Learning to Race through Coordinate Descent Bayesian Optimisation

Rafael Oliveira, Fernando H.M. Rocha, Lionel Ott, Vitor Guizilini, Fabio Ramos and Valdir Grassi Jr.

***Abstract*— In the automation of many kinds of processes, the observable outcome can often be described as the combined effect of an entire sequence of actions, or controls, applied throughout its execution. In these cases, strategies to optimise control policies for individual stages of the process might not be applicable, and instead the whole policy might have to be optimised at once. On the other hand, the cost to evaluate the policy's performance might also be high, being desirable that a solution can be found with as few interactions as possible with the real system. We consider the problem of optimising control policies to allow a robot to complete a given race track within a minimum amount of time. We assume that the robot has no prior information about the track or its own dynamical model, just an initial valid driving example. Localisation is only applied to monitor the robot and to provide an indication of its position along the track's centre axis. We propose a method for finding a policy that minimises the time per lap while keeping the vehicle on the track using a Bayesian optimisation (BO) approach over a reproducing kernel Hilbert space. We apply an algorithm to search more efficiently over high-dimensional policy-parameter spaces with BO, by iterating over each dimension individually, in a sequential coordinate descent-like scheme. Experiments demonstrate the performance of the algorithm against other methods in a simulated car racing environment.**

I. Introduction

In the automation of various kinds of physical systems, sometimes a controller, or a control policy, needs to be optimised based only on a total cost or reward evaluating its performance. In robotics, the physical processes of interest are often related to the motion or navigation of a robot. In some cases, it is impossible to quantify the effect that control actions at individual steps have on the final outcome of the process. Examples include tasks such as ball-throwing [1] or trying to hit a target on a wall with a dart [2]. In other cases, individual rewards might be quantifiable for each step taken, but a global approach, considering a whole episode of execution, could yield better results, such as in autonomous racing [3].

In the case of autonomous racing, the problem of finding a control policy that will allow a robot racer to finish a track in minimal time has been approached in many ways. Model-predictive control (MPC) techniques have been applied to locally optimise driving policies over receding-horizons based on external sensor data and internal dynamical models, which are either pre-designed [4] or learnt [5]. From a global optimisation perspective, the same problem has been approached as racing line optimisation [3], i.e. finding the path within the track that will allow the car to finish the race in minimal time, given a map of the race track and a kinematic model of the vehicle. When global information about the racing environment is available through external images, deep reinforcement learning architectures have also been applied to fine-tune control policies [6]. Lastly, this problem has also been approached by using evolutionary algorithms to optimise parameters for policies that either combine different pre-designed heuristics [7], or directly map the state of the car and the opponents to control actions via some type of network structure [8].

In this paper, we are concerned with the problem of enabling a robot to learn the control policy that will allow it to complete a lap in a given race track as fast as possible, improving over a single initial example given by a sub-optimal controller. This is performed without the need for a model of the robot or a map of the track, but by performing multiple laps and learning from the outcome of each of them. Model-based approaches are generally limited by the ability of the model to represent the behaviour of the robot [9], which can be hard to capture for real robots. In addition, these approaches usually require the Markov assumption to be valid for the model representation, which implies that enough information about the dynamic state of the robot is observable.

In a policy-search framework, ideally we look for policies which are flexible enough to represent a variety of behaviours and an optimisation procedure that can find the best policy within a minimal amount of episodes and requires only minimal information about the system. Radial basis function (RBF) networks [9] are among the classes of policy parameterisations with high representation power. These policies, however, can be very high dimensional and challenging to optimise with conventional optimisation algorithms within a limited budget of policy evaluations. Bayesian optimisation (BO) [10], on the other hand, is a strategy to optimise expensive-to-evaluate functions within a limited budget of function evaluations that has shown promising results in policy search problems [11], [12]. Although methods which apply BO in high-dimensional search spaces have been proposed [13], [14], it still remains an open question how to do so without making restrictive assumptions about the objective function.

Our contribution is a method that applies Bayesian optimisation (BO) [10] to optimise control policies with high-dimensional parameter spaces. In each iteration of the BO loop, our method sequentially optimises each parameter

Rafael Oliveira, Lionel Ott, Vitor Guizilini and Fabio Ramos are with the School of Information Technologies, The University of Sydney, Australia. rdos6788@uni.sydney.edu.au

Fernando H.M. Rocha and Valdir Grassi Jr. are with the Sao Carlos School of Engineering at the University of Sao Paulo, Sao Carlos, SP, Brazil, fernandorocha@usp.br

This work was supported by *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* (CAPES), Brazil, and Data61/CSIRO, Australia.

of the policy in turn, following a randomised coordinate descent [15] scheme. This is performed over a Gaussian process [16] surrogate, modelling a reward function, that is sequentially updated. The method is applied to optimise control policies for a racing car in a race simulator. We employ a general class of control policies, defined by feature expansions in a reproducing kernel Hilbert space (RKHS) [17]. RBF networks can be seen as a specialised instance of this generic representation.

The remainder of this paper is organised as follows. In the next section, we review relevant related work in the area of Bayesian optimisation. In Section III, we formally present the optimisation problem approached by this paper and some background information about BO. Then, in Section IV, we present our method. In Section V, we present experimental evaluations of the method using a race car simulator, comparing the proposed method against other optimisation algorithms. Finally, in Section VI, we conclude and propose some directions for future work.

II. Related Work

In this section, we review relevant prior work in the areas of reinforcement learning and Bayesian optimisation.

Learning a control policy that maximises a reward that depends on the combined effect of its actions can also be approached as episode-based policy search in reinforcement learning (RL) [9]. In [1], for example, a framework is proposed to allow algorithms that learn policies using an internal, also learned, forward model of the robot to simulate trajectory roll-outs. In [2], a robot learns how to hit a target throwing a dart using a model-free approach. This algorithm models the policy parameters distribution with a Gaussian process (GP) [16] prior over the contexts space for each single parameter.

Bayesian optimisation [10] has also been applied to problems involving policy search. A similar approach to the one in this paper is presented in [11], where the authors model cost as a direct function of policy parameters. In that paper, the robot's task was to reduce uncertainty about its location and its surroundings, and the policies were parameterised by a small number of variables. Since the policy-execution outcome/reward is usually more-closely a function of the resulting behaviour of the robot than of the policy parameters, another approach to this problem is applying a GP prior over this mapping from behaviours to rewards [12]. This approach, however, usually requires that enough information can be observed from the system. Our aim in this paper is to learn policies that can have enough representation power, which generally implies high-dimensional parameter spaces, while requiring the least amount of information from the system.

One common issue with the presented applications of BO is the curse of dimensionality. The great majority of BO algorithms uses GPs to learn and model the objective function, which does not scale well with high dimensions and/or large amounts of data, degrading the performance of BO in such settings. Several approaches have been recently developed to tackle the scaling of GP/BO to high dimensions. One could, for example, assume that the objective function only depends on a small subset of input coordinates and do variable selection to find these [18], [19]. Another approach is to assume there is a lower-dimensional linear embedding containing most of the variation of the function, including its optimum, and using linear projections to reduce the dimensionality of the search space [13], [20]. Another common approach is to assume that the function is formed by a set of low dimensional disjoint functions [14] combined in an additive structure, which allows learning separate GP models and optimising over them separately.

Given the particular nature of our problem, we chose to tackle issues that arise in high-dimensional BO by approaching it from the search side, while still using all available information to build a model of the objective function. In problems like racing, due to the constraints imposed by the environment, i.e. the track, and the dynamic limits on the agent, i.e. the car, we end up having a reward function whose mass is highly concentrated within a particular region of the search space. Therefore, we argue that, by applying a relatively simple method such as randomised coordinate descent (CD) [21], and starting the search from a valid initial solution, we can optimise over a high-dimensional GP model efficiently. CD relies on the fact that each subproblem is a lower-dimensional optimisation problem, which is solved more easily than the full problem. We empirically demonstrate that this simple search strategy when combined with BO can be effective in the particular class of problems we approach.

III. Preliminaries

In this section, we present some preliminary information for the work in this paper. We start with a formal description of the particular problem we are dealing with, followed by the formulation of our policies parametrisation. After that, we review some basic concepts in Bayesian optimisation and Gaussian process regression.

A. Problem Statement

Consider policies mapping the position $x \in \mathcal{X}$ of the robot along the track to a corresponding control action $a \in \mathcal{A}$. Both $\mathcal{X}$ and $\mathcal{A}$ are continuous spaces. Our goal is to find the control policy $\pi : \mathcal{X} \rightarrow \mathcal{A}$ that minimises the time T required to complete the track.

We utilise a reward R defined as:

$$R = \begin{cases} L/T, \text{ if track completed} \\ 0, \text{ if failed to complete the track} \end{cases} \tag{1}$$

where L is the length of the track. Therefore, for success cases, the reward is equivalent to the average linear speed of the robot. In this sense, minimising T is equivalent to maximising R. With that, we search for:

$$\pi^* = \underset{\pi \in \mathcal{A}^{\mathcal{X}}}{\operatorname{argmax}} \; R[\pi]. \tag{2}$$

We seek a method that solves the above problem for any kind of mobile robot and without needing a map of the track.

Therefore, model-based solutions are out of scope, for they would have to learn an approximate transition model of the robot, whose representation varies among different driving mechanisms, and use this model to simulate trajectories over the track map.

B. Policy Parameterisation

We assume that the optimal π belongs to a reproducing kernel Hilbert space (RKHS) [17]. This approach has been relatively successful in modelling trajectories for motion planning [22] and components of stochastic control policies for reinforcement learning [23], [24]. This formulation allows the policy to assume a variety of shapes, depending on the choice of kernel function, allowing one to encode prior knowledge about the ideal control policy. Considering a 1-D action space $\mathcal{A} \subset \mathbb{R}$, a policy can be represented by:

$$\pi(x) = \sum_{i=1}^{N} \alpha_i k(x_i, x), \tag{3}$$

where $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a positive-definite kernel function, and $\alpha_i \in \mathbb{R}$ and $x_i \in \mathcal{X}$ are arbitrary. Besides that, it is possible that $N \to \infty$.

In a RKHS, kernels are equivalent to inner products between features mappings $\phi : \mathcal{X} \to \mathcal{H}$ in the corresponding Hilbert space, such that $k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}}$. With that, $\pi(x)$ can also be represented by $\pi(x) = \sum_{i=1}^{N} \alpha_i \langle \phi(x_i), \phi(x) \rangle_{\mathcal{H}}$. By the linearity property of inner products, we can move the sum inside, and have:

$$\pi(x) = \langle \sum_{i=1}^{N} \alpha_i \phi(x_i), \phi(x) \rangle_{\mathcal{H}} = \langle w, \phi(x) \rangle, \tag{4}$$

where $w \in \mathcal{H}$, which can be infinite in dimensions. Several techniques in the kernel machines literature, e.g. [25], however, propose approximating $\phi(x)$ by a vector $\hat{\boldsymbol{\phi}}(x) \in \mathbb{R}^M$, $M < \infty$, such that $\hat{\boldsymbol{\phi}}(x)^{\mathrm{T}} \hat{\boldsymbol{\phi}}(x') \approx k(x, x')$. As a consequence,

$$\pi(x) \approx \pi_{\mathbf{w}}(x) = \mathbf{w}^{\mathrm{T}} \hat{\boldsymbol{\phi}}(x), \tag{5}$$

where $\mathbf{w} \in \mathbb{R}^M$ is a vector of scalar weights, which uniquely determines the policy π for a given feature mapping $\hat{\boldsymbol{\phi}}$. Therefore, we can rewrite Equation 2 as:

$$\mathbf{w}^* = \underset{\mathbf{w} \in \mathbb{R}^M}{\operatorname{argmax}} \; R[\pi_{\mathbf{w}}]. \tag{6}$$

In our case, the advantage of the features approximation is that we don't need to deal with the individual α_i's and x_i's, which can be many more than M, but only with their combined effect on the control policy. More importantly, we also reduce the dimensionality of the search space, projecting it to M dimensions.

Although the problem has been formulated for 1-D actions, it could be easily extended to actions composed by multiple independent controls, by simultaneously optimising the weight vectors for each corresponding policy. The drawback, however, is the multiplication of the number of weights to optimise.

C. Bayesian Optimisation

We use BO [10] to perform the policy weights optimisation as it allows finding the global optimum of arbitrary functions that are expensive to evaluate. To aid in that, BO applies a Bayesian model, which is typically a Gaussian process (GP) [16], as a prior to internally approximate the objective function.

At each iteration, BO selects the point to perform the next evaluation of the objective by maximising an acquisition function over the model. This acquisition function provides a utility value that enables the algorithm to perform a guided search for the global optimum, and is usually much simpler to evaluate than the objective function. It represents a natural trade-off between exploration (searching for areas with high uncertainty) and exploitation (searching for areas where the objective function is expected to be high), and aims to minimise the number of objective function evaluations.

After evaluating the objective function at the selected point, the prior is updated with the new observation and the algorithm proceeds to the next iteration, keeping track of the current optimum. As iterations proceed, it can be shown that the prior approximation converges to the objective function within the given search space, and consequently the global optimum of the objective can be found.

D. Gaussian Processes

Gaussian process [16] regression is a Bayesian non-parametric framework that places a Gaussian distribution as a prior over the space of functions mapping the inputs $\mathbf{w} \in \mathbb{R}^M$ to outputs $z \in \mathbb{R}$, where $z = f(\mathbf{w}) + \epsilon$ is a noisy observation of the true underlying reward value $f(\mathbf{w})$, and $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$ is Gaussian-distributed noise with zero mean and standard deviation σ_n. A GP model can be completely specified by a mean and a covariance function, k_R, which is a positive-definite kernel. Using zero as the mean for the prior, the values of f for a set of Q points $W^* = \{\mathbf{w}_i^* \in \mathbb{R}^M\}_{i=1}^{Q}$ obey a multivariate normal distribution:

$$\mathbf{f}^* = f(W^*) \sim \mathcal{N}(\mathbf{0}, k_R(W^*, W^*)), \tag{7}$$

where $f(W^*) = [f(\mathbf{w}_1^*), \ldots, f(\mathbf{w}_Q^*)]^{\mathrm{T}}$, and $k_R(W^*, W^*)$ is an Q-by-Q matrix whose elements are determined by $k_R(W^*, W^*)_{ij} = k_R(\mathbf{w}_i^*, \mathbf{w}_j^*)$. Given a set $\{W, \mathbf{z}\}$ of N observations of f, where $W = \{\mathbf{w}_i \in \mathbb{R}^M\}_{i=1}^{N}$ and $\mathbf{z} = \{z_i \in \mathbb{R}\}_{i=1}^{N}$, the joint distribution of the observed outputs and the function values at the query points under the GP prior is given by:

$$\begin{bmatrix} \mathbf{z} \\ \mathbf{f}^* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} k_R(W, W) + \sigma_n^2 I & k_R(W, W^*) \\ k_R(W^*, W) & k_R(W^*, W^*) \end{bmatrix}\right) \tag{8}$$

Conditioning this joint on the observations, inference in a GP can be done by:

$$\mathbf{f}^* | W, \mathbf{z}, W^* \sim \mathcal{N}(\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*), \tag{9}$$

where:

$$\boldsymbol{\mu}^* = k_R(W^*, W) K_W^{-1} \mathbf{z} \tag{10}$$

$$\boldsymbol{\Sigma}^* = k_R(W^*, W^*) - k_R(W^*, W) K_W^{-1} k_R(W, W^*), \tag{11}$$

using $K_W = k_R(W, W) + \sigma_n^2 I$, with each $k_R(W, W)_{ij} = k_R(\mathbf{w}_i, \mathbf{w}_j)$.

IV. Coordinate Descent Bayesian Optimisation

We approach the problem in Section III-A from a Bayesian optimisation (BO) perspective, which places a Gaussian process (GP) prior over the objective function, in our case, $R[\pi_{\mathbf{w}}] = f(\mathbf{w})$, and optimises it by doing searches over the GP-based surrogate model.

A. Acquisition Function Optimisation

In this paper, another problem that we face is the possibly high dimensionality of the search space for the optimisation of the acquisition function (AF) that BO utilises. Quite a few methods have been proposed in the BO literature to deal with high-dimensional search spaces, such as [13], [14]. However, they usually require a few possibly strong assumptions about the objective function. We instead employ a very simple method, Stochastic Coordinate Ascent (Algorithm 1), which uses a random axis selection scheme to optimise the AF over each axis individually, starting from the current optimum candidate. This way, we bias the search locally, avoiding excessive exploration in a high-dimensional space. Depending on the choice of acquisition function, however, we can still perform a global search with BO by following regions of high uncertainty around the current optimum location.

Our AF optimisation technique was inspired by Coordinate Descent (CD), a class of algorithms that are one of the oldest in the optimisation literature [26]. It is based on the idea that an n-dimensional problem can be decomposed in n one-dimensional sub-problems, which makes it suitable to be applied to large or high-dimensional datasets [15]. In the CD strategy, each coordinate is updated sequentially by solving the one-dimension problem with any suitable optimisation algorithm, while all the others dimensions are kept fixed. The methods vary in the way the sequence of dimensions is chosen, if it optimises only one dimension or a block, and if it uses gradients on the iterations or not. Several works focused on describing the convergence characteristics of these algorithms [15], [27], [28]. Under some assumptions (e.g. Lipschitz continuity, strong convexity) they prove a linear convergence rate for the sequential and randomised CD method, and also for the block-CD. The most intuitive scheme for this technique would be to optimise the dimensions in an ordered cyclical fashion. But [29] showed an example of non-convex function that, when applied this sequential scheme, the optimisation process cycles without converging. To avoid these kind of issues, in our work we adopt a randomised sequence that changes once every dimension has been optimised.

Algorithm 1 starts the search from the current optimum location. It randomly shuffles a sequence of M dimension indices $\{1, \ldots, M\}$, and follows the shuffled sequence to optimise one axis after the other. After passing through all the dimensions, it returns the corresponding optimised vector of weights.

Algorithm 1: StochasticCoordinateAscent

Input: $h, \mathbf{w}^*, \mathcal{D}$

$\mathbf{w} = \mathbf{w}^*$;

$\mathcal{I} = \texttt{randomShuffle}(\{1, \ldots, M\})$;

for $i \in \mathcal{I}$ **do**

  $w_i = \underset{v \in \mathbb{R}}{\operatorname{argmax}}\ h(w_0, \ldots, w_{i-1}, v, w_{i+1}, \ldots, w_M | \mathcal{D})$;

return $\mathbf{w}$

Although not using derivatives, as in standard coordinate descent methods [15], in practice our approach is still able to achieve similar results. We performed preliminary tests applying other CD methods, e.g. the randomised accelerated coordinate descent method (RACDM) [15]. We observed that their performance is usually not as good as the simple scheme in Algorithm 1. Reasons for that involve issues with escaping saddle points and restrictions in the GP covariance function, which needs to be differentiable for the method to be applicable. The latter is not the case of Matérn 1 [16], for example, the best performing in our experiments.

B. The Policy-Search Algorithm

We propose an algorithm that improves a valid initial policy so that the robot racer can finish the track in minimal time. The initial policy can be obtained by recording the actions of a simple controller or a human driver, obtaining an initial set of points $X = \{x_i\}_{i=1}^n$ and the corresponding observed actions $\mathbf{a} = \{a_i\}_{i=1}^n$. The initial weights $\mathbf{w}_0$ can then be fitted by minimising the quadratic cost function:

$$\mathbf{w}_0 = \underset{\mathbf{w} \in \mathbb{R}^M}{\operatorname{argmin}}\ \|\mathbf{a} - \hat{\boldsymbol{\phi}}(X)\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_2^2, \tag{12}$$

where $\hat{\boldsymbol{\phi}}(X) = [\hat{\boldsymbol{\phi}}(x_1), \ldots, \hat{\boldsymbol{\phi}}(x_n)]^{\mathrm{T}}$ is a matrix with the features for each x_i on the corresponding row and λ is a regularisation factor, to avoid extreme values for the weights. A solution to this problem can be found analytically by zeroing out the gradient of the fit term with respect to $\mathbf{w}$, yielding:

$$\mathbf{w}_0 = [\hat{\boldsymbol{\phi}}(X)^{\mathrm{T}} \hat{\boldsymbol{\phi}}(X) + \lambda I]^{-1} \hat{\boldsymbol{\phi}}(X)^{\mathrm{T}} \mathbf{a}. \tag{13}$$

Before starting the policy search with BO, an initial training of the GP is needed to provide an estimate of its hyper-parameters, which are the noise variance and the parameters of the covariance function in our case. Due to the vast majority of the search space being, in general, composed of invalid policies, using uniform or Latin hypercube random samples could provide too many observations with zero reward. This can cause over-fitting to the initial GP hyper-parameter selection. To avoid this, we sample and execute a set of S samples from a normal distribution $\mathcal{N}(\mathbf{w}_0, I\sigma_0^2)$ to form an initial dataset $\{\mathbf{w}_i, R_i\}_{i=1}^S$ to feed the GP with, so that BO can have an informative prior over the reward function. In addition, the hyper-parameters can also be re-estimated online after each observation of a reward.

The proposed method is summarised in Algorithm 2. In lines 1 through 4, it collects the initial set of observations for the GP. In lines 5 through 12, the search for the optimal policy is performed. Line 7 executes the policy parameterised by the current weights. Lines 8 through 10 keep track of the optimum. Line 11 updates the GP dataset. Line 12 performs the maximisation of the acquisition function to select the next vector of weights to evaluate. The algorithm proceeds until its budget of N function evaluations is exhausted.

Algorithm 2: CDBO

Input: $\mathbf{w}_0$: weights of the initial policy
σ_0^2: initial samples variance
S: number of initial samples
N: number of laps

1 **for** $s = 1 \dots S$ **do**
2 Sample $\mathbf{w}_s \sim \mathcal{N}(\mathbf{w}_0, I\sigma_0^2)$
3 $R_s \leftarrow$ Execute $\pi(x; \mathbf{w}_s)$
4 $\mathcal{D} \leftarrow \{\mathbf{w}_s, R_s\}_{s=1}^{S}$
5 $R^* \leftarrow \max R_{s=1,\dots,S}$
6 **for** $i = 0 \dots N - S$ **do**
7 $R_i \leftarrow$ Execute $\pi_{\mathbf{w}_i}$
8 **if** $R_i > R^*$ **then**
9 $R^* \leftarrow R_i$
10 $\mathbf{w}^* \leftarrow \mathbf{w}_i$
11 $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{w}_i, R_i)\}$
12 $\mathbf{w}_{i+1} =$ `StochasticCoordinateAscent`$(h, \mathbf{w}^*, \mathcal{D})$
13 **return** $\mathbf{w}^*, R^*$

V. Experiments

In this section, we present the experimental evaluation of the performance of our method in simulation. We performed tests with a robot car driving on race tracks performing realistic full physics simulation using the race engine of an open-source game, called *Speed Dreams*[1], which is based on TORCS [30]. In all the tests, we compared the performance of the algorithm against:

- CMA-ES, which has been applied to reinforcement learning problems [31], in particular, we use active CMA-ES [32], with an implementation provided by an open-source library[2];
- standard BO, using CMA-ES to optimise the acquisition function.
- REMBO [13], a Bayesian Optimisation technique that uses random embeddings, developed to deal with high-dimensional problems. We tested with both 5 and 10-dimensional random embeddings.

To run the internal optimisation along each coordinate in our method, we used COBYLA [33], a local derivative-free optimisation algorithm, with an implementation provided by a popular non-linear optimisation library [34].

[1]Speed Dreams: https://sourceforge.net/projects/speed-dreams/
[2]`https://github.com/beniz/libcmaes`

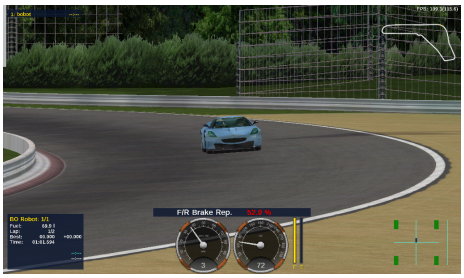


Fig. 1. Screen-shot of the race car used by our algorithm in simulations using the game engine.

A. Setup

The state space of the robot is represented as its position along a given race line normalised by its length, i.e. $x \in [0, 1]$, where 0 corresponds to the start line, increasing to 1 when the robot crosses the finish line. As race line, we used the centre line of the track, but it could be any other valid trajectory, allowing our method to be combined with racing line optimisation algorithms. The control policy actuates the car's acceleration by optimising throttle and braking, which are combined into a single scalar output $a \in [-1, 1]$, with positive values for throttle and negative for braking. The steering control was performed using a simple proportional-integral (PI) controller, which tries to minimise the distance between the car and the racing line. We have not approached the steering control in this paper, since optimising only the acceleration control of the car is already a challenging problem and sufficient to demonstrate the capabilities of the proposed BO method in dealing with high dimensions.

In this setup, the algorithm needed to be cautious not to set huge acceleration values for the car at critical parts of the track, like curves, both to not destabilise the steering controller and also to respect the friction limits of the tires. Figure 1 presents a screen-shot with the car model we used, a *Spirit 300*.

For features $\hat{\boldsymbol{\phi}}$, we utilised an array of M kernels placed over a set of inducing points along the track, i.e.:

$$\hat{\boldsymbol{\phi}}(x) = [k_a(x, \hat{x}_1), \dots, k_a(x, \hat{x}_M)]^{\mathrm{T}}. \tag{14}$$

We chose $\hat{X} = \{\frac{i}{M-1}\}_{i=0}^{M-1}$, so that it forms a set of M regularly-spaced points along the track. In this sense, no prior information about where the critical parts of the track are was assumed. However, a possible way to choose $\hat{X}$ would be to place points around portions of the track requiring a significant change in acceleration, such as curves, which would require a map of the track. This could also be learnt by analysing the initial control policy, searching for areas of high variation.

If we set k_a to be the RBF squared exponential kernel, we have the RBF network policy parameterisation, which has already been applied to some reinforcement learning problems [9]. However, that kernel has very smooth transitions, and would not be flexible enough to provide fast transitions in the commands profile. Therefore, for all the experiments, we used the *Matérn* class of kernel functions (see [16], Chapter

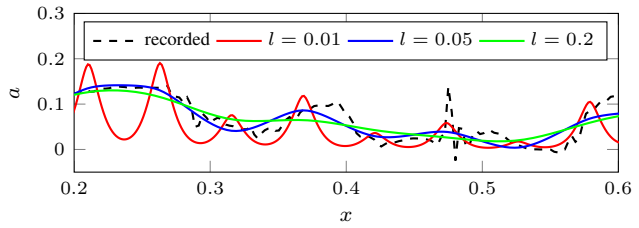


Fig. 2. Detail of the effect of different lengths scales on the fitting of the initial policy for the same kernel placement. The recorded actions are shown in a dashed line. Shorter length scales allow sharper transition, but might compromise interpolation. Longer length scales yield smoother curves, but might compromise flexibility

4) for the policies, k_a, in particular, we used Matérn 3:

$$k_a(x, x') = \left(1 + \frac{\sqrt{3}}{l}|x - x'|\right) \exp\left(-\frac{\sqrt{3}}{l}|x - x'|\right), \tag{15}$$

where l is a length-scale parameter controlling the smoothness of the curve. We set l with values around the spacing of the inducing points, i.e. $l \approx \frac{1}{M-1}$, in our experiments so that the gaps between the kernels can be filled without compromising the flexibility of the function. Figure 2 demonstrates the effect that different length scales have on the shape of the curve. The fitting of the length scale can also be numerically optimised before starting the race.

For the GP model, we used the Matérn 1 kernel as covariance function, which allows us some flexibility to model sharp transitions in the reward function. This Matérn kernel is equivalent to the exponential covariance function:

$$k_R(\mathbf{w}, \mathbf{w}') = \sigma_f^2 \exp\left(-\sqrt{d^2(\mathbf{w}, \mathbf{w}')}\right), \tag{16}$$

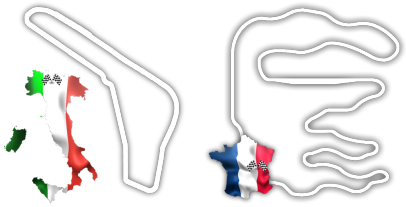
where σ_f^2 is a signal variance parameter and $d^2(w, w') = (w - w')^{\mathrm{T}} \Lambda^{-1} (w - w')$, with $\Lambda = \mathrm{diag}(l_i^2), i = 1, \ldots, M$, as a length-scales matrix for automatic relevance determination (ARD). The same GP hyper-parameters adaptation scheme proposed in [13] was applied to all BO methods. In the case of the GP noise model, since the simulations of the physics engine in the game are deterministic, we set the noise variance, σ_n^2 to 0.

As acquisition function for BO we applied the upper confidence bound (UCB) criterion:

$$h(\mathbf{w}|\mathcal{D}) = \mu(\mathbf{w}) + \beta\sigma(\mathbf{w}), \tag{17}$$

where $\mu(\mathbf{w})$ is the mean of the GP posterior at $\mathbf{w}$, $\sigma(\mathbf{w})$ is the square root of the GP posterior variance, and β is a parameter controlling the exploration-exploitation trade off. In most of our experiments, we were able to obtain good performance results with $\beta \in [0.5, 2]$ using CDBO. For the experiments with REMBO, we used the Expected Improvement acquisition function, since the results on [13] were achieved using this function.

The initial policy demonstration is given by a PI controller, whose only task is to drive the car at a constant speed of 15 m/s along the track, in all the experiments. An example of such data for one of the test tracks is shown in Figure 2. That raw data is then fit through the specified number of policy kernels, yielding the initial set of weights $\mathbf{w}_0$. All methods under comparison are informed with this initial solution to start the optimisation.



(a) Forza (b) Allondaz

Fig. 3. The race tracks for the experiments (Source: *Speed Dreams*)

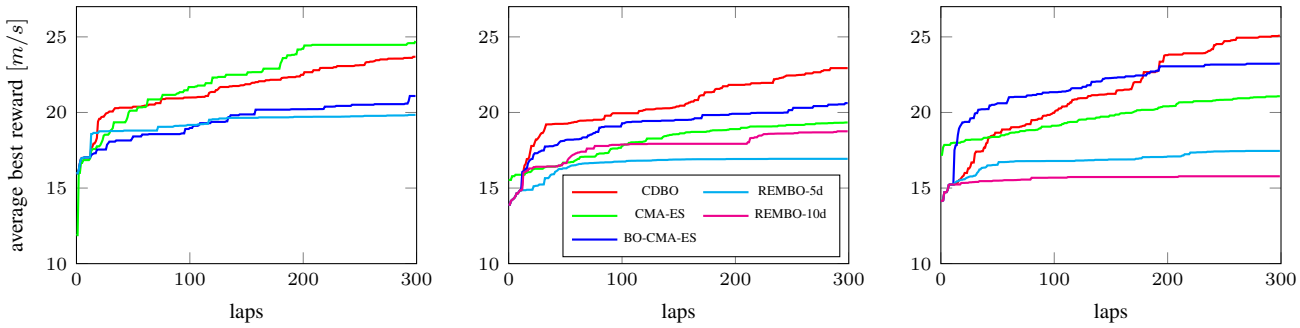
B. Results

We tested the algorithms on two different tracks. Each one of them was given a budget of 300 policy evaluations. Each policy evaluation corresponds to one lap. In the case of BO and REMBO, the first 10 laps correspond to the initial samples (Section IV-B). Before each policy evaluation, to optimise the acquisition function, all versions of BO were allowed a maximum of 50,000 acquisition function evaluations, and had as starting point the best weights found up to that lap. To minimise randomness effects in the algorithms, each run of 300 laps was repeated 4 times, and the results were averaged.

The first track, called *Forza* by the game, was a relatively simple circuit inspired by a real race track in Monza, Italy. This track is 5,784 metres long and 11 metres wide over flat asphalted terrain. The critical parts of this track are the sharp curves at the bottom of Figure 3a, which in terms of x value, happen around 40% of the track.

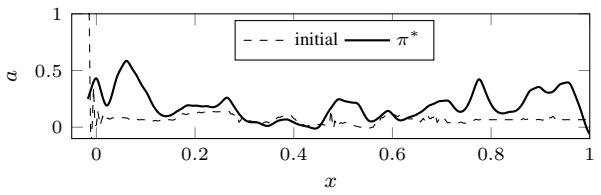
Figure 4 presents the overall performance of the analysed methods on *Forza* for policies with different numbers of kernels. These experimental results allow us to assess how each method handles the increase in dimensionality, which adds flexibility to the control policy, but makes its optimisation harder. As we can see, although finding better solutions in the 10 kernels setting, CMA-ES's performance severely degrades with the increase in dimensionality. When combined with BO (BO-CMA-ES), CMA-ES helps it to improve performance on average, but what we observed across individual runs is that this behaviour is actually bimodal: sometimes very good, and other times very bad. REMBO wasn't able to achieve good results for any number of kernels in this track. For the 100 kernel test, one of the problems with REMBO is clearly visible: if the used random embedding is not able to capture a (or if there isn't any) relevant subspace, the performance is poor. Overall, it is possible to see that CDBO's performance remains relatively stable with the increase in dimensionality and it is able to find better solutions for this policy search problem.

Figure 5 presents the best policy with 50-kernels obtained by our method and the resulting speeds the car achieved along the *Forza* track. From Figure 5b, it is possible to see

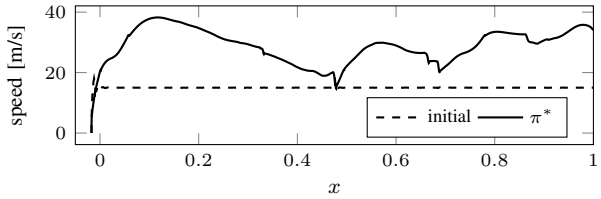


(a) Performance with 10 kernels (b) Performance with 50 kernels (c) Performance with 100 kernels

Fig. 4. Performance comparison w.r.t. dimensionality of the optimisation problem, evaluated on the *Forza* track.



(a) Control policy



(b) Resulting speeds

Fig. 5. Best policy obtained for track *Forza*

TABLE I

AVERAGE RUNTIME (IN SECONDS) ON *Forza*

Method/Dimensions	10	50	100
CDBO	127	281	318
CMA-ES	173	234	252
BO-CMA-ES	553	680	3048
REMBO-5d	154	242	257
REMBO-10d	-	244	299

that the algorithm adapts itself to speed up on the straight portions of the track and reduce speed close to curves, reducing the lap time.

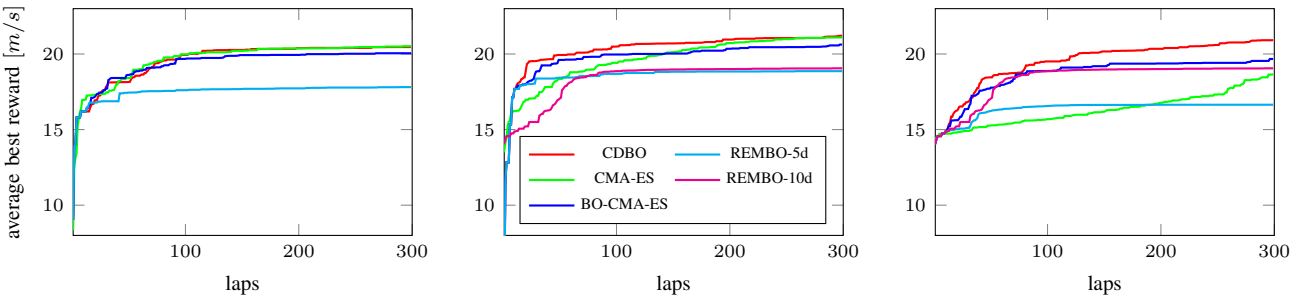
The second test track, *Allondaz*, shown in Figure 3b, is a road track with varying elevation along the path and filled with portions of complex geometry. It is 6,356 metres long and 12 metres wide, around the same dimensions as *Forza*. The overall performance of the methods is presented in Figure 6. In this track no algorithm was able to achieve average speeds as high as in *Forza*. Despite increasing the number of kernels to optimise, the best performing policy, achieved with CDBO, does not significantly change when using beyond 50 kernels. That's why, for this track, we only present results for setups with up to 50 kernels.

Similarly to the previous track, CMA-ES is not able to handle the high-dimensionality, and CDBO still maintains a consistent performance throughout the increase in dimensionality, demonstrating the capabilities of the method in high dimensions. Also, it's possible to see in all the tests that CDBO can achieve an even better result, if it is allowed to run for more iterations. REMBO achieved better results than the previous track, but it is still outperformed by BO-CMA-ES and CDBO. One interesting detail about REMBO's performance is that, after some iterations it does not improve any more, which means that it reached the optimum for the subspace used, and the global optimum is not in that subspace.

When compared to standard BO, another interesting feature of CDBO can be seen in Table I, which shows the runtime for each experiment (300 laps) on *Forza*. It is possible to see that with the increase in dimensionality, standard BO with CMA-ES (BO-CMA-ES) significantly increases in runtime when compared to the other methods. On the other hand, REMBO and our method maintain a low runtime through all the different problem dimensions. So, even if BO-CMA-ES achieves results close to CDBO, the runtime for the standard BO method is 10 times longer for the 100 dimensions case, which highlights the efficiency of the CDBO method for high-dimensional problems.

VI. CONCLUSION

In this paper, we presented a method to optimise control policies to allow a robot to complete a given race track faster, which is an instantiation of a more general class of problems involving delayed rewards and costly policy evaluations. Our method applies Bayesian optimisation to guide the exploration of the parameter space towards the optimal policy. By making use of ideas from randomised coordinate descent methods, optimising a function one dimension at a time in a randomised sequence, and by starting the search from a valid initial solution, our method is able to be effective in the high-dimensional acquisition function optimisation sub-problem. Experiments with a car racing simulator demonstrated that this relatively simple approach is able to outperform other state-of-the-art black-box optimisation and BO methods in complex scenarios, some times in a fraction of the time. As



(a) Performance with 10 kernels (b) Performance with 30 kernels (c) Performance with 50 kernels

Fig. 6. Performance comparison w.r.t. dimensionality of the optimisation problem, evaluated on the track *Allondaz*

future work, the model can be improved to deal with policies over higher-dimensional state and action spaces and to work together with other conventional motion planning algorithms.

REFERENCES

[1] A. G. Kupcsik, M. P. Deisenroth, J. Peters, and G. Neumann, "Data-Efficient Generalization of Robot Skills with Contextual Policy Search," in *AAAI Conference on Artificial Intelligence*, 2013.

[2] J. Kober, E. Oztop, and J. Peters, "Reinforcement learning to adjust robot movements to new situations," in *Proceedings of Robotics: Science and Systems*, Zaragoza, Spain, June 2010.

[3] T. Rizano, D. Fontanelli, L. Palopoli, L. Pallottino, and P. Salaris, "Global path planning for competitive robotic cars," in *IEEE Conference on Decision and Control*, Florence, Italy, 2013.

[4] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1:43 scale RC cars," *Optimal Control Applications and Methods*, vol. 36, pp. 628–647, 2015.

[5] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. Rehg, B. Boots, and E. Theodorou, "Information theoretic MPC for model-based reinforcement learning." in *Proceedings of the 2017 IEEE Conference on Robotics and Automation (ICRA)*, 2017.

[6] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep Reinforcement Learning framework for Autonomous Driving," in *IS&T International Symposium on Electronic Imaging*. IS&T, 2017, pp. 70–76.

[7] M. V. Butz and T. D. Lonneker, "Optimized sensory-motor couplings plus strategy extensions for the torcs car racing challenge," in *2009 IEEE Symposium on Computational Intelligence and Games*, 2009.

[8] S. Sanchez and S. Cussat-Blanc, "Gene regulated car driving: Using a gene regulatory network to drive a virtual car," *Genetic Programming and Evolvable Machines*, vol. 15, no. 4, 2014.

[9] M. P. Deisenroth, "A Survey on Policy Search for Robotics," *Foundations and Trends in Robotics*, 2013.

[10] E. Brochu, V. M. Cora, and N. de Freitas, "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning," University of British Columbia, Tech. Rep., 2010.

[11] R. Martinez-Cantin, N. de Freitas, A. Doucet, and J. A. Castellanos, "Active Policy Learning for Robot Planning and Exploration under Uncertainty," *Robotics: Science and Systems*, 2007.

[12] A. Wilson, A. Fern, and P. Tadepalli, "Using Trajectory Data to Improve Bayesian Optimization for Reinforcement Learning," *Journal of Machine Learning Research*, vol. 15, 2014.

[13] Z. Wang, M. Zoghi, F. Hutter, D. Matheson, and N. D. Freitas, "Bayesian Optimization in a Billion Dimensions via Random Embeddings," in *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence.*, 2013.

[14] C.-L. Li, K. Kandasamy, B. Poczos, and J. Schneider, "High Dimensional Bayesian Optimization via Restricted Projection Pursuit Models," in *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, 2016.

[15] Y. Nesterov, "Efficiency of Coordinate Descent Methods on Huge-Scale Optimization Problems," *SIAM Journal on Optimization*, vol. 22, no. 2, pp. 341–362, 2012.

[16] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. Cambridge, MA: The MIT Press, 2006.

[17] B. Schlkopf and A. J. Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. Cambridge, Mass: MIT Press, 2002.

[18] B. Chen, R. M. Castro, and A. Krause, "Joint Optimization and Variable Selection of High-dimensional Gaussian Processes," in *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, 2012.

[19] D. Ulmasov, C. Baroukh, B. Chachuat, M. P. Deisenroth, and R. Misener, "Bayesian Optimization with Dimension Scheduling: Application to Biological Systems," in *Computer Aided Chemical Engineering*. Elsevier Masson SAS, 2016, vol. 38.

[20] J. Djolonga, A. Krause, and V. Cevher, "High-Dimensional Gaussian Process Bandits," *Advances in Neural Information Processing Systems 26*, 2013.

[21] S. J. Wright, "Coordinate descent algorithms," *Mathematical Programming*, vol. 151, no. 1, jun 2015.

[22] Z. Marinho, A. Dragan, A. Byravan, B. Boots, S. Srinivasa, and G. Gordon, "Functional Gradient Motion Planning in Reproducing Kernel Hilbert Spaces," in *Robotics: Science and Systems (RSS)*, 2016.

[23] J. A. Bagnell and J. Schneider, "Policy Search in Kernel Hilbert Space," Robotics Institute, Carnegie-Mellon University, Pittsburgh, PA, Tech. Rep., 2003.

[24] N. A. Vien, P. Englert, and M. Toussaint, "Policy Search in Reproducing Kernel Hilbert Space," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)*, 2016.

[25] Q. V. Le, T. Sarlos, and A. Smola, "Fastfood Approximating Kernel Expansions in Loglinear Time," in *Proceedings of the 30th International Conference on Machine Learning*, vol. 28. Atlanta, Georgia, USA: JMLR: W&CP, 2013.

[26] H. Rosenbrock, "An automatic method for finding the greatest or least value of a function," *Comput. J.*, vol. 3, no. x, 1960.

[27] Y. Nesterov, "A method of solving a convex programming problem with convergence rate o (1/k2)," in *Soviet Mathematics Doklady*, vol. 27, no. 2, 1983.

[28] D. P. Bertsekas, *Parallel and Distributed Computation: Numerical Methods*. Athenas Scientific, 2015.

[29] M. J. D. Powell, "On search directions for minimizaiton algorithms," *Mathematical Programming*, vol. 4, 1973.

[30] B. Wymann, E. Espié, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner, "TORCS, The Open Racing Car Simulator," http://www.torcs.org, 2014.

[31] T. Rückstieß, F. Sehnke, T. Schaul, D. Wierstra, Y. Sun, and J. Schmidhuber, "Exploring parameter space in reinforcement learning," *Paladyn*, vol. 1, no. 1, 2010.

[32] D. V. Arnold and N. Hansen, "Active covariance matrix adaptation for the (1+1)-CMA-ES," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation - GECCO '10*, Portland, OR, 2010.

[33] M. Powell, "A view of algorithms for optimization without derivatives," Cambridge University DAMTP, Cambridge, United Kingdom, Tech. Rep., 2007.

[34] S. G. Johnson, "The nlopt nonlinear-optimization package," http://ab-initio.mit.edu/nlopt, 2014, accessed: 2016-08-16.