Evaluating Techniques for Learning a Feedback Controller for Low-Cost Manipulators

Oliver M. Cliff*, Sildomar T, Monteiro
Australian Centre for Field Robotics
University of Sydney
NSW, 2006, Australia
*ocli5568@uni.sydney.edu.au

***Abstract*—Robust manipulation with tractability in unstructured environments is a prominent hurdle in robotics. Learning algorithms to control robotic arms have introduced elegant solutions to the complexities faced in such systems. A novel method of Reinforcement Learning (RL), Gaussian Process Dynamic Programming (GPDP), yields promissing results for closed-loop control of a low-cost manipulator however research surrounding most RL techniques lack breadth of comparable experiments into the viability of particular learning techniques on equivalent environments. We introduce several model-based learning agents as mechanisms to control a noisy, low-cost robotic system. The agents were tested in a simulated domain for learning closed-loop policies of a simple task with no prior information. Then, the fidelity of the simulations is confirmed by application of GPDP to a physical system.**

I. Introduction

The focus of this paper is to diverge from expensive manipulators and evaluate less precise, self-adaptive manipulator systems for simple tasks. Off-the-shelf robotic arms are not constrained by a standard, and as such there are kinematic and dynamic discrepancies in the manufacturing of each arm. We assess the viability of learning control, without any prior knowledge, of a noisy manipulator system.

The main contribution of this paper is the concise evaluation of common techniques used for learning manipulator control. In particular, we rehash the work presented in [1] and qualitatively compare two classes of reinforcement learning agents in the manipulator domain. In doing so, this paper presents the means for comparing several vastly different algorithms in a comparable manipulator domain. The environment is executed in simulation and the more efficient algorithm is then evaluated in a real, low-cost and inaccurate robotic system.

There has been a lot of recent investigation into robotic systems that are potentially efficient in the application of everyday tasks, however these algorithms segregate the guidance, navigation, and control components of a system, requiring integration of complex modules to yield a complete system. This approach typically involves expensive robotic hardware and robot specific dynamics [2]–[4]. Introducing learning agents to our system results in the inherent handling of both path planning and state-oriented control in a low-cost, generic robotic arm system.

Learning without any dynamics model a priori is a desirable attribute for portability across systems. Recent developments in Reinforcement Learning (RL) show progress toward effective control for complex systems [1], [5], [6]. However there is scant research on *comparable* manipulator environments across the board. RL is an approach of machine learning derived from behavioural psychology in that the agent learns from direct experimentation in experience-based, goal-directed learning methodologies. The objective of RL is to find a strategy which optimizes a long-term performance measure, a goal which can be acheived without any prior knowledge of a system. This results in RL being an ideal candidate for generalized control.

We propose that Gaussian Process Dynamic Programming (GPDP) [7] is an effective approach to manipulator control based on the agent's evaluated performance in the classic cart-pole RL environment ([8]) and contrast the response of the system against a similar environment for the conventional agents. To maximise the performance of applicable conventional RL agents, the base agents were overlayed with planning modules and decision trees.

The experiment domain is simple but perceptive: the goal of each episode is to move from an initial Cartesian co-ordinate to another attainable co-ordinate with no dynamics model or planned path known prior to each experiment. The experiments are performed for up to three degrees of freedom (DOF). Our results support the hypothesis that GPDP is far superior at learning closed-loop control in terms of reliability and efficiency.

To evaluate the applicability of any method for a particular purpose, unbiased experimentation must be undertaken. We assess the performance of a novel GPDP learning algorithm, PILCO [8], for the purposes of closed-loop control of a low-cost and noisy manipulator. The effectiveness of GPDP will be contrasted with more conventional RL algorithms: Q-learner [9]; SARSA [10]; DYNA [11]; R-MAX [12]; and TEXPLORE [13].

II. Background

This background information is intended to familiarize the reader with the nomenclature and subject matter covered in this paper. The motivation and implications of these algorithms are elaborated in section IV.

A. Reinforcement Learning

RL is characterized as learning by interaction; the *agent* makes decisions and interacts with the *environment*. To evaluate the efficacy of an *action* a_t, the environment indicates the reward r_t the agent receives as a function of the next-step state $\boldsymbol{x}_{t+1}$ of the environment. A task becomes episodic given a terminal state $\boldsymbol{x}^+$.

The crux of RL is the Markov Decision Process (MDP), the 4-tuple $\mathcal{M} \equiv (\mathcal{X}, \mathcal{A}, \mathcal{P}^a_{\boldsymbol{x}\boldsymbol{x}'}, \mathcal{R}^a_{\boldsymbol{x}\boldsymbol{x}'})$ of the non-empty set of states $\mathcal{X} \ni \boldsymbol{x}_t$; the set of possible actions $\mathcal{A} \ni a_t$; the reward probability kernel $\mathcal{R}^a_{\boldsymbol{x}\boldsymbol{x}'}$; and the state transition probability kernel $\mathcal{P}^a_{\boldsymbol{x}\boldsymbol{x}'}$.

Finally, the action-value transition $\hat{Q}$ designates the expectation of stochastic return given some initial conditions.

B. Conventional Reinforcement Learning Algorithms

We now consider the base agents used in finite MDP, where the set of states are a finite set of Cartesian coordinates $|\mathcal{X}| \in \mathbb{R}$. In a conventional setting, the discretised environment has a (tabular) database or decision model based on previous experiences.

1) Q-learning: Q-learning is a simple interpretation of model-free RL techniques, and in combination with SARSA provides an informative basis for PILCO against conventional model-free methods. The core of Q-Learning comes from value iteration, where the action-value prediction method stems from Temporal-Difference (TD) planning [9]. 1-step Q-learning uses an off-policy TD learning control method, where the learnt action-value function $\hat{Q}(\cdot, \cdot)$ directly approximates the optimal action-value $\hat{Q}^*(\cdot, \cdot)$ independent of the policy that is being followed, [9].

2) SARSA: 1-step SARSA is a simple application of policy iteration with an on-policy TD control method; updating the action-value function based on the policy being followed, [11]. The currently hreaded policy is updated online after each non-terminal state $\boldsymbol{x}$, in contrast to Q-learning, where the policy is updated offline.

3) Dyna: DYNA agents extend the typical RL architecture to form a world model by using planning steps in an online simulated model, [11]. These agents consider the limitation of RL that agents often rely heavily on a policy and return basis, rather than a causal model. As a result, less emphasis is placed on the scalar reward but on the *relative* state information $\delta\boldsymbol{x}$.

4) R-max: R-MAX is a model-based RL agent where is it expected that each unsure transition yields maximal reward; implicitly handling the dilemma of exploration versus exploitation. This level of uncertainty is proportional to the number of times the agent has visited a state [12]. The laborious technique of executing each state-action pair until a model is learnt is inefficient and often unapproachable in larger state spaces.

5) T-explore: TEXPLORE is a more recent method to avoid the time intensive trial-and-error approach to balancing exploration and exploitation in model-based algorithms, [13]. The agent assumes the probability of the state transition kernel is a product of each of its n state feature's probabilities, $\mathcal{P}^a_{\boldsymbol{x}\boldsymbol{x}'} = \prod_{i=0}^{n-1} \mathcal{P}^a_{\boldsymbol{x}_i \delta \boldsymbol{x}'}$. TEXPLORE agents recursively build a decision tree using the C4.5 algorithm, [14], rather than the typical tabulated transition, using (Shannon) entropy to spawn a *discrete* decision tree.

6) UCT: The model planning module UCT build on the model-based algorithms introduced above, [15]. The UCT method is a rollout-based algorithm in that it builds a look-ahead tree by continuously sampling episodes from the initial state. UCT uses an online model, where branches are added to the tree throughout an episode, allowing for continuous modelling.

7) M5 Regression Trees: The UCT algorithm employs continuous M5 regression trees in order to diverge from the rigidity of a discrete tree model into more flexible domains, enabling optimal behaviour convergence in larger state spaces [16]. The common approach in a small and finite MDP is keeping a table of observations and targets. The regression tree segments the continuous model into a linear-piecewise model (linear approximations of the continuous model) supporting continuous MDPs.

C. Gaussian Process Dynamic Programming

Gaussian processes (GPs) are a generalization of the Gaussian distribution and popular application of Bayesian inference, [17]. These processes are applicable to machine learning tasks by placing a prior over the function space. The GP combines flexible non-parametric modelling with inference techniques based on Bayesian data analysis, inherently handling the issue of the infinite function space to draw from in nonparametric function approximation methodologies. This data efficient process draws the near-optimal function from the mean and covariance input-space,

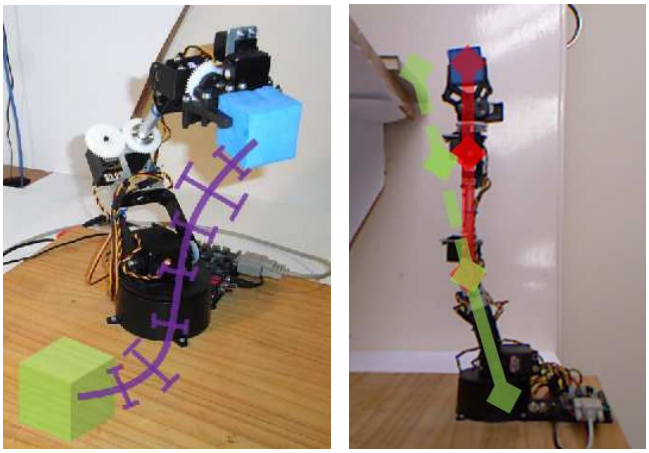
$$f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')). \tag{1}$$

GPDP is a value-function based RL algorithm that generalizes dynamic programming to continuous state and action spaces. The methodology is characterized by the use of Bayesian GP models to describe value functions in DP recursion, [18]. In this paper, we analyse the merits of the successor to [19], the Probabilistic Inference for Learning Control (PILCO) algorithm ([7]) that learns a probabilistic dynamics model and incorporates model uncertaintly into long-term planning.

PILCO is a model-based, offline learning agent capable of closed-loop control. As such, the algorithm learns a policy from cascading one-step predictions through the state space. Once a policy is learnt PILCO performs near-optimal closed-loop control given the current state $\boldsymbol{x}_t$.

III. Implementation

The process for experiments were pipelined; each agent was tested on a similar simulated platform and assessed for



(a) (b)

Fig. 1. The Lynxmotion AL5B manipulator. Figure 1(a) illustrates the simple but perceptive task used in these experiments, showing the probabilistic steps each agent takes to move the cube, blue, to the target state, green. In figure 1(b) an exaggeration of the method used for empirically selecting system noise. The arm's actual position (red) deviates from the expected (dashed green) extended length.

viability in a physical system, if applicable or necessary, the agent would be tested within the framework of the real system.

Motivated by [1], the domain for testing was a simple cartesian setup in three dimensional state space with *up to* a three dimensional control vector $\boldsymbol{u}$. In replicating [1], the cartesian state used is the centre of a block in the end-effector $\boldsymbol{x} = \{x_c, y_c, z_c\}^T$, see figure 1(a). As the control set is variable, the number of parameters, or joints, controlled is D. As a result, the action is mapped to control signal (PWM) outputs for the arm $a : \mathbb{R}^1 \rightarrow \mathbb{R}^D$. We extend [1] by incrementally increasing the number of enabled joints and introducing conventional agents and simulated environments for a more informed analysis of the PILCO agent.

The basic conventional agents and planner with regression tree were provided by the open-source TEXPLORE ROS package ([13]) and the PILCO framework source code provided access to PILCO's underlying algorithms.

A. Conventional Agent Architecture

The environment must map a scalar action in the action set $a_t \in \mathcal{A}$ to the control vector $|\boldsymbol{u}| = D \leq 3$. To do so, we use bitwise manipulation to extract each control u_k at a given base value. To test different avenues for optimal control, two modes of operation were experimented with: *incremental* and *unconstrained* motion for the manipulator's revolute joints. Unconstrationed motion allows each active servomotor to utilize the full range of motion to a specified resolution - the actions increase exponentially with each DOF. Incremental motion can only perturb each servo one increment either direction of remain still, thus the actions also increase exponentially, however orders of magnitude lower. After some trial and error, the base value was set to 16 as a conservative measure toward minimizing the action-space i.e. a 11.25^o increment for 180^o of motion. This results in $\mathcal{A} : \mathbb{R}^{16^D}$ for unconstrained motion and $\mathcal{A} : \mathbb{R}^{4^D}$ for incremental.

The basic conventional algorithms have proven effective at gaining optimality given the ability to explore the domain thoroughly. With the exception of TEXPLORE, the model-based algorithms use a tabular model of transition, exponentially increasing the memory and inefficiency for higher order MDP's. In manipulator control a higher order MDP enables higher precision of both goal state and motor precision and thus we wish to effectively maximise the MDP for our environment. The inclusion of UCT and M5 decision trees for continuous states were a means to smooth the discrete MDP algorithms.

B. Extension to larger MDPs

To improve on the underlying conventional agents and encourage learning in a higher order MDP, we use Multi-agent architecture and a Monte-Carlo style planner with continuous decision tree models.

[20] proved minimizing the available action-space is an effective way to overcome the 'curse of dimensionality' in a higher order MDP, in contrast to methodologies with concentration on state-space truncation. [21] implement multi-agent architecture to minimize the action-selection dilemma on a manipulator and in turn optimize the underlying Q-learning algorithm. The value function converges efficiently however these results are obtained through a noiseless simulation and are not compared to other algorithms, rendering them ineffective at ascertaining viability to real systems. The TEXPLORE ROS package supports centralized agent architecture, with one node interacting with the environment. We extend upon this to package to support decentralized control for agents, assigning one agent to each controllable motor of the manipulator. Distributing control to each joint minimizes the action-space, nullifying the exponents in both unconstrained and incremental motion.

Inspired by [13], the model-based algorithms are enhanced through the UCT planner with a flexible M5 regression tree for conventional agents. The motivation for rollout-based algorithms such as UCT have an advantage in our simplified manipulator domain where the set of successor states can concentrate to a small number of states for near optimal control, [15]. Including M5 regression trees over other decision tree algorithms, such as C5.4, promotes continuous valued transition models.

1) Terminating the episode: In our experiments, we use proximity to a target as a basis for terminating episodes. It was observed that the single agent results are fairly robust to the reward function if the episode was terminated based on the Euclidean distance d to the target state $\boldsymbol{x}^+$ where $d(\boldsymbol{x}) = |\boldsymbol{x}^T\boldsymbol{x}|$. However, [21] prove non-scalar reward functions are advantageous in multi-agent architectures. From [21] the continuous, saturating functions for both penalty r^- and terminal r^+ rewards are

$$r^-(\boldsymbol{x}) = -\alpha d^n, \qquad r^+(\boldsymbol{x}) = \frac{\beta}{1 + d^m}.$$

The experiments presented in this domain are episodic, meaning there is a desired goal (Cartesian coordinate) and the environment reinitializes to a starting condition where the agent maintains any previous state-action-reward experience. By using the conventional tools for learning control, such as storing tabular transitions and Monte-Carlo learning, the agents do not make an assumption of the environment akin to a continuous probabilistic function, as does GPDP. As a result, the conventional agent will not incline toward the completion of the task until multiple observations of transitions have occured. This defining feature of conventional RL requires a *state terminated* episode, where, in these experiments, the episode terminates when a state is reached within a proxied target state space.

C. Gaussian Process Dynamic Programming Architecture

The GP model learnt is the latent function $f : \mathbb{R}^{3+D} \rightarrow \mathbb{R}^3$ where the training inputs and respectively mapped targets are $(\boldsymbol{x}_t, \boldsymbol{a}_t) \rightarrow \Delta \equiv \boldsymbol{x}_{t+1} - \boldsymbol{x}_t + \epsilon_t, \epsilon_t \in \mathbb{R}^3$ and i.i.d Gaussian noise. GP enables multivariate regression, resulting in a higher dimensional action-space than conventional agents $\mathcal{A} \subseteq \mathbb{R}^D$. Each action a_k is then trivially mapped to the respective control signal $\boldsymbol{u}_k$.

From [1] we set the reward function to

$$r(\boldsymbol{x}) = \exp(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^+)\boldsymbol{T}(\boldsymbol{x} - \boldsymbol{x}^+))$$

where $\boldsymbol{T} = \boldsymbol{\sigma}_c^2 \boldsymbol{I}$ and the cost width $\boldsymbol{\sigma}_c = \{\frac{1}{2}b, b, 2b, 4b\}$ and b is the edge length of each block. The reward function is therefore a function of scale mixtures of squared exponentials to minimize reliance on one single weighting and yield non-zeros policy gradients even far from x^+.

Through function approximation *step terminated* episodes can effectively be used. GPDP can still cascade one-step predictions for parameter gradients, such as those experienced even relatively far away from $\boldsymbol{x}^+$ in an initial rollout. The result is that we are able to terminate the episode based on a time step horizon, as opposed to proximity based. This measure additionally enabled PILCO to perform offline learning for on-line control, ensuring minimal randomly seeded actions. Theoretically, allowing the system with the minimum time to reach a target will suffice to learn a control policy.

D. Simulation

To simulate the kinematics of the manipulator, the homogeneous transformation of the end-effector $\{3\}$ is performed relative to the base (shoulder) frame $\{1\}$,

$$\begin{bmatrix} {}^1\boldsymbol{x} \\ 1 \end{bmatrix} = {}^1_3\boldsymbol{T} \begin{bmatrix} {}^3\boldsymbol{x} \\ 1 \end{bmatrix} + \varepsilon \tag{2}$$

where ε is i.i.d Gaussian noise $\varepsilon : \mathbb{R}^3$. Given joint angles $\{\theta_1, \theta_2, \theta_3\}$ and limb lengths $\{L_1, L_2, L_3\}$ indexed from the shoulder joint,

$${}^1_3\boldsymbol{T} = \begin{bmatrix} c_{\theta_{123}} & -s_{\theta_{123}} & 0 & L_1 c_{\theta_1} + L_2 c_{\theta_{12}} + L_3 c_{\theta_{123}} \\ s_{\theta_{123}} & c_{\theta_{123}} & 0 & L_1 s_{\theta_1} + L_2 s_{\theta_{12}} + L_3 s_{\theta_{123}} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To accurately simulate responses, it was assumed that both the actuation and sensing system noise could be considered Gaussian. The simulated system noise is deduced by an empirical approach, assuming the fully extended arm (with the highest moment about the revolute shoulder joint) yields the maximum normal force to each joint, see figure 1(b). We then observe a conservative estimate of noise by perturbing the shoulder joint, keeping the upper arm rigid and recording disparaty of measurements from expected outcomes. The results were confirmed Gaussian, producing the empirical system noise as $\varepsilon \sim \mathcal{N}(0, 20^{-6})$.

E. Real System

The physical system is designed for use of the PILCO agent. The system consists of a local and remote machine, a Kinect sensor and a robotic arm.

Our system is a distributed network communicating through TCP/IP. The computationally intensive PILCO learning algorithm models the GP latent function through one-step predictions on a remote server. A machine local to the manipulator receives an updated policy and periodically transmits state transition information to improve the PILCO model.

The physical manipulator used is the Lynxmotion AL5B Robotic Arm, see figure 1(a), a low-cost and relatively inaccurate servomotor controlled manipulator with little feedback. We obtain the markov state through a blob tracker with streamed depth registered images from a Microsoft Kinect, tracking a foam block in the end-effector of the manipulator. For communication and open-source libraries we used ROS.

IV. Experiments

The experiments covered conventional agents in a simulated environment and PILCO agents in both a simulated and physical environment.

A. Conventional Agent Algorithms

The results in figure 2 were collated through experiments spanning 1000 episodes; each episode terminated within a pre-defined proximity kernel of the goal state with an upright, fully extended initial configuration. The designated goal state was a consistent but arbitrary position within the workspace of the manipulator. Each plot shows the typical accumulated reward per episode. Recall that higher reward over the episode is optimal, with a maximum reward of 0.

Figure 3 captures findings from the agents optimized for continuous MDPs. Only model-based agents are supported this framework with the exception of R-MAX, requiring a tabular environmental model. Further, the higher computational requirements of TEXPLORE (by considering each feature of the state) renderred the algorithm intractable in such a large MDP (although the lower order MDP results can be seen in table I). As a result, DYNA agent's are the only plausible agent to consider for the extension to a continuous state space through the UCT planner and continuous M5 regression trees.

Figure 2 highlights results from the conventional agents that act optimally in a finite MDP. In general, DYNA agents

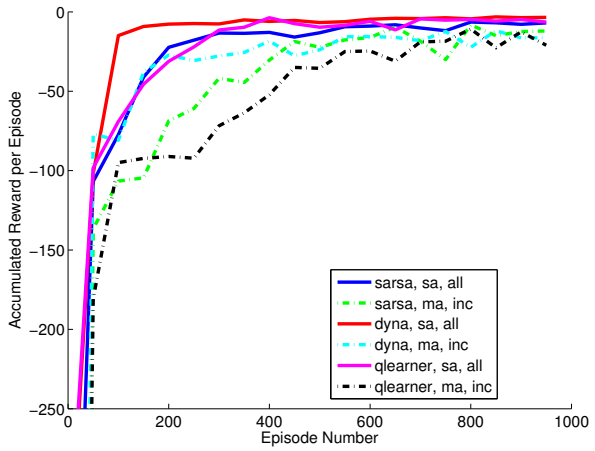


Fig. 2. Collated and averaged results for basic conventional agents (not optimized for continuous MDPs). The graph depicts the typical accumulated reward per episode over 1000 episodes. The legend describes: the agent e.g. DYNA or Q-learner; the agent network format e.g. multi-agent (ma) or single agent (sa); and the mode of operation e.g. incremental (inc) or unconstrained (all).

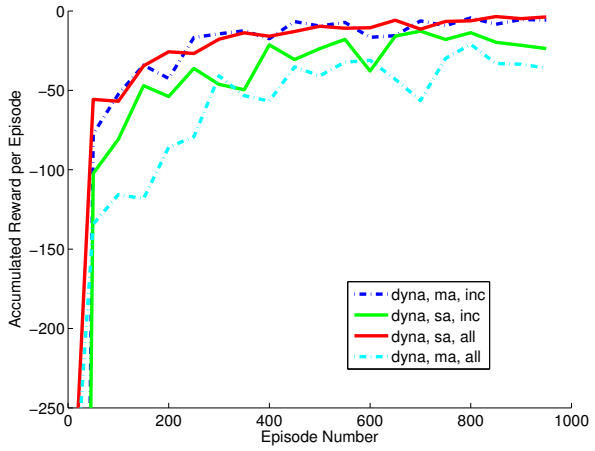


Fig. 3. Results for the Dyna agent optimized through the UCT planner and M5 decision tree as per figure 2.

outperform the SARSA and Q-learner agents for both unconstrained and incremental motion proving these model-based methods are sample efficient however they are also computationally intensive (as suggested by the omission of TEXPLORE and R-MAX). This is most likely due to the fact that DYNA agents are able to hypothesis any number of outcomes from any number of environmental models, as opposed to updating a static world-view based on previous observations.

Figure's 2 and 3 in conjunction with table I show that distributed architecture ousts single agent control when operating in incremental mode. This might be due to the significant reduction in action-space for incremental mode if it weren't also the case that unconstrained motion generally benefits from single-agent control. These two results seemingly contradict one another, as distributed, incremental architecture supports the minimization of action space whereas centralized, unconstrained architecture does not. More investigation

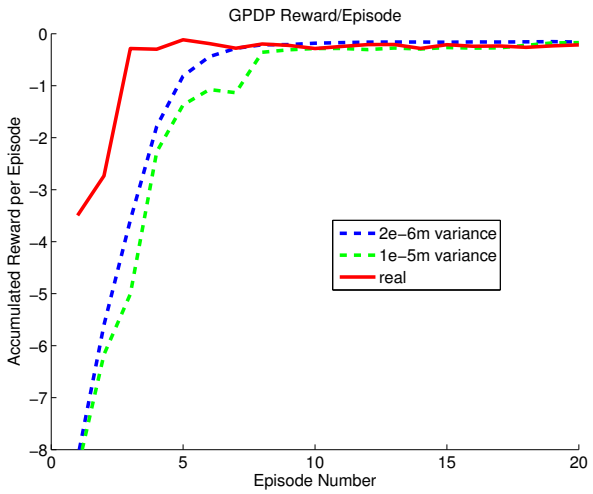


Fig. 4. Typical reward response of the Gaussian Process Dynamic Programming agent, PILCO, when applied to a physical system (red solid) and simulated systems with differing noise (green and blue, dashed lines). The episode terminates after 20 time steps.

should be considered for rewards given to different modes of operation in this domain. The multi-agent architecture improves the initial learning curve of the DYNA agent (see figure 3) however this accelerated exploitation is clearly detrimental to the agent's exploration as the distributed agent network response tapers off quickly compared to the single-agent equivalent that converges on a near-optimal policy much faster.

An important feature table I captures is the effect the increase in state- and action-spaces has on the agents. For lower order MDPs the agents can typically converge on a near optimal policy, evident y the relatively high reward. Further supporting this notion is that the continuous MDP optimized DYNA with incremental control outperforming the single agent architecture of the same configuration.

B. Gaussian Process Dynamic Programming Results

Figure 4 illustrates the results for the PILCO agent through a kinematic simulation, and physical system. The simulated noise was empirically selected as $\varepsilon \sim \mathcal{N}(0, 2 \times 10^{-6})$ as specified in section III. In addition, we show results for $\varepsilon \sim \mathcal{N}(0, 10^{-5})$ to illustrate the efficacy of GPDP at handling noisy input.

The real system is not wholly consistent with the simulation. This is primarily due to the conservative estimate of the maximal expected noise and that the (95%) confidence interval about the first step is approximately ± 3.5.

Learning manipulator control allows agents to see a relationship between their actions and the state-reward tuple, however there are significant advantages to one method over another in a particular domain; in this setup it is clear that the PILCO agent trumps conventional agents at generalized manipulator control from scratch, noting the clear improvement of figure 4 over both figure's 2 and 3.

In GPDP the loss function minimizes by differentiation w.r.t. the policy parameters. The algorithm then infers the

TABLE I

AVERAGED RESULTS OVER 100 EPISODES FOR EACH EXPERIMENT EXECUTED WITHIN THE CONVENTIONAL AGENT ARCHITECTURES. HIGHER ACCUMULATED REWARD PERTAINS TO A BETTER RESPONSE FROM THE AGENT. '-' REFERS TO INVALID OR INTRACTABLE RESULTS.

	Single Agent Architecture						Multi Agent Architecture			
	Unconstrained			Incremental			Unconstrained		Incremental	
DOFs	1	2	3	1	2	3	2	3	2	3
Q-learner	-0.20	-16.93	-156.7	-14.35	-84.8	-950	-25.5	-245.82	-52.1	-220
SARSA	-0.37	-13.23	-141.7	-9.29	-44.9	-616	-24.15	-247	-27.9	-201
DYNA	0.00440	-10.67	-108.75	-13.05	-52.07	-656	-19.11	-236	-31.6	-135
R-MAX	-4.08	-61.4	-	-3.05	-176.3	-	-59.0	-	-60.5	-
TEXPLORE	0.742	-48.09	-	-2.53	-	-	-	-	-	-
Continuous DYNA	0.19	-10.37	-85.28	-3.41	-21.2	-301	-16.14	-158.8	-14.08	-108.4
Continuous TEXPLORE	-7.47	-28.0	-	-15.72	-21.80	-	-	-	-	-

DP value function via Bayesian probabilistic measures from a function space view. In the conventional implementation, however, value- and policy-iteration without function approximation use a discretised (tabulated) input-space that is typically updated based on some type of Frequentist methodology. Therefore, in a continuous state-space the conventional approach is typically to quantize all states within a discrete cell. The result is that GPDP agents estimate a smooth continuous function and the discretised, conventional architecture will observe on a state-by-state basis and not assume any relationship to neighbouring states. These characteristics explain the extraordinary disparity in the learning slope of figure 4 versus figures 2 and 3.

V. CONCLUDING REMARKS

This paper evaluated the efficacy of a variety of agents toward learning closed-loop control of a manipulator in a simple task. The algorithms used were: Q-learner, SARSA, DYNA. R-MAX, TEXPLORE and the GPDP agent PILCO. This work extends [1] by comparing the PILCO agent qualitatively against predeceding agents for up to three degrees of freedom (DOFs).

The experiments were conducted with the intention of maximising common environmental factors in an effort to ascertain more valued agents for the manipulator domain. Further, the basic conventional agents were optimized for continuous MDPs with the inclusion of the UCT planner with M5 decision trees.

Our results show that PILCO outperforms all conventional agents by orders of magnitude in terms of efficiency and exploitation. The results clearly allude to the merit of the conventional agents in finite state spaces with lower order Markov Decision Processes, however the curse of dimensionality applies upwards of three DOFs.

VI. ACKNOWLEDGEMENTS

This work has been supported by the Australian Centre for Field Robotics and the Rio Tinto Centre for Mine Automation. We would also like to extend our gratitude to Marc Deisenroth for supplying the PILCO framework source code and ongoing assisting with its implementation; and Todd Hester for his RL package in ROS and informative discussions over continuous MDP algorithms.

REFERENCES

[1] M. P. Deisenroth, C. E. Rasmussen, and D. Fox, "Learning to control a low-cost manipulator using data-efficient reinforcement learning," in *Robotics: Science and Systems*, H. F. Durrant-Whyte, N. Roy, and P. Abbeel, Eds., 2011.

[2] J. Maitin-Shepard, M. Cusumano-Towner, J. Lei, and P. Abbeel, "Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding," in *ICRA*. IEEE, 2010, pp. 2308–2315.

[3] M. T. Mason, A. Rodriguez, S. S. Srinivasa, and A. S. Vázquez, "Autonomous manipulation with a general-purpose simple hand," *I. J. Robotic Res.*, vol. 31, no. 5, pp. 688–703, 2012.

[4] A. L. Thomaz and C. Breazeal, "Teachable robots: Understanding human teaching behavior to build more effective robot learners," *Artif. Intell.*, vol. 172, no. 6-7, pp. 716–737, 2008.

[5] J. Kober and J. Peters, "Policy search for motor primitives in robotics," *Machine Learning*, vol. 84, no. 1-2, pp. 171–203, 2011.

[6] J. Pazis, "Reinforcement learning in multidimensional continuous action spaces," in *ADPRL*. IEEE, 2011.

[7] M. P. Deisenroth, "Efficient reinforcement learning using gaussian processes," Ph.D. dissertation, 2010.

[8] M. P. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *ICML*, L. Getoor and T. Scheffer, Eds. Omnipress, 2011, pp. 465–472.

[9] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, UK, 1989.

[10] G. A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," Tech. Rep., 1994.

[11] R. S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *ML*, B. W. Porter and R. J. Mooney, Eds. Morgan Kaufmann, 1990, pp. 216–224.

[12] R. I. Brafman and M. Tennenholtz, "R-max - a general polynomial time algorithm for near-optimal reinforcement learning," *Journal of Machine Learning Research*, vol. 3, pp. 213–231, 2002.

[13] T. Hester and P. Stone, "Real time targeted exploration in large domains," in *ICDL*, 2010.

[14] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[15] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *ECML*, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds., vol. 4212. Springer, 2006, pp. 282–293.

[16] R. J. Quinlan, "Learning with continuous classes," in *5th Australian Joint Conference on Artificial Intelligence*. Singapore: World Scientific, 1992, pp. 343–348.

[17] C. E. Rasmussen and C. K. I. Williams, *gaussian processes for machine learning*. MIT press, 2006.

[18] M. P. Deisenroth, C. E. Rasmussen, and J. Peters, "Gaussian process dynamic programming," *Neurocomputing*, vol. 72, no. 7-9, pp. 1508–1524, 2009.

[19] C. E. Rasmussen, "Gaussian processes in machine learning," in *Advanced Lectures on Machine Learning*, O. Bousquet, U. von Luxburg, and G. Rätsch, Eds., vol. 3176. Springer, 2003, pp. 63–71.

[20] H. Kimura, "Reinforcement learning in multi-dimensional state-action space using random rectangular coarse coding and gibbs sampling," in *IROS*. IEEE, 2007.

[21] J. A. M. Hernández and J. de Lope Asiaín, "A distributed reinforcement learning control architecture for multi-link robots - experimental validation," in *ICINCO-ICSO*, J. Zaytoon, J.-L. Ferrier, J. Andrade-Cetto, and J. Filipe, Eds. INSTICC Press, 2007, pp. 192–197.