Data Efficient Direct Policy Search For Manipulation Actions Using Bayesian Optimisation

Matthew Broadway, **Jeremy L. Wyatt**, **Michael J. Mathew**, **Ermano Arruda**
School of Computer Science, University of Birmingham, UK

***Abstract*— A challenge in robot manipulation is how to learn tasks efficiently. We combine learning from demonstration with data efficient exploration guided by Bayesian optimisation. We use dynamic movement primitives to encode manipulation actions. These permit temporal and spatial scaling of the demonstrated trajectory. We demonstrate the effectiveness of direct policy search for the scaling parameters with Bayesian optimisation (BO). We evaluate BO against random search on two real robot tasks: a 'throw object to target' task, and a 'flip object to target' task. We are able to obtain good policy parameters despite large amounts of noise and a weak relationship between the parameters and the policy score.**

***Index Terms*— Bayesian Optimisation, DMP, Policy Search**

I. Introduction

A. Motivation

Direct policy search is an approach to reinforcement learning which can be useful in situations where the environment is too difficult to model accurately. The policy search is 'direct' because it treats a parameterised policy as a black box, and uses a black box optimisation algorithm to tune the policy parameters to improve a score on a task.

Bayesian optimisation is preferred to other black box optimisation algorithms such as random search or covariance matrix adaptation evolutionary strategy (CMA-ES) due to its data efficiency, allowing good parameters to be discovered with a limited budget of trials (attempts at the task). This is especially important when experimenting with robots in the real world since it is costly (in time, human effort and energy) to perform a large number of trials.

B. Bayesian Optimisation

Bayesian optimisation is a black box global optimisation algorithm which focuses on data efficiency [1]. Black box global optimisation is the problem of finding an input which maps to the global minimum or maximum (depending on the task) of a function without using any derivatives or knowledge of its internal structure, since these may not be available. The only possible interaction with the function is to evaluate it for a given input and observe the result, which may include noise if the function is stochastic.

Given an unlimited evaluation budget, a reasonable approach to global optimisation would be to evaluate randomly chosen points in the input space and take the best. This 'random search' approach is in some ways optimal if nothing is known about the function being optimised [2]. However, for many functions encountered in the real world, prior knowledge of the function's structure and smoothness can be exploited to guide the search more intelligently. In addition, some tasks involve objective functions which are costly to evaluate, reducing the effectiveness of random search.

A classic example of where Bayesian optimisation can be beneficial is the optimisation of hyperparameters for a machine learning model. Each 'evaluation' requires retraining the model with the chosen hyperparameters which can take several hours or even days.

Bayesian optimisation has been used for robotic tuning problems and some robot policy search tasks, however, the environments for policy search are often simulated [3][4].

Bayesian optimisation usually begins by sampling in an 'uninformed' manner for several trials before a surrogate model can be used. Typically, inputs are sampled uniform-randomly in the input space or pseudo-randomly with techniques such as Latin hypercube sampling (LHS) [5]. Each input is evaluated by the objective function f to obtain an initial data set $\mathcal{D} = \{(\boldsymbol{x}_i, f(\boldsymbol{x}_i)), \dots\}$. This data is used to fit a 'surrogate model', a probabilistic model (typically a Gaussian process) capable of predicting the mean and variance (uncertainty) of f at any point in the input space. An 'acquisition function' α then uses the model's predictions to evaluate the desirability of sampling at a particular input with the goal of reaching a global optimum. This function performs an exploration/exploitation trade-off and often has a parameter to control how much exploration is desired.

For this investigation, the negative lower confidence bound (-LCB) acquisition function was used with a gradually increasing β parameter as proposed in [6] to achieve bounded cumulative regret.

$$\alpha_{-\mathrm{LCB}}(\boldsymbol{x}; \beta) = -(\mu(\boldsymbol{x}) - \beta\sigma(\boldsymbol{x})) \tag{1}$$

where μ and σ are the predicted mean and standard deviation respectively, and β is the trade-off parameter.

The best choice of β for LCB is sensitive to the task and often difficult to determine a priori, due to factors such as the surrogate predicting greater uncertainty for high dimensional $\boldsymbol{x}$. For this reason, another acquisition function: expected improvement (EI) was used for the higher dimensional tasks.

$$\alpha_{EI}(\boldsymbol{x}; \xi) = \begin{cases} \sigma(\boldsymbol{x})\big(\gamma(\boldsymbol{x})\Phi(\gamma(\boldsymbol{x})) + \phi(\gamma(\boldsymbol{x}))\big) & \text{If } \sigma(\boldsymbol{x}) > 0 \\ 0 & \text{Otherwise} \end{cases}$$

$$\gamma(\boldsymbol{x}) = \big((y^+ - \xi) - \mu(\boldsymbol{x})\big)/\sigma(\boldsymbol{x}) \tag{2}$$

$\phi(\cdot), \Phi(\cdot)$ are the standard normal PDF and CDF respectively. ξ is the exploration trade-off parameter which controls the minimum desired improvement over the incumbent (current best) y^+. The ξ parameter is easier to set a priori and some implementations exclude it entirely, making EI parameter free.

The acquisition function is globally maximised over the input space to obtain the next input $\boldsymbol{x}_i$. The input is evaluated and another sample $(\boldsymbol{x}_i, f(\boldsymbol{x}_i))$ is added to the data set $\mathcal{D}$. The process then repeats with the new data set. The algorithm terminates after a fixed number of trials or if the best value (incumbent) does not change for a number of trials.

For this investigation, a highly configurable Bayesian optimisation framework: 'turbo', was created. It is free (GPLv3) and available for download on github [7].

C. Dynamic Movement Primitives

Dynamic movement primitives (DMPs) [8] are a mechanism for encoding the changes in a single degree of freedom throughout an action in a way that can be generalised both spatially and temporally. This means that a set of DMPs can be used to encode a trajectory demonstrated to a robot, and then the trajectory can be played back at different speeds or with different initial/goal positions. When altering the initial/goal positions, the trajectory retains the behaviour of the original demonstration. For example, an action for picking up a cup could be demonstrated and then generalised in order to pick up a cup in a different location.

A discrete DMP encodes an action which does not repeat (unlike a rhythmic action such as walking) using a combination of a point attractor and a non-linear 'forcing function'. The point attractor is a dynamical system which acts like a spring-damper system to draw the value of the DoF (degree of freedom) towards the goal state. The forcing function imposes an arbitrary trajectory on the way to the goal state but has its influence decreased as the action progresses to let the attractor take over, ensuring that the goal is always reached. The forcing function can be learned from demonstration using a weighted set of basis functions spread throughout the duration of the action. This is what allows the temporal generalisability. If the duration is scaled then the positions of the basis function scale by the same amount, retaining the same forcing function shape. The forcing function is multiplied by the difference between the goal and initial values, so if either is moved then the amplitude of the forcing function scales accordingly. This gives the spatial generalisability.

The motion equation for the point attractor (spring damper) is $\ddot{y} = K_p(g-y) - K_d\dot{y}$ where y is the value for the DoF, g is the goal value, and K_p and K_d are the proportional and derivative gain terms. Rather than defining the system in terms of time (t), an exponentially decaying 'canonical system' is used instead $\tau\dot{x} = -\alpha x \implies x = e^{-\alpha t/\tau}$. Removing the dependence on time allows for temporal generalisation by altering the time scaling factor τ, which is the variable we are interested in optimising for a task. The forcing function can then be defined as:

$$f(x) = \sum_{i=1}^{N} w_i\psi_i(x) \;/\; \sum_{i=1}^{N} \psi(x) \tag{3}$$

$$\psi_i(x) = \exp(-h_i(x - c_i)^2) \tag{4}$$

Where ψ_i is a Gaussian basis function centered at some 'time' (canonical system value) c_i with width h_i. Overall the motion equation for the DMP dynamical system is:

$$\tau^2\ddot{y} = K_p(g - y) - K_d\dot{y} + x(g - i)f(x) \tag{5}$$

$$\tau\dot{x} = -\alpha x \qquad \implies \qquad x = e^{-\alpha t/\tau} \tag{6}$$

where i is the initial value. By multiplying the forcing function by x it is guaranteed that as time progresses, the forcing function loses influence (since x decays to 0) and $g-i$ provides the spatial generalisability as described earlier.

Note that if DMPs are defined in the joint space then spatial generalisation in terms of task space locations is lost. Simply mapping the new task space goal into the joint space is insufficient since the joint space trajectory to reach the goal may be very different from the demonstration.

For this investigation, the pydmps library [9] was used to learn DMPs from demonstration and play them back.

II. Method

The investigation was carried out on a Baxter robot with an overhead camera used to detect ArUco markers in order to determine the pose of a small box (10cm cube). A target was placed down on a table in front of the Baxter robot, which would attempt to move the box to the target location using various actions.

Several experiments were carried out to evaluate the efficacy of Bayesian optimisation for the setup being tested. Each experiment run consists of 80 trials (attempts at reaching the target) of the chosen optimisation technique, followed by a 'verification run' of 10 trials with the best parameters found during the run to evaluate the reliability of the policy.

All Bayesian optimisation runs were initialised with 20 trials[1] selected with LHS, followed by 60 Bayesian optimisation trials. The sum of an ARD[2] Matern 5/2 kernel and white kernel was used for the Gaussian process, which was trained using L-BFGS-B[3] with a different number of restarts each trial (cycling between 10, 5 and 2).

Both tasks use the squared distance from the box resting position to the target (in metres) as the objective function to be minimised.

A. Flipping

This experiment tests the effectiveness of Bayesian optimisation on a simple real-world policy search task. This experiment used the -LCB acquisition function with $\beta = 1.5\log(n-8)$ where n is the trial number.

For this task, Baxter was given a spatula to hold, and for each trial, the box is placed onto the spatula and then flipped in the direction of the target, which was placed approximately 1.5m from the robot. The flipping action was recorded as joint space DMPs, meaning that the time scaling factor is the only parameter to be optimised.

Another experiment (referred to as 'with gains') was carried out to determine whether including parameters of the PD controller which carries out the policy has any positive effect on the search. In this experiment, the proportional and derivative gains (K_p and K_d respectively), as well as the tracking error threshold for the PD controller, were optimised in addition to the time scaling factor of the DMP.

B. Throwing

This experiment compares Bayesian optimisation to random search, as well as evaluating the performance of Bayesian optimisation on variations of the same task. The

[1] except for the flipping task without gains which only used 10

[2] automatic relevance detection using different lengthscales in each dimension

[3] Limited-memory-Broyden-Fletcher-Goldfarb-Shanno-bounded

Bayesian optimisation runs used the EI acquisition function with $\xi = 0.001$.

For this task, Baxter was given a block of wood to hold which the box was placed on top of and then thrown to a target placed approximately 1.5m from the robot. The difference between this experiment and the first is that the task of moving the box to the target is achieved with a choice of actions, and actions which are inherently less accurate than the flipping action due to the limitations of the robot.

For this task, three different throwing actions were recorded as joint space DMPs. This approach was chosen rather than utilising the spatial generalisation of a single task space DMP to ensure that the trajectories are free from kinematic singularities. With joint space DMPs, the trajectory through joint space is identical to the demonstration, unlike task space DMPs where it is dictated by the inverse kinematics solver.

The choice between the n discrete actions is encoded as a point $\boldsymbol{p}$ the continuous space $[0, 1]^n$ using $\text{choice} = \text{argmax}_{i \in 1..n}\, p_i$ (similar to a one-hot encoding). This results in large portions of the space have the same meaning, but this encoding is required for the Gaussian process to fit the data since it does not accept categorical inputs.

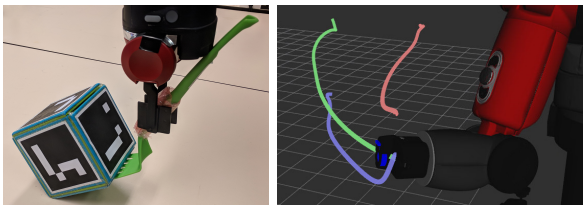

Fig. 1. Left: The initial state for the flipping action. Right: the trajectories of the throwing actions (red: 'throw1', green: 'throw2', blue: 'throw3')

III. Results

A. Flipping

gains	Run No.	time_scaling	ctrl_kp	ctrl_kd	ctrl_err_t	cost (m^2)
N	1	2.39	5	$\sqrt{0.01}$	0.3	3.63×10^{-4}
N	2	2.43	5	$\sqrt{0.01}$	0.3	4.98×10^{-4}
N	3	2.46	5	$\sqrt{0.01}$	0.3	$\mathbf{1.91\times10^{-4}}$
Y	1	2.50	4.93	$\sqrt{0.00544}$	0.329	$\mathbf{1.48\times10^{-4}}$
Y	2	2.40	4.50	$\sqrt{0.00702}$	0.303	1.06×10^{-3}

1) Discussion: The results are inconclusive as to whether including the controller parameters in the optimisation process makes a difference, however in both cases, the error consistently decreased by several orders of magnitude throughout the entire run. The ARD kernel determined the K_p and K_d parameters to be unimportant, whereas the error threshold parameter did make a slight difference. Only small ranges around some known good gain values were used to avoid damaging the robot, however, this may be the reason why these parameters were deemed not useful. From earlier testing, it is clear that increased dimensionality reduces the effectiveness of Bayesian optimisation, so the gains may slightly improve the search, but the increased difficulty from more parameters cancels this out.

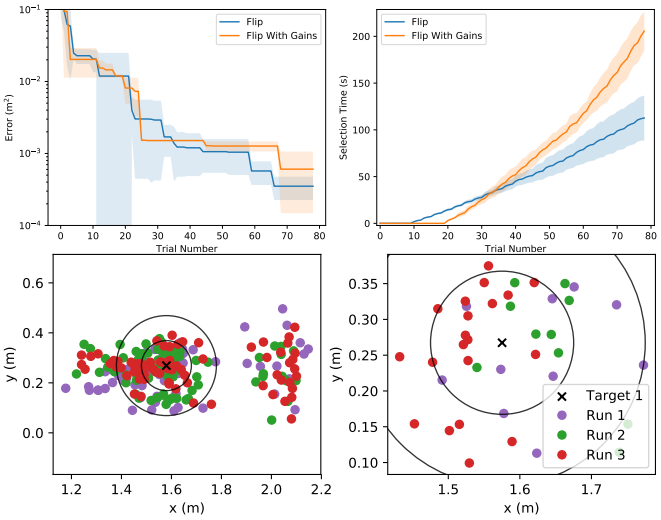

Fig. 2. Top Left: minimum objective function error after each trial. Top Right: cumulative selection time up to each trial. Bottom Left: the resting positions of the box on the table for every trial in experiment 1. Concentric circles mark one and two box widths from the target. Bottom Right: the resting positions of the box on the table for every trial in the verification runs for experiment 1.

However, this is a small sample over a single task and so more experiments are required to draw any conclusions.

B. Throwing

Target	Run No.	time_scaling	action1	action2	action3	Chosen	cost (m^2)
1	1	2.01	0.91	0.58	0.11	1	8.77×10^{-03}
1	2	3.00	1.00	0.47	0.76	1	$\mathbf{2.72\times10^{-04}}$
1	3	2.58	0.11	0.00	0.00	1	9.06×10^{-04}
2	1	1.71	0.53	0.22	0.24	1	2.51×10^{-03}
2	2	2.87	0.43	0.62	0.02	2	$\mathbf{1.36\times10^{-03}}$
2	3	2.48	0.03	1.00	0.73	2	5.66×10^{-03}
2	Random 1	2.27	0.05	0.51	0.17	2	5.45×10^{-03}
2	Random 2	1.31	0.04	0.06	0.06	3	$\mathbf{2.76\times10^{-03}}$
2	Random 3	2.82	0.57	1.00	0.87	2	6.51×10^{-03}

1) Discussion: In this experiment, a small difference is observed between the performance of Bayesian optimisation and random search in Figure 4, with both achieving reasonable results after only about 30 trials. For both targets, the best policy found by Bayesian optimisation improved by an order of magnitude over the course of the run.

Figure 4 shows both a large range of outcomes for similar time scales and a weak relationship between the time scale and the resting position. With the significant noise present in the resulting resting box position, the influence of the time scale on the position would be very hard to determine accurately from only a few samples, especially because only the squared distance to the target is given as feedback, rather than the resting position.

It is clear from Figure 3 that the policy using 'throw3' found by experiment 2 of random search is the best in terms of reliability since it consistently lands close to the target, whereas all the other policies only occasionally land close. However, the table shows that this policy did *not* achieve the best cost during optimisation.

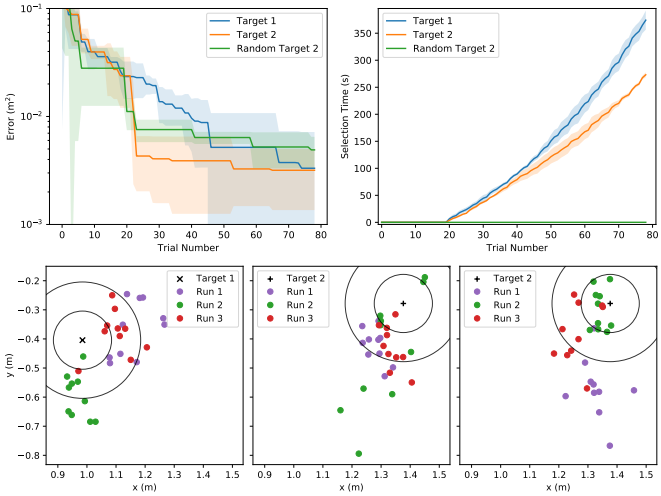

Fig. 3. Top Left: minimum objective function error after each trial. Note that the results from the first and second target locations cannot be directly compared because they are essentially different tasks. Top Right: cumulative selection time up to each trial. Bottom: The resting box positions of the verification runs (left: BO target 1, middle: BO target 2, right: random search target 2).

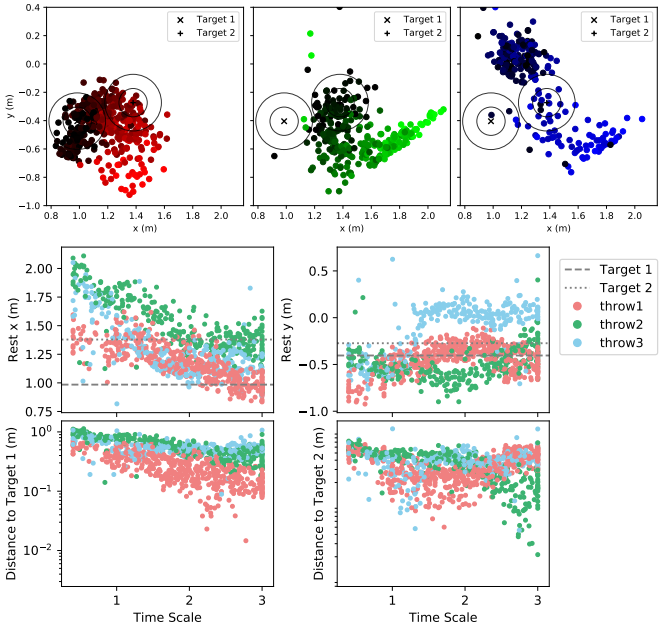

Fig. 4. Top: Projectile resting positions for each throwing action (left: 'throw1', middle: 'throw2', right: 'throw3'), colored by time scale (brighter is smaller time scale and therefore faster). By inspection, the influence of time scale on the resting position is very weak. One and two box widths from the targets are shown. Bottom Left 2: the x and y resting resting positions as a function of the time scale. Bottom Right 2: the distances to target 1 and target 2 as a function of the time scale respectively.

IV. Conclusion

The results from the flipping task demonstrates that Bayesian optimisation can be used to obtain reliable policies (which consistently score well) for real world tasks in a small number of trials. The results from the throwing task show some improvement using Bayesian optimisation over random search in a scenario which is very challenging due to number of factors. The throwing actions are less precise than flipping, leading to a lot of noise in the resting position and only a weak relationship between the input parameters and the score of the policy.

Figure 4 shows that for any given time scale, none of the actions reliably reached the target location, meaning it is unlikely that any policy in the search space could achieve very reliable performance. Despite all of these challenges, Bayesian optimisation was able to obtain policies which scored well in every run, indicating that the method is fairly robust to the difficulties encountered in real world tasks.

The verification results identify a potential problem with the approach to Bayesian optimisation used in this work. The optimiser does not take the reliability of the policy into account when deciding which parameters are best, since the policy is only tested once, meaning unreliable policies may be chosen if they happen to score well by chance. If reliable policies are desired then the optimiser must be modified to distinguish reliable policies from well scoring policies which were lucky once but not reliable. Possible techniques to investigate in future involve minimising/maximising the surrogate mean rather than the trial costs when taking the best parameters [10], or scoring each policy using the mean of multiple evaluations to reduce noise in the objective function.

Future investigations could assess the performance of BO in optimising 'upper level policies' (which can generalise to different variations of a task) for use in the real world. In addition, evaluating the effectiveness of including controller gains into the optimisation process was not fully explored in this investigation, with restrictions placed on the range of values due to safety concerns. However, there are modifications to BO which can incorporate safety constraints [11].

References

[1] J. Močkus, 'On Bayesian methods for seeking the extremum', in *Optimization Techniques IFIP Technical Conference*, Springer, 1975, pp. 400–404.

[2] M. O. Ahmed, B. Shahriari and M. Schmidt, 'Do we need "harmless" Bayesian optimization and "first-order" Bayesian optimization', *NIPS BayesOpt*, 2016.

[3] J. H. Metzen, A. Fabisch and J. Hansen, 'Bayesian Optimization for Contextual Policy Search', in *Proceedings of the Second Machine Learning in Planning and Control of Robot Motion Workshop., Hamburg*, 2015.

[4] P. Karkus, A. Kupcsik, D. Hsu and W. S. Lee, 'Factored Contextual Policy Search with Bayesian Optimization', *arXiv preprint arXiv:1612.01746*, 2016.

[5] M. T. Morar, J. Knowles and S. Sampaio, 'Initialization of Bayesian Optimization Viewed as Part of a Larger Algorithm Portfolio', 2017.

[6] N. Srinivas, A. Krause, S. M. Kakade and M. Seeger, 'Gaussian process optimization in the bandit setting: No regret and experimental design', *arXiv preprint arXiv:0912.3995*, 2009.

[7] M. Broadway, *turbo*. [Online]. Available: https://github.com/mbway/turbo.

[8] S. Schaal, 'Dynamic movement primitives-a framework for motor control in humans and humanoid robotics', in *Adaptive motion of animals and machines*, Springer, 2006, pp. 261–280.

[9] T. DeWolf, *PyDMPs*. [Online]. Available: https://github.com/studywolf/pydmps.

[10] M. W. Hoffman and B. Shahriari, 'Modular mechanisms for Bayesian optimization', in *NIPS Workshop on Bayesian Optimization*, 2014, pp. 1–5.

[11] Y. Sui, A. Gotovos, J. Burdick and A. Krause, 'Safe exploration for optimization with Gaussian processes', in *International Conference on Machine Learning*, 2015, pp. 997–1005.