Continuous State-Action-Observation POMDPs for Trajectory Planning with Bayesian Optimisation

Philippe Morere[1,2], Roman Marchant[1] and Fabio Ramos[1]

Abstract— **Decision making under uncertainty is a challenging task, especially when dealing with complex robotics scenarios. The *Partially Observable Markov Decision Process* (POMDP) framework, designed to solve this problem, was subject to much work lately. Most POMDP solvers, however, focus on planning in discrete state, action and/or observations spaces, which does not truly reflect the complexity of most real world problems. This paper addresses the issue by devising a method for solving POMDPs with continuous state, action and observations spaces. The proposed planner, *Continuous Belief Tree Search* (CBTS), uses *Bayesian Optimisation* (BO) to dynamically sample promising actions while constructing a belief tree. This dynamic sampling allows for richer action selection than offline action discretisation. CBTS is complemented by a novel trajectory generation technique, relying on the theory of *Reproducing Kernel Hilbert Spaces* (RKHS), yielding trajectories amenable for robotics applications. The resulting trajectory planner kCBTS outperforms other continuous planners on space modelling and robot parking problems.**

I. Introduction

Planning in real world scenarios presents complex challenges. Robots performing in this setting often need to plan based on incomplete and limited information from the environment and simultaneously deal with continuous variables and their infinite dimensionality.

In many robotics tasks, robots start with very little knowledge of their environment and gather data as they move around. Newly gathered data improve planning quality by allowing robots to take better informed decisions. This improvement motivates online planning methods, which recompute policies whenever the robot executes an action, using an updated belief of the environment.

Bayesian optimisation was proposed as a planning method by [1] and extended to online informative path planning by [2]. While choosing the immediate best trajectory yields acceptable results, the myopic character of BO leads to suboptimal policies. Indeed, the same planning problem was reformulated as a POMDP by [3] to benefit from the framework's look-ahead planning capabilities, and shown to perform better than its myopic equivalent. However, the proposed non-myopic planner only addresses planning with a set of discrete actions. While discretising the action space is perfectly sound in engineered scenarios, it often leads to suboptimal policies in real world problems. Furthermore, action generation can be challenging in the continuous case.

This paper proposes a POMDP planner for fully continuous environments based on *Monte-Carlo Tree Search* (MCTS). MCTS is an approximate tree-based method proposed by [4], able to handle continuous states and observations but limited to discrete actions. Our contributions are twofold. We first present CBTS, an extension of MCTS, for planning on continuous action spaces, which relies on dynamic action sampling. Because dynamic sampling favours promising regions of the action space, it allows finding and selecting more precise actions than traditional sampling techniques, consequently yielding better policies. Our second contribution is a kernel-based trajectory generation method in RKHS. Trajectories are easily generated from parameter optimisation, and display consistent properties such as smoothness and differentiability, which are enforced by the kernel functions. These properties are often desired when generating realistic trajectories for real world robotics. Our planning algorithm, CBTS, and the proposed kernel trajectory generation technique are complementary; they are combined into a trajectory planner we call kCBTS.

The final algorithm, kCBTS, is validated on simulated and real robotics systems. kCBTS is first applied to a space modelling problem in which a robot learns an objective function by gathering noisy measurements along continuous trajectories. The robot maintains a belief over the studied function using a Gaussian Process, and monitoring behaviour is achieved by defining a specific reward function balancing exploration and exploitation of high-valued areas. The planning algorithm is then used to solve a simulated parking problem, in which a robot must manoeuvre to a park with restricted steering angle. Lastly, kCBTS is employed on a real-world robotics problem analogous to the previous parking task, validating the practical applicability of kernel trajectories. Experiments show our method outperforms other existing trajectory planners, while confirming that planning with continuous actions results in higher accumulated reward than when using a discrete set of actions. Additionally, continuous actions allow for better space coverage, resulting in lower errors in final models of the monitored function.

II. Related Work

The planning problem in POMDPs has received much attention over the last decades. Classic planning methods compute a *policy*, a mapping from states to actions, before experiments start. Such offline planners, presented in the work of [5], generally rely on sampling techniques such as the point-based approach, trading off optimality for speed. In most realistic scenarios, new information is gathered as robots interact with their environment, which offline methods do not take advantage of. Unlike online techniques, offline

[1]The University of Sydney [2] DATA61. {philippe.morere, roman.marchant, fabio.ramos}@sydney.edu.au

planners are typically computationally expensive and don't improve their policies over time.

Many successful online POMDP planners have been developed. While [6], [7] and [8] presented basic planners restricted to discrete and low-cardinality states, actions and observations, some were extended to larger and continuous state and observation spaces in [9], [4], and [10]. Most of these techniques sample spaces to compute more compact representations. Random sampling leads to expectation estimates for states and observations. However, finding actions yielding maximum reward is not an expectation problem and hence cannot be tackled with the same sampling techniques.

Several techniques were proposed by [11], [12], [13] and [14] to plan in POMDPs with continuous actions. While successful in their case studies, these methods rely on diverse assumptions. [11] constrain finding the best policy of a predefined class, greatly restricting policy outcomes. Assuming beliefs to be Gaussian distributions as in [12] limits the range of applications. [13] assumes to receive the most likely observations, which does not result in a true state estimate. The method of [14] relies on basic heuristics to search the continuous action space, which converge to local minima. This paper proposes an approximate and online POMDP solver for continuous state, action and observation spaces, which does not restrict the type of belief nor makes strong assumptions on the nature of observations.

In robotics applications, generating continuous and plausible trajectories is paramount. Much work uses waypoints given by classic planners such as RRT. [15] makes robots navigate a sequence of waypoints, and [16] need to further work out robot kinematics. The first approach is not realistic and does not consider robot capabilities, and the second requires extensive problem-specific knowledge and computation to match robot kinematics. Cubic splines provide smoother trajectories [2], but often result in unrealistic steering angles. Bezier curves use controlling points to constrain curvature, yielding more realistic trajectories [17]. However, they require many parameters to tune and do not necessarily reflect the robot's physical constraints. Recently, kernels were used to define trajectories with desired properties, when a cost function is well defined [18]. In this work, we propose a method to build realistic trajectories using RKHS. These trajectories are easily generated from a set of parameters, adapting to the limitations and capabilities of specific robotic platforms. Their execution requires no further tuning.

The techniques presented in this paper are applied to spatial modelling, a problem which has received increased attention over the past few years. Air pollution was successfully monitored in the work of [2] using a ground robot and an online myopic planner to generate trajectories from a continuous space of actions. [3] then extended this work to non-myopic planning by solving a POMDP using MCTS, then applied to environmental mapping with UAVs in [19]. However, this technique only handles discrete actions. Our method shows planning with continuous action POMDPs for space modelling yields superior results compared to planning with previously discretised actions.

III. Method

A. Overview

We wish to achieve non-myopic planning in fully continuous and partially observable environments. We formulate the problem as solving a POMDP, and describe MCTS as a solution in Section III-B. MCTS simulates sequences of trajectories, effectively building a belief tree by iteratively selecting branches to expand. We generalise MCTS to continuous actions by incorporating a continuous optimisation method at each node. The proposed planner *Continuous Belief Tree Search* (CBTS), is described in Section III-C. We then present a kernel trajectory generation technique in Section III-D which directly handles action parameters optimised by CBTS. The combination of kernel trajectory generation and CBTS, defined as kCBTS, is described in Section III-E.

B. Belief tree search

We formulate trajectory planning as solving a Partially observable MDP, a well-defined framework for non-myopic decision making under uncertainty. A POMDP is a tuple $< S, A, T, R, \Omega, O, \gamma >$ where S, A, Ω are spaces of states, actions and observations respectively. At each step t, the agent arrives at state $s' \in S$, receives a reward $r \in \mathbb{R}$ and an observation $o \in \Omega$ for taking action $a \in A$ in its previous state s at step $t-1$. The transition dynamics distribution T satisfies the Markov property, expressing the probability of transitioning to state s' when executing action a in state s, $T(s, a, s') = p(s'|s, a)$. Rewards r are given by a reward function R which only depends on the current state and action, $r = R(s, a)$. The observation distribution O expresses the probability of observing o when executing action a resulting in state s', $O(o, a, s') = p(o|a, s')$. $\gamma \in [0, 1]$ is a user-defined long-term reward discounting parameter.

Given the partial observability of the problem, agents do not have access to the true state of the environment. Instead, they rely on maintaining a belief $b \in B$ over possible current states. POMDP planners compute policies $\pi : B \rightarrow A$ reflecting the action an agent should take when in a belief state. Solving a POMDP is equivalent to finding the optimal policy π^*, maximising the expected infinite sum of future discounted rewards,

$$\pi^* = \arg\max_{\pi} E\left[\sum_{t=0}^{\infty} \gamma^t r_t^{\pi} | b_o\right], \qquad (1)$$

where b_0 is the initial belief, and r_t^{π} is the reward for executing policy π at time t. Diverse techniques were proposed to solve POMDPs [7]; we focus here on a stochastic tree search method.

Monte-Carlo Tree search was first used by [4] to plan in POMDPs. MCTS is an any-time method used to partially and stochastically search trees. As a POMDP solver, it builds a tree in which nodes are beliefs and branches are actions. The tree represents numerous sequences of simulated actions an agent can take at step t. Accumulated rewards and visit counts are kept on each branch. Finding the branch with the maximum accumulated reward approximates Equation 1, with the infinite sum replaced by a finite one.

Unlike full tree search, MCTS does not require all branches to be explored. MCTS approximates the true value of an action by simulating sequences of random actions, and computing the empirical mean of their values. Exploration of new branches is guided by an acquisition function. A popular choice is the *Upper Confidence bound for Trees* (UCT), where a branch a_i from v is selected when maximising

$$UCT(v, a_i) = \frac{r_i}{n_i} + c\sqrt{\frac{\ln t}{n_i}}. \tag{2}$$

r_i and n_i are the accumulated reward and visit count of a_i respectively, c is an exploration parameter, and t is the visit count of v. The first term in Equation 2 favours branches with high accumulated reward (exploitation), while the second one encourages evaluating neglected branches (exploration).

C. Continuous Belief Tree Search (CBTS) with BO

MCTS is only defined to solve POMDPs with finite and discrete actions. In this paper, we propose to generalise the method to infinite and continuous action spaces. CBTS is an approximate tree search algorithm based on PO-UCT [4], a variant of MCTS which was proven to plan efficiently in large discrete POMDPs. CBTS extends PO-UCT to the case of planning with continuous actions, alleviating the need to discretise the action space prior to planning. Classic MCTS partially explores a belief tree by choosing promising actions from the discrete set of available actions, maximising Equation 2. CBTS extends this maximisation problem to continuous action spaces, by only exploring a subset of sampled actions. Choosing this subset of actions is crucial, as it directly impacts the accumulated reward of all subsequent branches of the belief tree. The method proposed here relies on dynamically sampling the space of actions at the most promising locations with Bayesian optimisation.

Bayesian Optimisation (BO) is a technique for finding the optimum $\hat{\Theta} \in \mathbb{R}^d$ of an objective function $f : \mathbb{R}^d \to \mathbb{R}$, particularly when one does not have access to f or it is expensive to evaluate. BO iteratively determines the best Θ at which to evaluate f at step t, based on previous noisy data $\mathcal{D} = \{(\Theta_i, r_i)\}_{i=1}^{t-1}$, where $r_i = f(\Theta_i) + \epsilon$ is a noisy evaluation of f and $\epsilon \overset{iid}{\sim} \mathcal{N}(0, \sigma_n^2)$ is Gaussian noise. Most BO implementations represent f as a Gaussian process, which can express a wide range of functions, trained on $\mathcal{D}$. The search of where to evaluate f next is guided by an acquisition function h. The optimisation problem is transferred from f to h because h is known and cheap to evaluate. Traditional optimisation techniques are used to find $\arg\max_\Theta h(\Theta)$. Acquisition functions typically balance exploration of the input space and re-sampling high-valued areas, effectively guiding the search towards optima.

We adapt Bayesian optimisation to action selection, defining actions by a vector of parameters Θ. The problem of choosing new action parameters Θ^* for simulation from node v is formulated as follows:

$$\Theta^* = \arg\max_{\Theta} h(\Theta | \mathcal{D}_v), \tag{3}$$

Algorithm 1 Bayesian Optimisation for action selection

```
1: Let $\mathcal{D}_v = \{\}$ be the data, $h$ be an acquisition function.
2: for $t = 1, 2, 3, ...$ do
3:     Find $\Theta^*$ with Eq. 3 or 4.
4:     Generate trajectory $\mathcal{T}$ from $\Theta^*$ (Sec. III-E).
5:     $r, o \leftarrow$ Simulate $\mathcal{T}$.
6:     Augment $\mathcal{D}_v$ with $(\Theta^*, r)$ and recompute GP.
7: end for
```

where $\mathcal{D}_v$ is a node-specific dataset. Balancing exploitation of high-reward action and exploration of unknown areas of the action space is achieved by appropriately choosing h. The Upper-Confidence Bounds (UCB) function is generally used for such balance in the BO literature. Equation 3 then becomes

$$\Theta^* = \arg\max_{\Theta} \mu(b_v(\Theta)) + \kappa\sigma(b_v(\Theta)), \tag{4}$$

where μ and σ are the mean and variance operators respectively, κ is a parameter balancing exploration and exploitation, and b_v is a belief over the action-reward mapping at a node level maintained using a Gaussian Process trained with $\mathcal{D}_v$. Gathering data to generate $\mathcal{D}_v$ is done as follows. When an action a is simulated from node v of the belief tree τ and yields reward r, the resulting pair $\{a, r\}$ is used to learn a GP mapping from actions to rewards at a node level.[1]. Note that the variance term of equation 4 encourages selecting actions furthest from that in $\mathcal{D}_v$, ensuring efficient coverage of the action space. Algorithm 1 shows a practical implementation of BO for action selection.

We now incorporate BO for action selection to MCTS to handle continuous action spaces and ensure non-myopic planning. We call the resulting algorithm *Continuous Belief Tree Search* (CBTS). Branches to explore are first determined by classic MCTS branching metric (Eq. 2), and BO chooses actions to simulate. Each node v stores data $\mathcal{D}_v$ of previous action-reward pairs, which are used by BO. Pseudo-code for CBTS given in Algorithm 2 details how MCTS and BO complement each other. As nodes are revisited, their action-rewards mappings get more accurate. In practice, mappings are often accurate enough with few data points. Therefore, one can limit the maximum number of generated actions per node to a problem-specific fixed value A_{max}, as shown in Algorithm 2 line 12. Alternatively, a node's action generation could stop whenever a convergence criterion is met. For example, such criterion can be $||\Theta_{i-1} - \Theta_i|| < \delta$, where Θ_{i-1} and Θ_i are the previously and newly generated actions respectively, and δ is a user-specified threshold.

Each node's belief on the action-reward mapping is implemented with a Gaussian Process, of complexity $O(N^3)$, where N is the number of actions generated at a given node. N is in practice very small, therefore leading to

[1]A mapping from actions to Monte-Carlo estimates of return could easily be learnt instead, by using a GP with heteroscedastic noise model to reflect the fact that returns do not have constant variance across the action space. However, using rewards in lieu of returns leads to good results in practice.

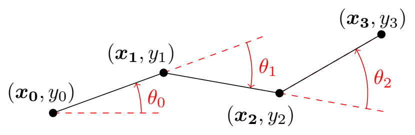
$(\boldsymbol{x_0}, y_0)$
$(\boldsymbol{x_1}, y_1)$
$(\boldsymbol{x_2}, y_2)$
$(\boldsymbol{x_3}, y_3)$
θ_0
θ_1
θ_2

Fig. 1: Picking data points used to construct trajectories in RKHS. This example uses 4 anchor points (including the stating point) and only requires 3 parameters $\theta_0, \theta_1, \theta_2$. Times y_i are evenly distributed between 0 and 1.

negligible computation time compared to the state belief update whose complexity typically grows faster, with the number of observations. In practice, the complete CBTS algorithm displays similar running times to using MCTS with discrete actions.

D. Kernel Trajectories

We now describe a technique for generating realistic robot trajectories based on the theory of *Reproducing Kernel Hilbert Spaces* (RKHS) presented by [20] and [21]. Resulting trajectories encode the required characteristics (eg. smoothness, acceleration, etc) by making use of kernel functions.

Trajectories are defined as functions $\mathcal{T}(\Theta, \boldsymbol{x})$ of $y \in [0,1] \to \mathbb{R}^d$ parameterised by a vector Θ, where d is the dimension of the physical space, so that $\mathcal{T}(\Theta, \boldsymbol{x})|_{y=0}$ is the starting pose $\boldsymbol{x}$ and $\mathcal{T}(\Theta, \boldsymbol{x})|_{y=1}$ is the ending pose $\boldsymbol{x'}$. This formulation does not make any assumption on the type of function used to represent trajectories.

We formulate trajectory generation as finding the agent's pose $\boldsymbol{x}$ at any time $y \in [0,1]$ given pose-time pairs $D = \{(\boldsymbol{x_i}, y_i)\}_{i=0}^{n}$. More formally, a trajectory $\mathcal{T}$ can be defined as

$$\mathcal{T}(y) = \mathbb{E}[X|Y=y], \quad \forall y \in [0,1], \tag{5}$$

were X and Y are random variables in the spaces of poses $\mathcal{X}$ and times $\mathcal{Y}$ respectively, and y is an element of $\mathcal{Y}$. Generating a trajectory simply comes down to producing waypoint data D, as explained in the next section, and sampling Equation 5 for different times $y \in [0,1]$.

Equation 5 can be solved by using an equivalent kernel formulation. Let $\mathcal{H}_x$ and $\mathcal{H}_y$ be two RKHS defined over spaces $\mathcal{X}$ and $\mathcal{Y}$ respectively, and fully defined by their reproducing kernels k_x and k_y. One can then compute the kernel mean $m_P \in \mathcal{H}_x$ of any arbitrary probability distribution P on $\mathcal{H}_x$ with

$$m_P = \int k_x(\cdot, \boldsymbol{x}) dP(\boldsymbol{x}). \tag{6}$$

Provided conditions on kernel k_x and k_y are met, the mapping $P \to m_P$ is one-to-one, and one can approximate m_P to estimate P from the available data [22].

Here, we are interested in finding an estimate of the kernel posterior mean $m_{X|y}$, the kernel mean of the posterior distribution $P(X|Y=\cdot)$, defined as

$$\hat{m}^{\pi}_{X|y} = \sum_{i=1}^{n} \omega_i^{(y)} k_x(\cdot, \boldsymbol{x_i}), \tag{7}$$

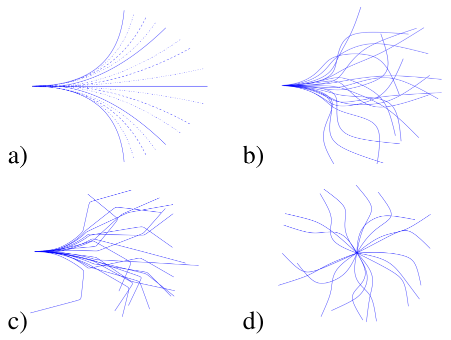

Fig. 2: Trajectories generated with discrete cubic splines (a). Plain splines are used for experiments with 5 actions, plain and dashed for 9 actions, dotted splines are added to form 17 actions. Kernel trajectories with smooth (b, d) and sharp (c) space kernel k_x. The starting angle restriction is relaxed to allow different types of motion (d).

where $\boldsymbol{\omega}^{(y)} = \{\omega_i^{(y)}\}_{i=1}^{n}$ is a weight vector depending on the query time y, data D, kernels k_x and k_y, and a prior π on $\mathcal{X}$. $\boldsymbol{\omega}^{(y)}$ is computed using kernel Bayes' rule, first proposed by [23]. We refer readers not familiar with kernel embeddings to the theory of kernel embeddings of conditional distributions described by [20]. The expected pose $\mathcal{T}(y)$ defined in Equation 5 can simply be estimated as a weighted sum of data points:

$$\mathcal{T}(y) = \mathbb{E}[X|Y=y] \approx \sum_{i=1}^{n} \omega_i^{(y)} \boldsymbol{x_i}, \tag{8}$$

$$\text{where } \boldsymbol{\omega}^{(y)} = \Lambda G_Y((\Lambda G_Y)^2 + \delta \mathbb{I})^{-1} \Lambda \boldsymbol{k}_Y, \tag{9}$$

$$\Lambda = diag((G_X + n\epsilon \mathbb{I})^{-1} \boldsymbol{m}_\pi), \tag{10}$$

and $G_X = (k_x(\boldsymbol{x_i}, \boldsymbol{x_j}))$, $G_Y = (k_y(y_i, y_j))$, $\boldsymbol{k}_Y = \{k_y(y, y_i)\}_{i=1:n}^{T}$, $\boldsymbol{m}_\pi = \{\frac{1}{l} \sum_{j=1}^{l} k_x(\boldsymbol{x_i}, \boldsymbol{u_j})\}_{i=1:n}^{T}$, $\boldsymbol{u_i}$ are i.i.d. drawn from π, and $\epsilon, \delta > 0$ are regularisation constants.

In practice, to generate a trajectory in RKHS, one needs to (i) generate pose-time pairs $(\boldsymbol{x_i}, y_i)$ as described in the upcoming example, (ii) compute $\boldsymbol{\omega}^{(y)}$ at discretised times with Equations 9 - 10, and (iii) estimate $\mathcal{T}$ with Equation 8.

E. Kernel CBTS

To generate kernel trajectories, one needs to define a set of anchor points to interpolate between. Pairs of anchor points and corresponding trajectory times $(\boldsymbol{x_i}, y_i)$ can be chosen in various ways. Parameter vector Θ given by Equation 4 and a starting pose $\boldsymbol{x}$ are sufficient to generate anchor points and define a kernel trajectory. Our contributions are combined as kCBTS in Algorithm 2; CBTS selects trajectory parameters Θ which are then converted to kernel trajectories with Equations 8 - 10, ready for execution.

We now present a simple and efficient method for generating anchor points. Assuming that a robot moves at constant speed on a plane, we define a set of equally spaced anchor poses $\boldsymbol{x_i}$ with associated times y_i evenly distributed between 0 and 1. In polar coordinates, one only needs to choose angles θ_i between two consecutive trajectory segments to fully describe the set of anchor points. See Figure 1 for a graphical explanation. This technique is efficient as it only requires one parameter per anchor point, effectively reducing the size of the action space the planner needs to search.

To compute $\boldsymbol{\omega}^{(y)}$ with Equations 9-10, parameters need be specified. In our experiments, a uniform prior π on $\mathcal{X}$ is chosen, therefore not penalising nor rewarding any part of the pose space. Note that other priors might be useful to encode obstacles, or to restrict specific moves as in the case of robotic manipulators. RBF kernels are used for both k_x and k_y for their smoothness, inducing desired properties such as low steering angles and bounded acceleration; Figure 2 shows a set of trajectories generated following this method with different parameter values.

Algorithm 2 Kernel CBTS (kCBTS)

```
1: function $a^*$ = KCBTS($b, depth_{max}$)
2:     $v_0 = NewNode(b, 0)$.
3:     for $i = 0$ to {Max CBTS iterations} do
4:         $v_l \leftarrow TreePolicy(v_0)$.
5:         $r \leftarrow$ Random actions from $v_l$ until $depth_{max}$.
6:         $BackUp(v_l, r)$.
7:     end for
8:     $a^* \leftarrow$ action from $v_0$ with max return.
9: end function
10: function $v$ = TREEPOLICY($v$)
11:     while $depth(v) \leq depth_{max}$ do
12:         if $length(\mathcal{D}_v) < A_{max}$ then
13:             Run one iteration Alg. 1 with $\mathcal{D}_v$.
14:             Collect $r, o$ and $\mathcal{D}_v$.
15:             Update $b$ with o, and $b_v$ with $\mathcal{D}_v$.
16:             return $v = NewNode(b, r)$.
17:         else
18:             $v = BestChild(v)$.
19:         end if
20:     end while
21: end function
22: function BACKUP($v, r$)
23:     while $v \neq v_0$ do
24:         Increase visited counter for $v$.
25:         Increase accumulated reward for $v$ with $r$.
26:         $v \leftarrow Parent(v)$.
27:     end while
28: end function
29: function $v_c$ = BESTCHILD($v_p$)
30:     $V \leftarrow$ Children of $v_p$.
31:     for $v_i \in V$ do
32:         $N_p, N_i \leftarrow$ Visited counter of $v_p$ and $v_i$.
33:         $R_i \leftarrow$ Accumulated reward.
34:         $g(i) = \frac{R_i}{N_i} + \kappa_{MC}\sqrt{\frac{2\ln(N_p)}{N_i}}$.
35:     end for
36:     $v_c \leftarrow \arg\max_{v_i \in V} g(i)$.
37: end function
```

In general, the choice of kernels k_x and k_y directly impacts physical properties of generated trajectories. Slowly varying kernels ensure trajectory smoothness, and conversely, sharp kernels result in spiky trajectories. Space constraints such as steering angle are determined by the space kernel k_x while velocity and acceleration are governed by the time kernel k_y. Note that the regularisation parameters from kernel Bayes' rule also impact the nature of trajectories. Further work is needed to draw rigorous conclusions on the impact of kernel types and parameter values. Lastly, because these choices are problem dependent, they are to be made offline by the user.

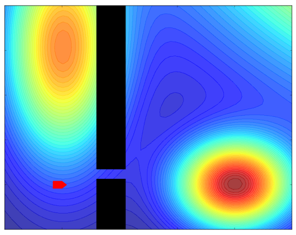
Fig. 3: Space modelling domain. Black rectangles are obstacles, the red polygon represents the robot's starting pose, and the background colours show the monitored function (ground truth). High values are red, low values are blue.

IV. Experimental Results

In this section, we present experiments of kCBTS on an environmental monitoring task, and on a robot parking problem in both simulation and real world.

A. Space Modelling Problem

In this experiment, a simulated robot monitors an unknown noisy environmental variable. The robot's task is to learn the monitored function and build the best possible spatial model in a limited time. The robot starts with no data and progressively enriches its belief of the monitored function by gathering observations along trajectories. Each observation is a noisy evaluation of the monitored function at the robot's position. The domain shown in Figure 3 features two high-valued areas separated by a wall, and was designed so that the robot can change areas by moving through a narrow corridor. The robot is given full knowledge of obstacle locations, but only receives information about the monitored function from noisy observations. Exploration is therefore essential to receiving higher long-term rewards.

1) POMDP Formulation: This formulation was proposed by [3], enabling a robot to carry out exploration guided by an acquisition function while benefiting from the non-myopic character of POMDP planners. The POMDP used is:

- S: States $s = \{f, \boldsymbol{x}\}$ with f the objective function and $\boldsymbol{x}$ the robot's pose. While the robot knows its pose, it only gets information about f from observations o.
- A: Continuous actions a are sets of parameters Θ defining kernel trajectories $\mathcal{T}(\Theta, \boldsymbol{x})$ starting from $\boldsymbol{x}$.
- T: The transition distribution $T(\{f, \boldsymbol{x}\}, \Theta, \{f', \boldsymbol{x}'\})$ models deterministic moves from $\boldsymbol{x}$ to $\boldsymbol{x}'$ with action Θ. Because transitions and f are independent, T is written as $T(\{f, \boldsymbol{x}\}, \Theta, \{f', \boldsymbol{x}'\}) = \delta(\mathcal{T}(\Theta, \boldsymbol{x})|_{y=1} - \boldsymbol{x}')$.

- R: The reward function for action Θ in pose $\boldsymbol{x}$ is:

$$r(\Theta, \boldsymbol{x}) = \int_0^1 h(\mathcal{T}(\Theta, \boldsymbol{x})|_{y=t})dt, \quad (11)$$

where h is an acquisition function yielding the desired monitoring behaviour. Note that h depends on the belief $b(f)$ the robot maintains in lieu of f. In practice, Equation 11 is approximated with discrete simulated rewards:

$$R(\Theta, \boldsymbol{x}) = \sum_{x \in \mathcal{T}(\Theta, \boldsymbol{x})} h(x) + cost(\Theta, \boldsymbol{x}), \quad (12)$$

where $cost(\Theta, \boldsymbol{x})$ is the application specific cost of moving along $\mathcal{T}(\Theta, \boldsymbol{x})$, e.g. colliding with a wall results in a -100 cost. UCB is often selected as the acquisition function h for the exploration behaviour it yields.
- Ω: Observations $o \in \mathbb{R}$ are noisy evaluations of f, typically sensor measurements. Similarly to rewards, observations are computed on a set of locations along $\mathcal{T}(\Theta, \boldsymbol{x})$.
- O: The observation function O generates a set of observations $\{o_i\}$ along the robot's trajectory. As noisy evaluations of f, observations only depend on the robot's pose:

$$O(\{o_i\}, \Theta, \{f, \boldsymbol{x}\}) = \prod_{x_i \in \mathcal{T}(\Theta, \boldsymbol{x})} p(o_i | f(x_i)). \quad (13)$$

The robot can simulate observations by generating noisy samples from its belief b.

The belief over f is maintained using a GP, allowing flexibility in the type of modelled functions that can be represented. GPs allow one to define a prior belief b_0 over the monitored function, and gives mean and uncertainty information for f.

2) *Results:* We compare kCBTS to other POMDP planning methods on the previously described continuous state-action-observation POMDP. Compared methods are a discrete action planner (SBO) [3], and a recent approximate POMDP solver compatible with continuous action spaces (GPS-ABT) [14]. For reference, we also included a random planner choosing discrete actions uniformly. Discrete cubic splines trajectories are used by SBO and Random, as shown in Figure 2a. GPS-ABT and kCBTS use smooth kernel trajectories with constrained starting angle, see Figure 2b. Because SBO and kCBTS bear similar planning algorithms, comparisons between the two methods underline advantages of planning with kernel trajectories compared to discrete ones. Conversely, GPS-ABT and kCBTS both use kernel trajectories, and comparisons only showcase differences in planning performance.

In this experiment, the acquisition function h in Equation 12 is UCB, with parameter $\kappa = 10$. Figure 4 reports accumulated rewards for each of the four methods, averaged over 200 episodes. Overall, kCBTS and GPS-ABT perform better than discrete action planner SBO, regardless of the number of discrete actions used. Several numbers of discrete actions were tested $|A| \in 5, 9, 17$, but for clarity only the best case is reported. The first 8 steps however, yield very similar

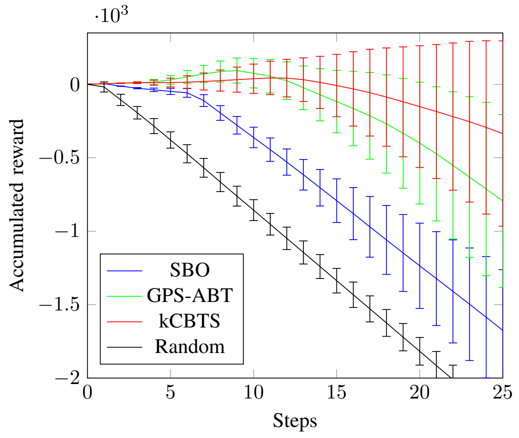

Fig. 4: Mean and standard deviation of accumulated rewards in the space modelling problem, averaged over 200 runs.

TABLE I: Errors on Final Model Reconstruction.

Algorithm	$\lvert A \rvert$	RMSE	WRMSE	runtime (s)
SBO	5	72.9	61.9	106
SBO	9	52.5	44.7	463
SBO	17	52.8	44.9	734
kCBTS	-	**42.6**	**36.9**	356

rewards across all methods because the robot first explores the left side of the domain. After step 8, the robot is drawn to explore the rest of the domain, and discrete actions are often not precise enough to navigate through the corridor. This experiment shows that planning with kernel trajectories enables robots to efficiently avoid obstacles and navigate more precisely than when planning with discrete actions. Figure 5 shows an example of trajectories using kCBTS and SBO. kCBTS can generate actions precise enough to navigate through the corridor and reach the rightmost area whereas actions available to SBO prevent the robot from navigating efficiently. Indeed, When in a tricky situation, discrete actions do not allow the robot to manoeuvre and lead to collisions. Lastly, kCBTS and GPS-ABT yield very similar performance on this problem, with a slight long-term advantage in favor of the CBTS planning technique.

At the end of each experiment, robots build a spatial model of the monitored function. Final models are compared to ground truth based on two error metrics: the root-mean-square error (RMSE) reflects how far from ground truth the belief is at each point in average; the weighted root-mean-square error (WRMSE) is weighted so that errors in high-valued locations are penalised. Note that because GPS-ABT does not build a model of the monitored function, it is omitted from the comparison. Table I shows kCBTS significantly outperforms SBO on both metrics regardless of the number of actions used. Overall, kCBTS yields similar running times to SBO, achieving superior performance in both accumulated rewards and model error.

SBO with 5 actions outperformed its equivalent with 9 and 17 actions both in terms of accumulated rewards and model error. While one might expect improvements for

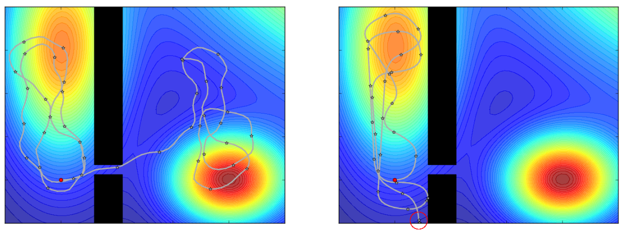

Fig. 5: Example of robot trajectories when using kCBTS (left) and SBO (right). With kCBTS, the robot manages to navigate through the wall gap. The red circle on the right figure denotes a collision.

increasing the number of actions available to the planner, this experiment shows the opposite. We believe this is due to the planner's inability to efficiently construct a belief tree with higher branching factors, given a fixed iteration budget. Better performance could be achieved by exponentially increasing the number of MCTS iterations.

When using a number of discrete actions similar to the number of continuous actions A_{max} generated with BO, kCBTS outperforms SBO. Indeed, BO takes advantage of the continuity of the action-to-reward mapping: very similar actions often result in similar rewards. This property allows BO to find optimal actions before trying A_{max} actions, therefore reducing the belief tree branching factor.

B. Robot Parking Problem

The robot parking problem is a domain in which a robot navigates in a two-dimensional space to a designated parking area. The state space is the space of positions within a delimited area, augmented with the robot's angle $S = [-4,4]^2 \times [0, 2\pi)$. Starting from a random location and angle, the robot's goal is to reach terminal state $(0,0,0)$ with a maximum of 15 steps by moving along continuous trajectories. Observations are noisy robot poses, and actions are defined as continuous trajectories in the ground plane, of constant velocity. All trajectories are normalised to have constant length, regarding of the trajectory type used. Rewards are granted according to

$$R(s) = \begin{cases} -|sin(\beta)| - d_{max} - 1, & \text{if } |sin(\beta)| > \epsilon_\beta \\ -|sin(\alpha)| - d_{max}, & \text{else if } |sin(\alpha)| > \epsilon_\alpha \\ -d, & \text{else if } d > \epsilon_d \\ 1000, & \text{otherwise} \end{cases} \tag{14}$$

where α is the difference between the robot's angle and the objective's angle, β and d are the angle and distance between the robot and the objective respectively, d_{max} is an upper-bound on d, and $\epsilon_d = 0.1$, $\epsilon_\alpha = \pi/10$, and $\epsilon_\beta = \pi/10$ are parameters reflecting required parking precision. A reward of 1000 is given for successfully parking in the delimited area, whereas exiting the limits of the domains results in a reward penalty of -1000. Both events terminate the episode.

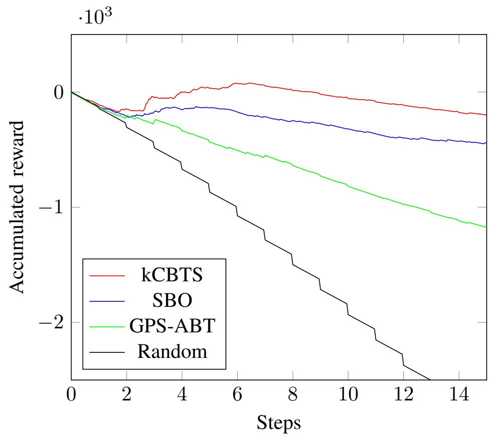

Fig. 6: Accumulated rewards in the simulated robot parking problem, averaged over 200 runs.

The difficulty of this domain arises from the robot's inability to turn on the spot, effectively constraining it to generate multi-step trajectories with a precise approach to reach the parking area. Experiments on this domain aim to highlight the advantage of continuous actions to manoeuvre in difficult scenarios.

1) Simulated results: Planner kCBTS is now compared to SBO, GPS-ABT and a random planner on a simulated version of the robot parking problem. The maximum MCTS budget is limited to 300, while the planning horizon is limited to 3 simulated steps for all algorithms. At each node of the tree search, kCBTS generates a maximum of $A_{max} = 20$ actions.

Figure 6 shows accumulated rewards for all planners on the robot parking problem starting from random poses, averaged across 200 episodes with random starting pose. Note that variance information is not included, as it mostly results from the randomness in starting poses.

kCBTS and SBO both manage to park properly in most episodes by step 6, as reported by the increased accumulated rewards at this point and steady decrease afterwards. However, GPS-ABT fails to park in most episodes while managing to keep within domain bounds. The random planner accumulates collision penalties and never manages to park. The performance difference between kCBTS and SBO shows the advantage of using continuous actions on this domain. Indeed, continuous actions yield a much richer spectrum of trajectories, allowing the robot to precisely navigate towards its end goal.

2) Robotics results: We now present results on the same robot parking task, this time applied to a real robot. The goal of this experiment is to demonstrate kernel trajectories are adapted to real robotics problems, and advantageous over classic cubic splines trajectories.

The platform used, shown in Figure 8, is a non holonomic wheeled ground robot, constrained to forward motion. The robot's position and orientation is computed by an external marker-based tracking system, and transmitted back to the robot's on-board computer; all other computations are made

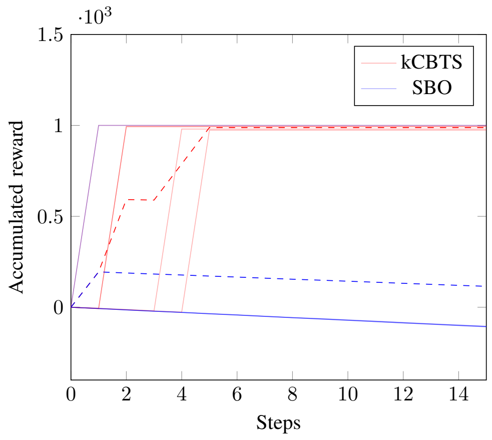

Fig. 7: Accumulated rewards for five runs (plain) and mean (dashed) in the robot parking problem.

on board, in real time. Planning times are around 3 seconds at each step. The operating area is reduced to 3 by 3 meters, and all other parameters are left unchanged from the simulation.

Figure 7 displays results on the robot parking domain for both kCBTS and SBO, averaged over 5 runs. SBO reached a successful parking pose in only one of the 5 runs, and circled around the parking location in other runs. This behaviour highlights the low expresivity of discrete trajectories, yielding poor manoeuvring. Conversely, kCBTS successfully parked on all runs, in a maximum of 5 steps. Although CBTS and SBO feature similar planning algorithms, results reflect the advantage of continuous trajectories. This experiments shows kCBTS is applicable to real robotics problems, by incorporating robot motion constraints to trajectory generation. Lastly, results demonstrate planning with kernel trajectories leads to enhanced robotic motion.

V. Conclusion

CBTS is a planner for continuous state-action-observation POMDPS, extending MCTS to continuous action spaces using BO. Our kernel-based technique to build trajectories, relying on the theory of RKHS, is combined with CBTS into trajectory planner kCBTS. Planned trajectories show properties important in robotics applications, such as smoothness and steering angle restrictions. kCBTS outperforms other POMDP planners on several simulated and real problems. Results show planning with CBTS and kernel trajectories yields better accumulated rewards and reduces modelling errors, while enabling robots to move smoothly and avoid obstacles.

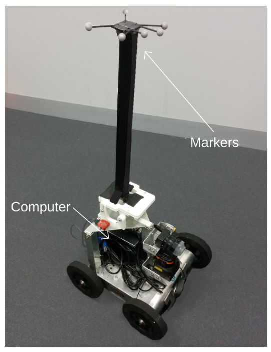

Fig. 8: Platform used in robot parking problem.

References

[1] R. Martinez-Cantin, N. de Freitas, A. Doucet, and J. A. Castellanos, “Active policy learning for robot planning and exploration under uncertainty.,” in *Robotics: Science and Systems*, 2007.

[2] R. Marchant and F. Ramos, “Bayesian optimisation for informative continuous path planning,” in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 2014.

[3] R. Marchant, F. Ramos, S. Sanner, *et al.*, “Sequential Bayesian optimisation for spatial-temporal monitoring.,” in *UAI*, 2014.

[4] D. Silver and J. Veness, “Monte-Carlo planning in large POMDPs,” in *Advances in neural information processing systems*, 2010.

[5] J. Pineau, G. Gordon, S. Thrun, *et al.*, “Point-based value iteration: An anytime algorithm for POMDPs,” in *IJCAI*, vol. 3, 2003.

[6] R. He, E. Brunskill, and N. Roy, “Puma: Planning under uncertainty with macro-actions.,” in *AAAI*, 2010.

[7] S. Ross, J. Pineau, S. Paquet, and B. Chaib-Draa, “Online planning algorithms for POMDPs,” *Journal of Artificial Intelligence Research*, 2008.

[8] H. Kurniawati, D. Hsu, and W. S. Lee, “SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces.,” in *Robotics: Science and Systems*, 2008.

[9] H. Kurniawati and V. Yadav, “An online POMDP solver for uncertainty planning in dynamic environment,” in *Robotics Research*, 2016.

[10] A. Somani, N. Ye, D. Hsu, and W. S. Lee, “Despot: Online POMDP planning with regularization,” in *Advances in neural information processing systems*, 2013.

[11] A. Y. Ng and M. Jordan, “Pegasus: A policy search method for large MDPs and POMDPs,” in *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, 2000.

[12] J. Van Den Berg, S. Patil, and R. Alterovitz, “Efficient approximate value iteration for continuous Gaussian POMDPs.,” in *AAAI*, 2012.

[13] R. Platt, R. Tedrake, L. Kaelbling, and T. Lozano-Perez, “Belief space planning assuming maximum likelihood observations,” in *Proceedings of Robotics: Science and Systems*, 2010.

[14] K. M. Seiler, H. Kurniawati, and S. P. Singh, “An online and approximate solver for POMDPs with continuous action space,” in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[15] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” 1998.

[16] Y. Kuwata, G. A. Fiore, J. Teo, E. Frazzoli, and J. P. How, “Motion planning for urban driving using RRT,” in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008.

[17] A. Sahraei, M. T. Manzuri, M. R. Razvan, M. Tajfard, and S. Khoshbakht, “Real-time trajectory generation for mobile robots,” in *Congress of the Italian Association for Artificial Intelligence*, 2007.

[18] Z. Marinho, A. Dragan, A. Byravan, B. Boots, S. Srinivasa, and G. Gordon, “Functional gradient motion planning in reproducing kernel Hilbert spaces,” *arXiv preprint arXiv:1601.03648*, 2016.

[19] P. Morere, R. Marchant, and F. Ramos, “Sequential Bayesian optimisation as a POMDP for environment monitoring with UAVs,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[20] L. Song, K. Fukumizu, and A. Gretton, “Kernel embeddings of conditional distributions: A unified kernel framework for nonparametric inference in graphical models,” *IEEE Signal Processing Magazine*, 2013.

[21] M. Kanagawa, Y. Nishiyama, A. Gretton, and K. Fukumizu, “Filtering with state-observation examples via kernel Monte Carlo filter,” *Neural computation*, 2016.

[22] J. Bourgain, “On Lipschitz embedding of finite metric spaces in Hilbert space,” *Israel Journal of Mathematics*, 1985.

[23] K. Fukumizu, L. Song, and A. Gretton, “Kernel Bayes’ rule,” in *Advances in neural information processing systems*, 2011.