpad	78	125	90	98	133	524
	Touchscreen	39	61	38	38	68	244
	Other	5	2	6	3	4	20

5.6.1 Blocking Variables

Among the blocking variables there is a high degree of variance in the "first" and "input" variables. These blocks are therefore broken out for separate analysis on their treatment effects.

It can be seen that when performance is separated on the "first" block, treatment effects tend to move in opposite directions (see figure 5.6). First time participants generally perform better on game 2 than game 1 while second time participants tend to get worse. Based on this we decide to simply look at first time players.

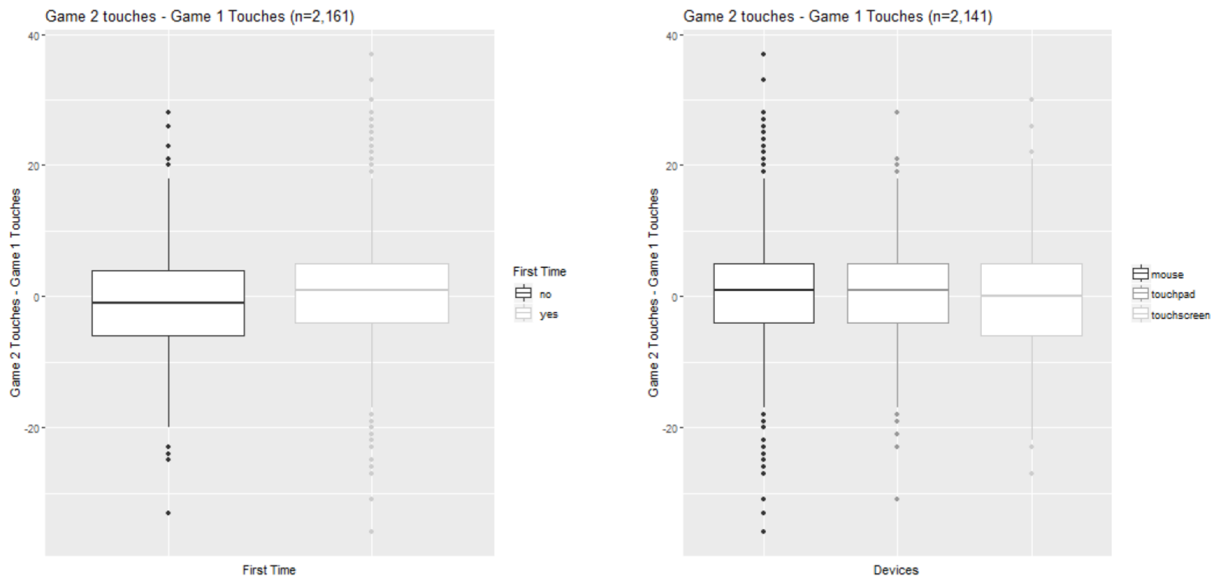


Figure 5.6: A Summary of effect size when blocking on first time and input device.

In terms of input device a similar pattern can be seen in figure 5.6. Individuals using either a mouse or touchpad tend to improve while individuals on touchscreens tend to get worse. Using a similar logic as above, touchscreen is excluded for the time being from analysis as well.

With just mice and touchpads the decision is made to block one more time. This is done because mouse data appears at this point to have more statistical power than the touchpad (i.e., a larger effect size and more observations), and the overall effect size already appears to be quite small.

The summary statistics for the selected block (i.e., first time participants using a mouse) is given in table 5.5 with the full summary in A.2.

5.6.2 Hypothesis Tests

With the block selected the normality assumption for each group is tested before performing a t-test (see figure 5.7). The groups appear to vary a moderate amount from normality at

Table 5.5: Statistics for all trials blocked on First=Y and Input=Mouse

Stat	Level	CT	HH	HL	LH	LL	Total
Gender	M	133	80	132	140	138	623
	F	89	76	92	96	128	481
Age	18-24	23	18	29	21	29	120
	25-34	117	59	102	131	122	531
	35-44	47	39	49	44	61	240
	45-54	18	24	29	21	34	126
	55+	17	16	15	20	20	88
Game 1	0-10	7	6	6	10	10	39
	10-20	29	15	29	41	40	154
	20-30	92	51	73	72	98	386
	30-40	44	38	57	54	66	259
	40-50	22	18	29	28	32	129
	50+	28	28	30	32	20	138
Game 2	0-10	4	3	3	11	9	30
	10-20	30	16	23	30	44	143
	20-30	80	50	78	76	108	392
	30-40	52	38	56	54	52	252
	40-50	25	19	32	31	27	134
	50+	31	30	32	35	26	154
Total	Count	222	156	224	237	266	1105

the tails. Therefore a Wilcoxon rank-sum test is performed since it doesn't assume normally distributed observations.

The two hypothesis are tested using the four treatments. Because each hypothesis has two treatments the Holm-Bonferonni adjustment is made to α. After the adjustment the null hypothesis for H1 is rejected. This suggests that showing participants the reward of a high performer can increase performance over the control. Before we can say for sure though a alternative explanations are analayzed. The results for all tests are given in table 5.6 along with the associated Cohen's d.

One interesting phenomenon is R_{LH}, which far from decreasing performance as hypothesized actually increases performance from game 1 to game 2. This seems to suggest that matching goal features with a reward function (in this high case touches) is almost equally important

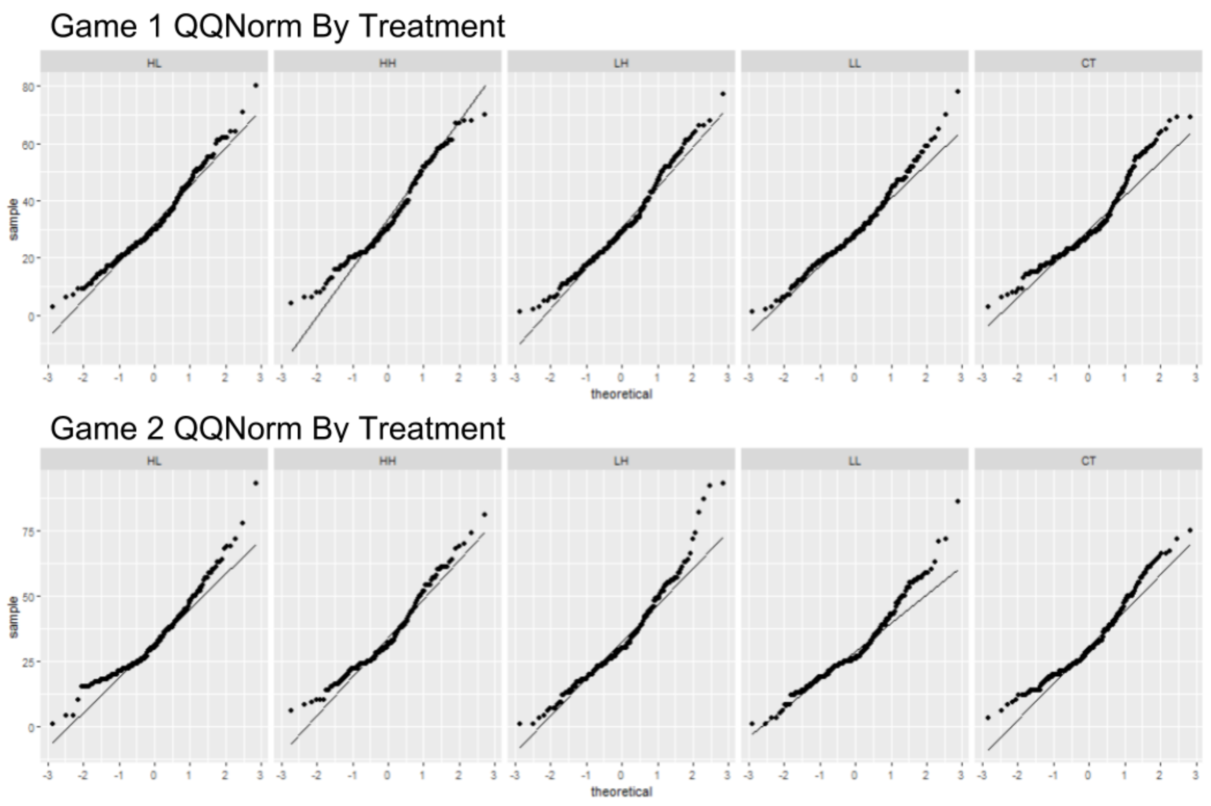


Figure 5.7: Q-Q plots against a normal distribution to test the normality assumption of t-tests.

to matching behavior (i.e. average velocity, location preferences, etc.). While matching both, as R_{HH} did, is best.

For all other input types, we fail to reject the null hypothesis. However, looking at the effects there do appear to be patterns. For touchpads, R_{HH} and R_{HL} do seem to increase performance more than any other reward though not enough to be significant. The number of touchpad observations though is substantially smaller than mice, so this isn't too surprising.

Table 5.6: Results of hypothesis tests for first time participants using a mouse

Hypothesis	**Reward**	**P-value**	Cohen's d
H1	R_{HH}	0.047	0.200
H1	R_{HL}	0.085	0.116
H2	R_{LH}	0.717	0.064
H2	R_{LL}	0.140	-0.158

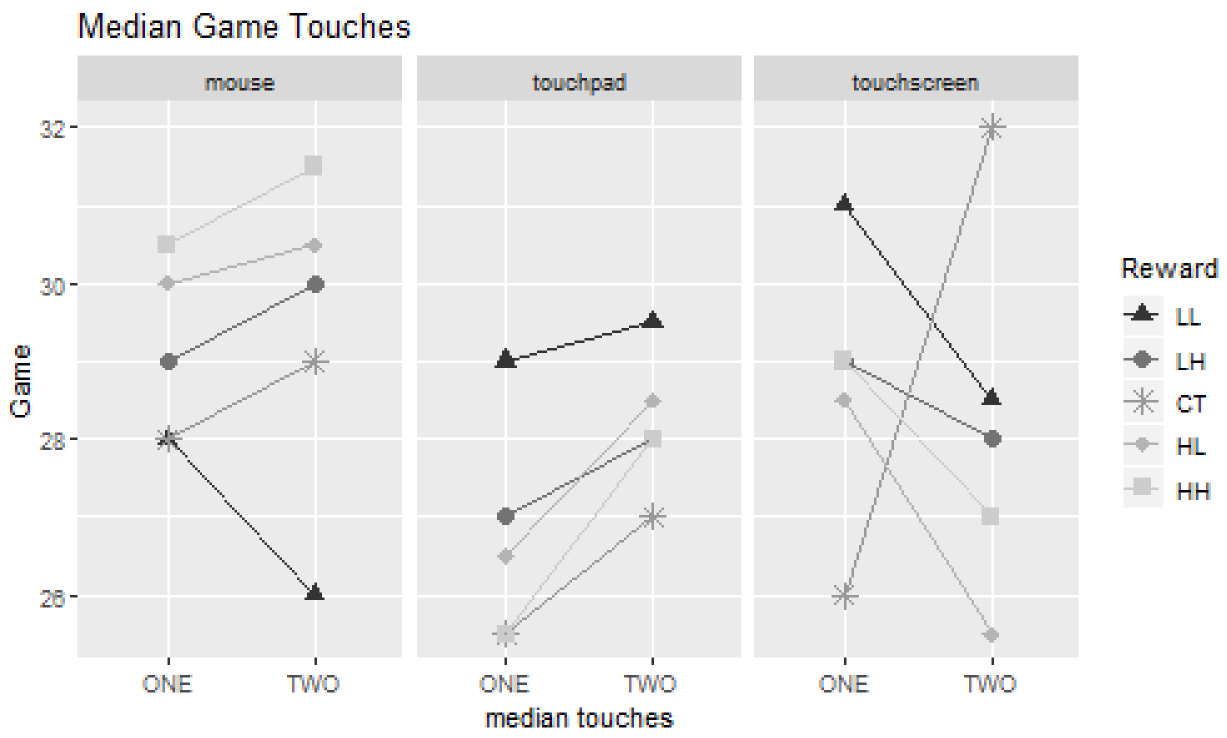


Figure 5.8: The change in median performance across all treatments.

For touchscreens, there is another interesting pattern. All rewards substantially decrease performance except the control (see figure 5.8). This seems to suggest that while targeted rewards can improve performance, these rewards don't generalize nearly as well as the true feedback signal.

5.6.3 Alternative Explanations

Finally, alternative explanations for the significant result are examined and can be ruled out.

The largest threat to validity, in this experiment, are between-group differences. The first test to check this threat is a Kruskal-Wallis test performed on all game 1 touch counts. The p-value for this test is 0.125 which means we fail to reject the null-hypothesis that the groups were different in pre-treatment performance.

The second check for differences in groups are in the blocking variables. The biggest difference seen here is the gender ratio (c.f. table 5.5). However, because this difference is seen in both high and low rewards (i.e., R_{HH} and R_{LL}), which had effects in opposite directions, this too is ruled out.

No other between group differences can be seen in the data leading us to make the claim that showing participants R_{HH} did, in fact, cause performance to increase over R_{CT} for individuals using a mouse and taking the test for the first time.

Application

To improve human performance using the method in this paper four criteria must be met:

1. The task of interest must be modelled as an MDP
2. Observations must be collected from human experts
3. A reward function must be learned from the observations
4. The learned reward function must be communicated meaningfully

While these criteria aren't easily satisfied neither are they, in our experience, overly restrictive. With careful thought and planning this approach should be applicable to many problems. This section provides more detail on the work to meet criteria 1 and 4.

6.1 MDP Model

As mentioned in the preliminaries, the traditional definition of an MDP is a five-tuple $<S, A, P, R, \gamma>$. The description of the MDP begins with the observational data and moves from there through the entire MDP definition. This approach is taken because many applications can begin with existing observations. These observations can then be leveraged to feed directly into the design.

6.1.1 Raw Observations

The observations for each participant are collected in two sets. The first set contains 450 observation for game 1 and the second set contains 450 observations for game 2 (i.e., 450 equals 30 observations a second for 15 seconds). Each observation contains the x, y coordinate of the cursor, the width and height of the browser window, the radius of the targets and, for each target on the screen, the target's x, y coordinates as well as its age in milliseconds. Because there could be a variable number of targets on the screen at any time each observation vector is of variable length.

6.1.2 State Space

From these observations the state is created. The model for the game defines a state as an observation vector plus the cursor's x, y coordinates from the three previous observations. Assuming a browser window size of 3000×1000 (a reasonable assumption), this gives $(3000 \times 1000)^4 = 8.1 * 10^{24}$ states from cursor movements alone. Adding in the potential number of targets (up to 12, see distributions above for more detail), their ages $\binom{1000}{12}$ and their locations $\binom{3000\times1000}{12}$, the state space of the game's model was approximately 10^{89}.

6.1.3 State Features

With the state space defined state features are selected. The features for the game model are defined as cursor $x, y, \dot{x}, \dot{y}, \ddot{x}, \ddot{y}, \dddot{x}, \dddot{y}$ plus browser width, height and target x, y and age for each target on screen. The only change from states to features are the derivatives. This feature representation is selected because it is easier to map from this representation to the basii functions used in the reward and value function approximations (i.e., equation 4.1 for KPIRL and s_a for KLA).

6.1.4 Expert Trajectories

Next the expert trajectories S_E are determined. To create S_E the 450 observations from game one are used as a starting point. From these observations the first and last 30 are removed (in order to reduce potential noise). Finally, a sliding window of size 10 with a step size of 1 splits the remaining observations into 380 expert trajectories. This gives $M = 380$, $T = 10$ and $|S_E| = 380 \times 10$. (Other methods to turn a single long observation into several trajectories were tested, but none seemed to work as well as this method).

6.1.5 Action Space

The action space, A, is where approximations are first introduced into the model. In terms of the game task, an action is simply moving the cursor to a specific location. The MDP model only has between 100 to 400 actions available at any given state (this is respective of how close the current state is to the edge). Compare this to the approximately 500×500 pixels a participant could realistically move to in a single time step. This means the choice of only having $20 \times 20 = 400$ actions is a considerable simplification. Still, this is sufficient

6.1.6 Transition Probabilities

In the MDP model, once an action is taken, two transitions occur. The first transition is the the post-decision state transition. This transition contains all the deterministic changes to the state. In the game's MDP model this includes cursor features $(x, y, \dot{x}, \dot{y}, \ddot{x}, \ddot{y}, \dddot{x}, \dddot{y})$, target age and if a target is removed from the state (i.e. it becomes older than 1 second). The second transition, the pre-decision state transition, includes all the stochastic elements of the state transition. In this model this is simply if a targets appear, and if so, where it appears.

6.1.7 Reward and Value Basii Functions

The final components to define, which are only necessary in order to use the KPIRL and KLA algorithms, are the basii functions for the reward and value functions. There is nothing that says these two sets must have the same basii.

The reward and value basii are defined as follows (where $T(s) = \mathbb{1}_{\text{a target is touched in } s}$):

$$\phi_R(s) = \begin{bmatrix} T(s) * X\text{-}Location(s) \\ T(s) * Y\text{-}Location(s) \\ T(s) * Velocity\text{-}Magnitude(s) \\ T(s) * Velocity\text{-}Direction(s) \\ T(s) * Acceleration\text{-}Magnitude(s) \\ |T(s) - 1| \end{bmatrix} \qquad \phi_V(s) = \begin{bmatrix} X\text{-}Location\text{-}Level(s) \\ Y\text{-}Location\text{-}Level(s) \\ X\text{-}Velocity\text{-}Level(s) \\ Y\text{-}Velocity\text{-}Level(s) \\ X\text{-}Acceleration\text{-}Level(s) \\ Y\text{-}Acceleration\text{-}Level(s) \\ Touch\text{-}Count(s) \\ Closer\text{-}Count(s) \end{bmatrix}$$

6.1.8 Value Function Structure

Finally, there is one last structure included in the game model that concerns the non-linear transformations made to the IRL algorithm. One way to think of the Gaussian kernel, described in the KPIRL algorithm, is that it learns a reward in terms of basii similarities. Given that, it made sense for the following relationship to be true.

$$C = ||\phi_R(s_1) - \phi_R(s_2)||$$
$$\forall s_1 \in \{s | s \in S \text{ and } T(s) = 1\}$$
$$\forall s_2 \in \{s | s \in S \text{ and } T(s) = 0\}$$

Where C is an arbitrary constant. The simplest way to think of this is all states that are touching a target are equidistant from all states that aren't touching a target. Or, in other words, are equally similar/dissimilar.

6.2 Communicating the Reward Function

This section briefly details the thought process for designing reward feedback. This study, unfortunately, was unable to vary any of this design to test how each component may have contributed. What this section provides is primarily conjecture and an aid for future researches who may want to replicate this research.

The raw reward returned by the IRL function is quite uninterpretable. Figure 6.1 shows two histograms of the raw reward values for R_{LL} and R_{HH}. Notice the incredibly small differences between reward values (both have a range of about .03), the relatively even split between positive and negative values, and long tails in both.

6.2.1 Transform to Increase Understanding

An underlying assumption in IRL is that the learned reward function results in the desired policy only if the agent who follows it is perfect and rational. This of course isn't the case with human decision making, so a few transformations are made to the reward in order to increase the likelihood an individual will respond correctly and rationally.

The first transformation makes it more clear which targets are worth touching and which

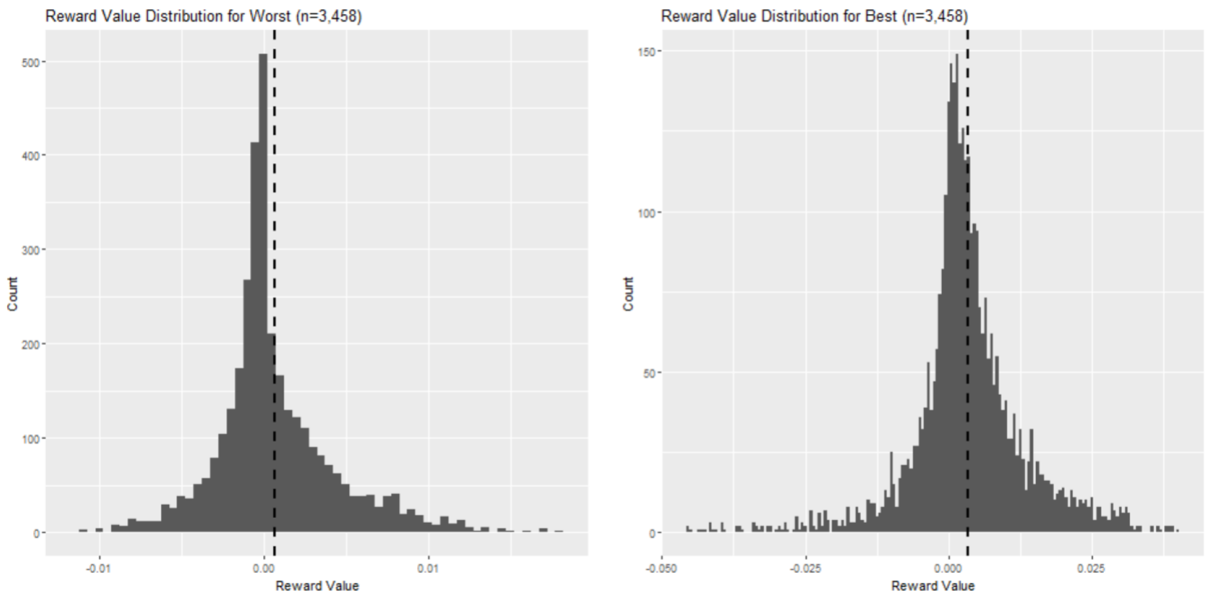


Figure 6.1: The distribution of reward values for a reward function learned from a person who performed very poorly, R_{LL}, (left) and very well, R_{HH}, (right) at the experimental task. It should be noted this is not a probabilistic distribution. The dashed line is the mean for each distribution.

aren't (i.e., encourage correct response to rewards). All states in the state space where a target isn't touched have the same reward value. This "no-touch" reward value is interpreted as a minimum threshold for action. That is, if any particular target is less rewarding than the no-touch state, then it isn't worth the effort to touch, since one would receive more reward for not touching.

In order to make this threshold-to-act more clear a translation is applied to the reward function to make the threshold equal to 0. With this transformation all targets not worth touching (according to the reward function) now have a negative reward value, while all targets worth touching (again, according to the reward function) have a positive reward value (see figure 6.2).

The next transformation encourages rational behavior. It has been shown that people exhibit an aversion to loss [29]. To avoid people performing poorly due to this aversion, all states with reward less than 0 are set to 0. Up to this point, the transformations have preserved

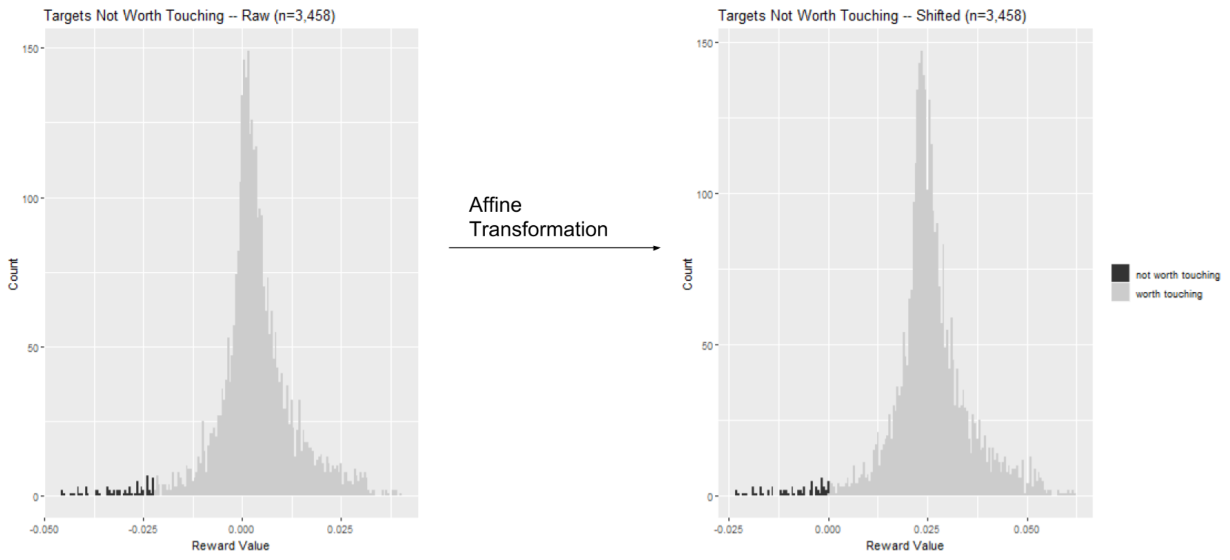


Figure 6.2: The distribution here is the one learned from the high performer (right side of figure 6.1). Notice the distribution is the same, the only change is the differentiation point is at 0 in the right figure.

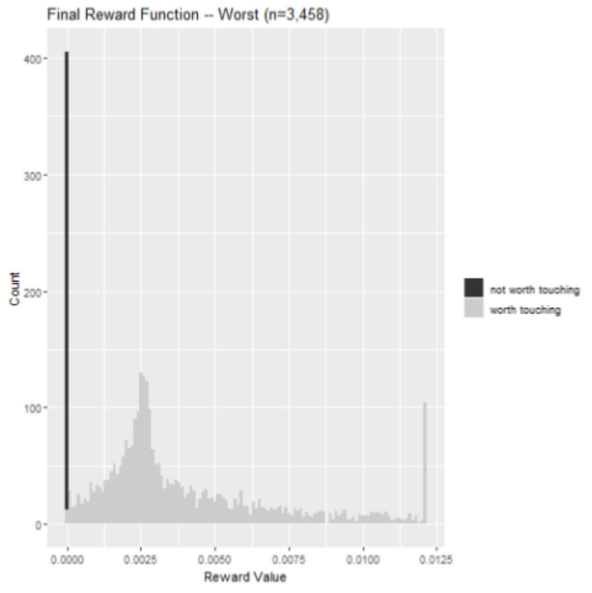
the optimal policy of the reward. With this transformation, the reward no longer generates the same behavior when followed by a computer.

The final transformation is to increase the information bandwidth. All reward values in the top 3 percentile are all set equal to the smallest value in the top 3 percentile. This reduces the overall range of the reward values leaving more bandwidth to differentiate between close rewards. The final reward distribution can be seen for the worst and best reward functions in figure 6.3.

6.2.2 Smooth to Improve Understanding

Because of the needs of computational tractability in such a large state space, the reward function is initially discrete. While, in general, states near one another tend to have similar reward values, this is not always the case. When drastic changes in reward value occur quickly it gives the feeling of randomness and lack of control.

For this reason smoothing techniques are used when showing reward values to participants.



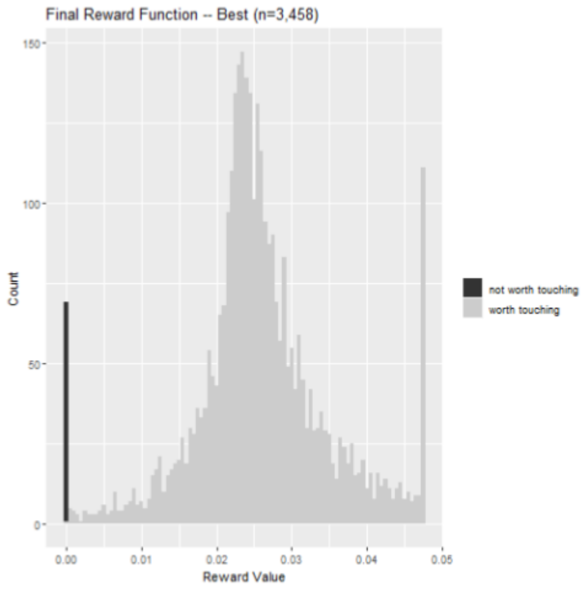


Figure 6.3: The final shape of the worst, LL, (left) and best, HH, (right) reward functions.

This has two advantages: one, the discrete reward function appears to change more continuously (feeling much more natural) and two, the reward values change more slowly, allowing time to learn which behaviors lead to high rewards and which to low rewards. Reward values are smoothed into the existing value according to the following formula (with $\hat{r}$ representing the instantaneous reward value at the time of the update):

$$R = (1-\alpha)R + \alpha\hat{r}, \quad \alpha = \frac{5}{18} \tag{6.1}$$

6.2.3 Forecast to improve understanding

The final design decision to communicate reward is to display the projected future reward rather than the reward according to the current cursor state. That is, the reward shown for a given target is as close as possible to the reward the person would receive if they chose to stop whatever they are doing and touch the target. To do this two things are done.

First, the third derivative is left out of reward calculations. The third derivative can change dramatically from time step to time step, making it much harder to project. This happens because it places more weight on fewer states which the participant doesn't have immediate control over (i.e.,$\dddot{x_4} = x_4 - 3x_3 + 3x_2 - x_1$).

Second, when calculating the projected $\hat{r}$ for a target, the direction component is calculated using the direction from the cursor to the target, regardless of the direction the cursor is traveling in. This is in direct contrast to how reward is calculated when a target is touched. When a target is touched reward is calculated according to the movement direction rather than location direction.

For the game model the approach was quite simple. For more complex, real world applications this would likely need to be more complicated, and without this component it would likely be much more difficult for people to know how their choices might impact their expected reward.

Conclusion

The observed effect size is small but statistically significant. The importance of this work, however, is not its effect size but in showing that rewards can be shaped and communicated to humans in such a way as to predictably alter.

Furthermore, this approach seems to have the potential for very general applicability to many human centered problems. This is demonstrated by how well it aligns with the research in other fields showing that environmental feedback can alter human performance for many different purposes (as referenced in the literature review).

Considerable work within the field of behavioral psychology has shown that relatively simple rewards (environmental feedback signals) can change behavior. Far less work has been done to examine how these rewards can be shaped to derive complex changes. This experiment, the author believes, begins to show how that path could be advanced.

In future work the author hopes to examine how this technique can be used to train a computer assistant for humans, as well as partner with an applied researcher in psychology to conduct a more rigorous analysis of the impact of this modelling technique on human behavior.

Appendices

Full Data Tables

Table A.1: Statistics for all trials by treatment

Stat	Level	CT	HH	HL	LH	LL	Total
First	Y/N	324/66	307/77	344/53	359/82	432/117	1766/395
Gender	M	213	180	212	224	257	1086
	F	177	204	184	216	291	1072
Age	18-24	44	57	55	50	85	291
	25-34	198	168	173	238	259	1036
	35-44	92	81	92	89	118	472
	45-54	33	48	42	36	56	215
	55+	23	30	35	28	31	147
Machine	Desktop	138	104	128	174	174	718
	Laptop	222	222	240	235	317	1236
	Other	30	58	29	32	58	207
Input	Mouse	268	196	263	302	344	1373
	Touchpad	78	125	90	98	133	524
	Touchscreen	39	61	38	38	68	244
	Other	5	2	6	3	4	20
Browser	Chrome	317	296	327	380	450	1770
	Firefox	38	45	42	34	47	206
	Safari	18	26	10	16	29	99
	Other	17	17	18	11	23	86
Game 1	0-10	8	12	9	14	16	59
	10-20	57	52	58	77	82	326
	20-30	152	130	141	142	181	746
	30-40	84	90	101	98	146	519
	40-50	44	55	46	60	68	273
	50+	45	45	42	50	56	238
Game 2	0-10	7	8	5	18	14	52
	10-20	49	58	59	58	83	307
	20-30	135	126	137	152	203	753
	30-40	98	95	98	110	120	521
	40-50	55	49	54	50	76	284
	50+	46	48	44	53	53	244

Table A.2: Statistics for all trials blocked on First=Y and Input=Mouse

Stat	**Level**	**CT**	**HH**	**HL**	**LH**	**LL**	**Total**
Gender	M	133	80	132	140	138	623
	F	89	76	92	96	128	481
Age	18-24	23	18	29	21	29	120
	25-34	117	59	102	131	122	531
	35-44	47	39	49	44	61	240
	45-54	18	24	29	21	34	126
	55+	17	16	15	20	20	88
Machine	Desktop	112	82	104	134	129	561
	Laptop	110	73	120	101	137	541
	Tablet	0	1	0	2	0	3
Browser	Chrome	186	124	191	209	225	935
	Firefox	23	25	21	19	31	119
	Safari	2	1	1	2	1	7
	Edge	4	4	6	2	5	21
	Other	7	2	5	5	4	23
Game 1	0-10	7	6	6	10	10	39
	10-20	29	15	29	41	40	154
	20-30	92	51	73	72	98	386
	30-40	44	38	57	54	66	259
	40-50	22	18	29	28	32	129
	50+	28	28	30	32	20	138
Game 2	0-10	4	3	3	11	9	30
	10-20	30	16	23	30	44	143
	20-30	80	50	78	76	108	392
	30-40	52	38	56	54	52	252
	40-50	25	19	32	31	27	134
	50+	31	30	32	35	26	154
Total	Count	222	156	224	237	266	1105

Bibliography

[1] Roland Sigrist, Georg Rauter, Robert Riener, and Peter Wolf. Augmented visual, auditory, haptic, and multimodal feedback in motor learning: a review. *Psychonomic bulletin & review*, 20(1):21–53, 2013.

[2] Gabriele Wulf and Charles H Shea. Principles derived from the study of simple skills do not generalize to complex skill learning. *Psychonomic bulletin & review*, 9(2):185–211, 2002.

[3] John W Krakauer and Pietro Mazzoni. Human sensorimotor learning: adaptation, skill, and beyond. *Current opinion in neurobiology*, 21(4):636–644, 2011.

[4] Kevin G Volpp, Leslie K John, Andrea B Troxel, Laurie Norton, Jennifer Fassbender, and George Loewenstein. Financial incentive–based approaches for weight loss: a randomized trial. *Jama*, 300(22):2631–2637, 2008.

[5] Kim Sutherland, Jon B Christianson, and Sheila Leatherman. Impact of targeted financial incentives on personal health behavior. *Medical Care Research and Review*, 65(6_suppl):36S–78S, 2008.

[6] Theresa M Marteau, Richard E Ashcroft, and Adam Oliver. Using financial incentives to achieve healthy behaviour. *Bmj*, 338:b1415, 2009.

[7] Gary S Becker. *The economic approach to human behavior*. University of Chicago press, 2013.

[8] Amos Tversky and Daniel Kahneman. Advances in prospect theory: Cumulative representation of uncertainty. *Journal of Risk and uncertainty*, 5(4):297–323, 1992.

[9] Herbert A Simon. A behavioral model of rational choice. *The quarterly journal of economics*, 69(1):99–118, 1955.

[10] John Scott. Rational choice theory. *Understanding contemporary society: Theories of the present*, 129, 2000.

[11] Gabriele Wulf, Charles H Shea, and Sabine Matschiner. Frequent feedback enhances complex motor skill learning. *Journal of motor behavior*, 30(2):180–192, 1998.

[12] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, pages 663–670, 2000.

[13] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in neural information processing systems*, pages 1–8, 2007.

[14] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

[15] Elnaz Nouri, Kallirroi Georgila, and David Traum. A cultural decision-making model for negotiation based on inverse reinforcement learning. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, 2012.

[16] Dylan Hadfield-Menell, Stuart J Russell, Pieter Abbeel, and Anca Dragan. Cooperative inverse reinforcement learning. In *Advances in neural information processing systems*, pages 3909–3917, 2016.

[17] Kamwoo Lee, Mark Rucker, William T Scherer, Peter A Beling, Matthew S Gerber, and Hyojung Kang. Agent-based model construction using inverse reinforcement learning. In *Simulation Conference (WSC), 2017 Winter*, pages 1264–1275. IEEE, 2017.

[18] Nobuhiro Yokoyama. Inference of aircraft intent via inverse optimal control including second-order optimality condition. In *AIAA Guidance, Navigation, and Control Conference*, page 1254, 2017.

[19] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

[20] Sergey Levine, Zoran Popovic, and Vladlen Koltun. Nonlinear inverse reinforcement learning with gaussian processes. In *Advances in Neural Information Processing Systems*, pages 19–27, 2011.

[21] Qifeng Qiao and Peter A Beling. Inverse reinforcement learning with gaussian process. In *American Control Conference (ACC), 2011*, pages 113–118. IEEE, 2011.

[22] Jaedeug Choi and Kee-Eung Kim. Bayesian nonparametric feature construction for inverse reinforcement learning. In *IJCAI*, pages 1287–1293, 2013.

[23] Warren B Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality, 2nd Edition.* John Wiley & Sons, 2011.

[24] Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine Learning Proceedings 1990*, pages 216–224. Elsevier, 1990.

[25] Michail G Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of machine learning research*, 4(Dec):1107–1149, 2003.

[26] Xin Xu, Dewen Hu, and Xicheng Lu. Kernel-based least squares policy iteration for reinforcement learning. *IEEE Transactions on Neural Networks*, 18(4):973–992, 2007.

[27] Jun Ma and Warren B Powell. Convergence analysis of kernel-based on-policy approximate policy iteration algorithms for markov decision processes with continuous, multidimensional states and actions. *Submitted to IEEE Transactions on Automatic Control*, 2010.

[28] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction.* MIT press, 1998.

[29] Amos Tversky and Daniel Kahneman. Loss aversion in riskless choice: A reference-dependent model. *The quarterly journal of economics*, 106(4):1039–1061, 1991.

[30] Frances K McSweeney and Eric S Murphy. *The Wiley Blackwell handbook of operant and classical conditioning.* John Wiley & Sons, 2014.

[31] Brian Ziebart, Anind Dey, and J Andrew Bagnell. Probabilistic pointing target prediction via inverse optimal control. In *Proceedings of the 2012 ACM international conference on Intelligent User Interfaces*, pages 1–10. ACM, 2012.

[32] Kamwoo Lee, Sinan Ulkuatam, Peter Beling, William Scherer, et al. Generating synthetic bitcoin transactions and predicting market price movement via inverse reinforcement learning and agent-based modeling. *Journal of Artificial Societies and Social Simulation*, 21(3):1–5, 2018.

[33] Abdeslam Boularias, Jens Kober, and Jan Peters. Relative entropy inverse reinforcement learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 182–189, 2011.

[34] Thomas D Cook, Donald Thomas Campbell, and William Shadish. *Experimental and quasi-experimental designs for generalized causal inference.* Houghton Mifflin Boston, 2002.

[35] Chi-Tsong Chen. *Linear system theory and design.* Oxford University Press, Inc., 1998.

[36] A Nedić and Dimitri P Bertsekas. Least squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems*, 13(1-2):79–110, 2003.

[37] Rosemary Garris, Robert Ahlers, and James E Driskell. Games, motivation, and learning: A research and practice model. *Simulation & gaming*, 33(4):441–467, 2002.

[38] Juho Hamari, Jonna Koivisto, and Harri Sarsa. Does gamification work?–a literature review of empirical studies on gamification. In *2014 47th Hawaii international conference on system sciences (HICSS)*, pages 3025–3034. IEEE, 2014.

[39] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.