An Uncertainty-aware Planning Approach to Robotized Liquid Pouring

Marco Faroni, Carlo Odesco, Andrea M. Zanchettin, Paolo Rocco

***Abstract*— Physics-based simulations and learning-based models are vital for complex robotics tasks like deformable object manipulation and liquid handling. However, these models often struggle with accuracy due to epistemic uncertainty or the sim-to-real gap. For instance, accurately pouring liquid from one container to another poses challenges, particularly when models are trained on limited demonstrations and may perform poorly in novel situations. This paper proposes an uncertainty-aware Monte Carlo Tree Search (MCTS) algorithm designed to mitigate these inaccuracies. By incorporating estimates of model uncertainty, the proposed MCTS strategy biases the search towards actions with lower predicted uncertainty. This approach enhances the reliability of planning under uncertain conditions. Applied to a liquid pouring task, our method demonstrates improved success rates even with models trained on minimal data, outperforming traditional methods and showcasing its potential for robust decision-making in robotics.**

I. Introduction

Physics-based simulation and learning-based models are extensively used in robotics to perform complex tasks such as deformable object manipulation [1]–[5], contact-rich manipulation [6]–[8], control of soft robots [9], [10], and liquid handling [11], [12]. These models are often inaccurate in predicting the outcome of actions (e.g., because of the epistemic uncertainty of learned models or the sim-to-real gap of physics simulators). A notable example is the liquid handling scenario in Fig. 1: A robot has to pour liquid from one container to another until it reaches a certain level. Ideally, one could accurately model the quantity of poured liquid based on the robot's tool motion (e.g., via fluid simulation), but this requires deep knowledge of the geometry and dynamics properties of the problem or the collection of large experimental datasets. Suppose a pouring model is built from a handful of pouring demonstrations. The model will be very inaccurate, especially for states and actions that are far from those seen in the training set. If the model is used within a planning algorithm, the robot may decide to take actions leading to inaccurate pourings.

If an estimate of the inaccuracy of the model predictions is available, this information can be embedded in the planning algorithm to avoid inaccurate states and actions [3], [13]. Interestingly, machine learning techniques commonly used for dynamics modeling in robotics come with an estimate of the model epistemic uncertainty at a given point. For example, the variance of a Gaussian Process (GP) is inversely proportional to the data density in a certain area of the feature space and can be seen as a proxy of the prediction uncertainty. Similarly, the variance of the predictions of Neural-Network ensembles measure the consensus of the networks: if the variance is large, the average prediction is likely to be inaccurate.

The authors are with the Department of Electronics, Information and Bioengineering, Politecnico di Milano, Italy. marco.faroni@polimi.it

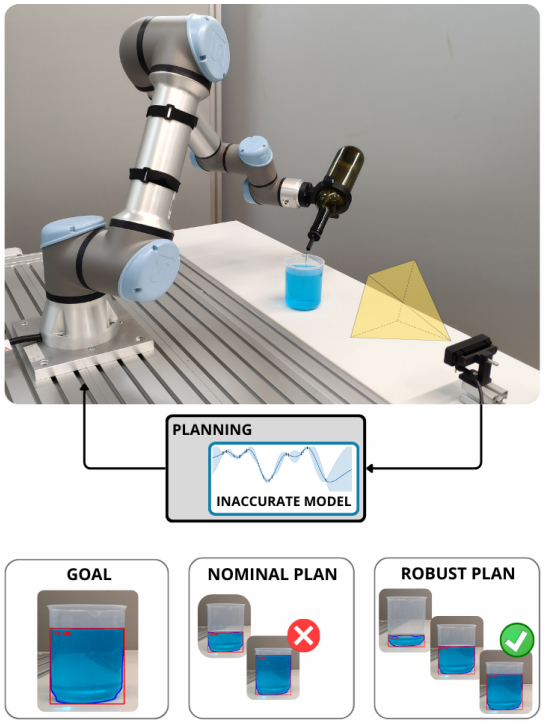


Fig. 1: A robotized pouring task: the robot arm shall pour liquid from one container to another until it reaches a reference level.

This work proposes an uncertainty-aware Monte Carlo Tree Search (MCTS) algorithm that uses such uncertainty information to provide a robust decision-making algorithm with inaccurate models. We use the uncertainty value of each prediction to inform the search strategy of MCTS and bias it towards transitions with small uncertainty. The result is that the planner is likely to choose accurate states and actions, leading to more accurate task executions.

We demonstrate the approach with a liquid pouring problem. To do so, we model pourings with Gaussian Process Regression (GPR) and use the variance of GPs as an accuracy index for the model predictions. Then, we use the uncertainty-aware MCTS to choose the sequence of pouring to reach a given filling level. We show that our approach performs pouring tasks even with very inaccurate models (e.g., a GP with a handful of points), leading to a much higher success rate than the baselines.

II. Related Works

Planning under uncertainty gathers all those planning methods that do not assume the model and the true system are identical. The goal of these methods is to find a sequence of actions that bring the system from an initial to a final state even though the model and the true dynamics differ.

If such a difference is owed to the inherent randomness of the system (i.e., aleatoric uncertainty), a common approach is to propagate the uncertainty on the search tree. The planner will search for a robust solution to reaching the goal despite the transitions' randomness [14]–[16]. If the model error is owed to the limited dataset or the lack of expressivity of the model, we are dealing with epistemic uncertainty. This case is less studied and is the focus of this work.

In the realm of sampling-based motion planning, existing approaches estimate a Model Deviation Estimate (MDE) predicting the accuracy of a transition and use it to bias the search. Mitrano et al. [3] train a classifier on the MDE to deem state-and-action pairs accurate or inaccurate during the search. Inaccurate transitions are not added to the tree in a sampling-based path planner. Faroni et al. [17] use the MDE as a cost function within an asymptotically optimal planner so that the resulting path should minimize the cumulative model mismatch. Other approaches limit the search space to a trusted region after an offline analysis of the model performance [18]. Considering graph-based search, Vemula et al. [19], [20] use A* to plan robot movements and, after the execution of each action, blacklist transitions whose deviation from the model is larger than a given threshold. All these approaches rely on a computationally expensive training phase to learn which actions can be trusted and use such information in a general motion planning algorithm.

MCTS is becoming popular in robotics planning. Recent applications include path planning with Partially Observable Markov Decision Processes (POMDP) [21], [22], multi-robot task allocation [23] and path planning [24], and rearrangement planning [25]. Existing works typically leverage MCTS to account for multi-agent decision-making, by modeling the multi-agency as a POMDP and using MCTS to solve the resulting problem. Differently from previous works, ours uses MCTS to explicitly reason through model uncertainty in the context of robotics planning. Compared to existing sampling-based and graph-based planners, MCTS can be easily extended to account for uncertainty [26].

Specific to robotized liquid pouring, most existing works focus on accurate modeling of the liquids' dynamics via deep learning or fluid simulation. For example, Pan et al. [27] combined a fine-grained fluid simulation with an optimization-based planner to achieve the pouring task. The motion planning problem was formulated as a continuous numerical optimization problem in the high-dimensional robot trajectory space, following an objective function that accounts for object avoidance, smoothness, and a target goal that ensures fluid particles enter the target container. Zhang et al. [28] used imitation learning to learn pouring tasks from human demonstrations. They devised a one-shot domain adaptation technique to reduce the sim-to-real gap between simulations and real-world environments. This led to a higher success rate in execution compared to non-adaptive methods. Babaians et al. [29] explored a deep reinforcement learning approach: the policy is learned in a simulated environment modeling the interaction between liquids and robots. The main drawback of all these methods is that they rely on the availability of accurate models, whose computation requires a large data collection or long computational times. We do not rely on precise modeling of the liquid dynamics. On the contrary, we aim to leverage the uncertainty of the model owing to limited data availability and show that the robot is still able to perform tasks accurately.

Other than planning-based methods, the most common approach to robotized pouring uses feedback control to continuously monitor the liquid level in the output container and regulate the pouring rate accordingly [30]–[32]. This approach can achieve very accurate results but assumes the liquid's level is measured at a high sampling rate and without interruptions. On the contrary, we measure the liquid's level only between discrete pouring actions, thus performing each action in open loop. Therefore, our assumption is less restrictive, especially when camera occlusions may happen or the robot has an in-hand camera (not allowing for simultaneous action and perception).

III. Problem Statement

Consider a dynamical system, $x_{k+1} = f(x_k, u_k)$ where $f : X \times U \to X$ and X and U are the state space and the action space, respectively. Ideally, we aim to find a sequence of actions, $u = (u_0, \dots, u_N)$, $N \in \mathbb{N}$, that brings the system from its initial state, x_{start}, to a desired goal set, X_{goal}. Possibly, we aim to do so in the least amount of actions. We can write this problem as follows:

$$\begin{aligned} \underset{u \in U}{\text{minimize}} \quad & N \\ \text{subject to} \quad & x_N \in X_{\text{goal}} \\ & x_0 = x_{\text{start}} \\ & x_{k+1} = f(x_k, u_k) \end{aligned} \tag{1}$$

In real-world problems, f is approximated by a dynamical model. We consider the deterministic model $\hat{f} : X \times U \to X$, approximating f, and define a Model Deviation Estimate, $\texttt{MDE}(x_k, u_k)$, that outputs an estimate of the model mismatch when action u_k is applied from state x_k. The model deviation estimate can be derived from real data [3], analytically [17], or as an estimate of the model epistemic uncertainty.

Under the assumption that x can be measured without uncertainty (i.e., x_{start} is known), we can solve the following approximation of (1):

$$\begin{aligned} \underset{u \in U}{\text{minimize}} \quad & N \\ \text{subject to} \quad & \hat{x}_N \in X_{\text{goal}} \\ & \hat{x}_0 = x_{\text{start}} \\ & \hat{x}_{k+1} = \hat{f}(\hat{x}_k, u_k) \end{aligned} \tag{2}$$

Problem (2) is similar to the standard formulation of a motion planning problem [33]. Motion planners usually find the whole solution to (2) beforehand and apply the entire sequence, u, in open loop. Because we follow an MCTS approach, we apply one action at a time and repeat the search after updating x_{start} with the observed state.

Problem (2) does not use MDE, meaning that its solution might contain state-and-action pairs for which $\hat{f}$ is very inaccurate.

Our goal is to overcome this issue by finding a solution whose state-and-action pairs have small MDE values. We do so by combining the GPs' variance with an MCTS algorithm to bias the search toward states and actions corresponding to low epistemic uncertainty.

IV. Method

We use GPs' variance to inform the search in a modified MCTS algorithm by devising an uncertainty-aware MCTS variant. This section describes how we address (2) within the MCTS framework, combine it with the MDE and give insights on the usage of GPR to compute the MDE.

A. Uncertainty-aware Monte Carlo Tree Search

MCTS is a search algorithm that uses experience-driven heuristics to bias the tree growth. The algorithm relies on the definition of three functions:

- a function returning all legal actions that can be taken at state x_k.
- a terminal-state checker returning whether a state is terminal (e.g., a goal or a state from which it is impossible to recover).
- a reward function, $r(x_k, u_k)$, which associates a state-and-action pair with a scalar reward.

The definition of these functions is problem-dependent. We will give examples for liquid pouring in Sec. V.

MCTS consists of four main phases, as shown in Alg. 1. Starting from the root node, it descends the tree by applying a **selection** rule until it reaches a leaf node. The leaf node undergoes an **expansion** procedure: all the actions available from that node are applied and the resulting nodes are appended to the tree. The algorithm picks one of the new nodes and applies random legal actions until it reaches a terminal state (**simulation**). At this point, the algorithm computes the reward and uses **backpropagation** to update the reward belief of all nodes on the tree walk from the root to the terminal state. The procedure repeats for a number of iterations and, finally, returns the most visited node.

We embed the MDE function in all these steps in a variant of the MCTS proposed in [26]. [26] modifies the selection, expansion, backpropagation, and simulation phases to account for action uncertainty. The ablation studies in [26] demonstrated that the uncertainty-aware variations of the selection and expansion phases are those that impact the search the most. On the contrary, backpropagation and simulation do not significantly affect the results. For this reason, we propose the following uncertainty-aware selection and expansion functions:

Algorithm 1: MCTS algorithm. Adapted from [26].

```
Function MCTS(x_0):
    create a root node v_0 with state v_0.x ← x_0
    for N_I iterations do
        v_s ← SELECT(v_0)
        if N(v_s) > 0 then
            v_s ← EXPAND(v_s)
        reward ← SIMULATE(v_s.x)
        BACKPROPAGATE(v_s, reward)
    v_best ← choose the most visited child of v_0
    return action that generated v_best
```

1) Seletion: Selection is applied to already visited nodes. The goal is to choose promising nodes from which to expand the tree. MCTS uses UCT, an extension of Upper Confidence Bound (UCB) to trees. Given a node in the tree, v, and a set of children, $\mathcal{V}$, UCT chooses the next node, v_{next}, as:

$$v_{\text{next}} = \underset{v_i \in \mathcal{V}}{\operatorname{argmax}} \frac{Q(v_i)}{N(v_i)} + c\sqrt{\frac{\ln N(v)}{N(v_i)}} \tag{3}$$

where $Q(v)$ returns the sum of rewards observed by v and $N(v)$ returns how many times v has been visited so far; $c > 0$ is a user-defined constant. The first term in (3) is an estimate of the average reward of v; the second one is inversely proportional to how many times v was chosen in the past. As a consequence, UCT trades off exploration of less visited nodes and exploitation of nodes that returned a high reward in the past. We modify the UCT rule to make it uncertainty-aware as follows:

$$v_{\text{next}} = \underset{v_i \in \mathcal{V}}{\operatorname{argmax}} \left(\frac{Q(v_i)}{N(v_i)} + c\sqrt{\frac{\ln N(v)}{N(v_i)}} \right) \cdot (1 - \delta_i) \tag{4}$$

where

$$\delta_i = \frac{e^{\texttt{MDE}(v_i)/\tau}}{\sum_{v_j \in \mathcal{V}} e^{\texttt{MDE}(v_j)/\tau}}$$

is the softmax function with temperature parameter τ and $\texttt{MDE}(v)$ refers to the MDE function evaluated in the state and action that generated v. In this way, the selection phase is biased towards nodes with a low MDE value. The modified selection function is in Alg. 2.

2) Expansion: Expansion means generating the children of a leaf node, v, by propagating a legal action a_i from the node state, $v.x$. The expansion function can be modified to favor certain children or to limit the branching factor (for computational efficiency). In our algorithm, we randomly discard children with a probability proportional to their MDE value. More in detail, we denote by θ the average MDE of all children of the node and discard a child v_i with a probability given by the following sigmoid function:

$$p(v_i) = \frac{1}{1 + e^{h(\texttt{MDE}(v_i) - \theta)}} \tag{5}$$

where h is the steepness of the sigmoid. The result is that children with MDE values greater than the average will have

a low probability of being expanded. The modified selection function is in Alg. 3.

Remark (Continuous action-spaces): MCTS is naturally suited for discrete action spaces. For this reason, our approach was formulated for such a case. The approach extends to sampling-based continuous-space MCTS variants (e.g., *progressive widening* [34]), which sample new actions from the continuous space and apply selection and expansion to a growing set of actions. In this case, we can simply substitute the selection and expansion functions with the proposed uncertainty-aware versions.

B. Modeling uncertainty

Previous works computed an MDE function analytically [17][35] or from a dataset of robot trajectory executions [3][13]. Our uncertainty-aware MCTS is independent of the approach used to retrieve the MDE. Nonetheless, it is worth noting that some of the most widespread modeling techniques in robotics (e.g., GPs and neural-network ensembles) come with an estimate of the epistemic uncertainty of the model's predictions. The validation of our approach uses GPR to model the system dynamics. For this reason, we give insights on the usage of GPR to this purpose.

GPR is a non-parametric regression approach. It uses GPs to infer a Gaussian distribution for a new point given a set of observations and a kernel function.

Let $y = \{y_1, \dots, y_H\}$ be the set of observed values corresponding to features vectors $z = \{z_1, \dots, z_H\}$ and $z' = \{z'_1, \dots, z'_H\}$ the set of new feature vectors for inference. In a GP, the joint distribution of observed and predicted values is a Gaussian distribution:

$$\begin{pmatrix} y \\ y' \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} k(z,z) & k(z,z') \\ k(z',z) & k(z',z') \end{pmatrix}\right) \tag{6}$$

from which it is possible to infer y' as a Gaussian distribution with mean μ and variance σ^2.

We can use GPR to model our system's dynamics, f. By denoting the feature vector by $z_k = (x_k^T, u_k^T)^T$, we aim to predict the state of the next step, $\hat{x}_{k+1}$. Because $\hat{x}_{k+1}$ is computed through GPR, it comes with a variance σ^2 that depends on the location of observed points, z. The lower the density of observations in the neighborhood of the inferred point, the larger the prediction variance. Hence, the variance can be seen as a proxy of the epistemic uncertainty of the model. We use such variance as our MDE value in Alg. 2–3.

V. Application to Robotized Pouring

We consider a robotized liquid pouring case study and show how our uncertainty-aware approach can be applied to increase the success rate during execution. This section describes the pouring problem, the experimental setup, and the modeling approach.

A. Problem definition

The pouring problem consists of filling an output container with a sequence of pouring actions. We assume we can measure the level only after executing each action, as typical of in-hand cameras.

Algorithm 2: Uncertainty-aware selection function

Function SELECT$(v,\ \tau)$**:**
  while v.isFullyExpanded *is true* **do**
    $\mathcal{V} \leftarrow$ getChildren(v)
    for $v_i \in \mathcal{V}$ **do**
      $\delta_i \leftarrow \frac{e^{\text{MDE}(v_i)/\tau}}{\sum_{v_j \in \mathcal{V}} e^{\text{MDE}(v_j)/\tau}}$
    $v \leftarrow \text{argmax}_{v_i \in \mathcal{V}} \left(\frac{Q(v_i)}{N(v_i)} + c\sqrt{\frac{\ln N(v)}{N(v_i)}}\right)(1-\delta_i)$
  return v

Algorithm 3: Uncertainty-aware expansion function

Function EXPAND$(v,\ h)$**:**
  $\mathcal{A} \leftarrow$ getPossibleActions(v)
  $\mathcal{V} \leftarrow \emptyset$
  for $a_i \in \mathcal{A}$ **do**
    $x \leftarrow f(v.x, a_i)$
    create a node v_i with state $v_i.x \leftarrow x$
    $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_i\}$
  $\theta = \frac{1}{|\mathcal{V}|}\sum_{v_i \in \mathcal{V}} \text{MDE}(v_i)$
  for $v_i \in \mathcal{V}$ **do**
    $p' \leftarrow$ uniform$(0,1)$
    $p'' \leftarrow \frac{1}{1+e^{h(\text{MDE}(v_i)-\theta)}}$
    if $p' > p''$ **then**
      add node v_i to the tree
  v.isFullyExpanded $\leftarrow$ True
  return a random child of v

The system state, x, is the liquid's level in the output container, measured as a percentage of the container's maximum filling. Each action consists of a robot's tool rotation, α, for a time duration, d, so that $u_k = (\alpha_k, d_k)^T$. The goal is to reach a reference filling percentage, x_{ref}, with a tolerance of $\pm c$, i.e., $X_{\text{goal}} = [x_{\text{ref}} - c, x_{\text{ref}} + c]$. We use GPR to predict the level, given the current level and a new action, so that $\hat{x}_{k+1} = \hat{f}(x_k, u_k)$.

We use a sparse reward function r (i.e., only terminal states receive a reward) that favors solutions reaching the goal with a small number of actions. Hence, we define

$$r(x_k, u_k) = \begin{cases} \rho(x_k, u_k), & \text{if isTerminal}(x_k, u_k) = \text{True} \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

$$\rho(x_k, u_k) = \begin{cases} 1 + \frac{1}{k+1}, & \text{if } \hat{f}(x_k, u_k) \le x_{\text{ref}} + c \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

$$\text{isTerminal}(x_k, u_k) = \begin{cases} \text{True}, & \text{if } \hat{f}(x_k, u_k) \ge x_{\text{ref}} - c \\ & \quad \text{or } k > N_{\max} \\ \text{False}, & \text{otherwise} \end{cases} \tag{9}$$

where $N_{\max}$ is the maximum search depth. Finally, we discretize the action space with a step of 0.25 rad and 0.1

TABLE I: Mean Squared Error of GPR models.

Dataset size	40 points	20 points	10 points	5 points
MSE	18.6	24.5	25.3	33.4

seconds for α and d, respectively.

B. Experimental setup

The setup is in Fig. 1. We use a 6-dof manipulator, Universal Robots UR5e, with a 3D-printed custom tool to hold a common wine bottle. The robot is controlled in ROS Noetic, through the *MoveIt!* library [36].

The perception system consists of a Luxonis OAK-D Pro RGB-D camera. The perception pipeline uses the RGB image from the camera and applies the following steps: (i) It crops the RGB image in a region of interest containing the container; (ii) It applies Gaussian blurring; (iii) It converts the image to HSV coordinates and thresholds it to isolate the hue of the liquid. (iv) It extracts the larger resulting blob and computes its bounding box; (v) Based on the aspect ratio of the bounding box, ar, and knowing the aspect ratio of the bounding box of the full container, $\mathrm{ar}_{\mathrm{full}}$, it computes the filling percentage as $l = 100\,\mathrm{ar}/\mathrm{ar}_{\mathrm{full}}$ [%].

The pipeline is illustrated in Fig. 2. Note that, to facilitate perception, we use dyed water and a transparent output container. With this perception pipeline, we were able to achieve a measurement accuracy of around 1%.

C. Pouring modeling

We used GPR to model the pouring dynamics. We modeled the pouring process with different dataset sizes (40, 20, 10, and 5 points). Each dataset point is a pouring and consists of the current level, x_k, and the action to be performed, $u_k = (\alpha_k, d_k)^T$. The target value is the next level, x_{k+1}. Points of the 40-point dataset were chosen randomly. The smaller datasets were created by randomly sampling points from the 40-point dataset. After a kernel selection phase, we opted for the combination of a Dot Product and a Rational Quadratic kernel. Table I shows the Mean Squared Error (MSE) of the GPR models computed on a test set of 20 points. As expected, the smaller the dataset, the greater the MSE value, meaning predictions are likely to be less accurate when the training dataset is smaller.

VI. Experiments

This section compares different planning algorithms showing that uncertainty-aware methods can carry out pouring tasks with a higher success rate.

A. Tests and metrics

We compare the following algorithms:

- MCTS: a standard MCTS algorithm without uncertainty awareness.
- UA-MCTS-0: the uncertainty-aware MCTS algorithm proposed in [26].
- MCTS-inflated: standard MCTS with a conservative model propagation, i.e., we compute $\hat{x}_{k+1} = \hat{f}(x_k, u_k) + w\,\mathtt{MDE}(x_k, u_k)$, with $w = 1.2$, during the search. This is a hand-crafted robust baseline that overestimates the effect of each pouring action proportionally to the uncertainty of transitions.
- UA-MCTS-1: the proposed approach with selection and expansion functions described in Alg. 2-3.

All methods use the same GPR models, reward functions, and terminal-state checkers and have the same allotted computational time (0.5 s).

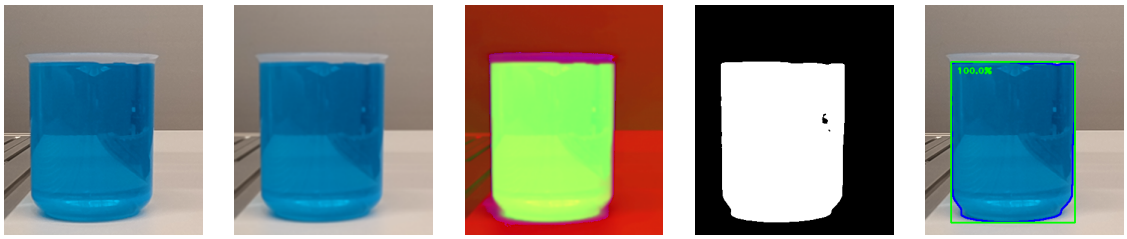
We performed 30 tests for each model and method and evaluated the success rate (% of tests where the final level belongs to X_{goal}) and the number of actions to reach a terminal state. We define $X_{\mathrm{goal}} = [x_{\mathrm{ref}} - c, x_{\mathrm{ref}} + c$, where $c = 2.5\%$ and x_{ref} is chosen randomly (the same value of x_{ref} is used for all four methods).

The experiments require choosing the Sigmoid steepness, h, and the Softmax temperature, τ, in Alg. 2-3, and the inflation factor w in MCTS-inflated. h and τ determine how biased UA-MCTS is toward low-uncertainty actions. In particular, large values of h will be more likely to reject actions whose uncertainty is higher than the average uncertainty θ in Alg. 3. Similarly, small values of τ will assign a higher selection probability to low-uncertainty actions in Alg. 2. The inflation factor w in MCTS-inflated determines how conservative the model propagation is. Large values of w will greatly overestimate the effect of pouring actions, leading to many small pourings. Vice versa, if w is close to one, MCTS-inflated will behave similarly to standard MCTS. In the experiments, we used $h = 10$, $\tau = 0.1$, and $w = 1.2$ for all tests. We leave the quantitative analysis of the effects of parameter tuning on the results of the method as future works.

B. Results

Table II shows the success rate and the number of actions to reach the goal for all methods and underlying models. MCTS-inflated and UA-MCTS-1 have comparable success rates across all models and are significantly better than MCTS and UA-MCTS-0. This difference becomes wider as the size of the training dataset decreases (up to 40 % for the 10-point model). This fact is expected for MCTS because the model becomes less accurate and MCTS does not take the GP's variance into account during the search. Unexpectedly, we did not not observe a significant difference between MCTS and UA-MCTS's policies. Possibly, the selection and expansion strategy proposed in [26] are not biased enough to discard inaccurate actions for the problem at hand.

Even though MCTS-inflated and UA-MCTS have similar success rates, the number of actions they take to reach the goal differs significantly. With more accurate models (e.g., 40-point GP), MCTS-inflated has a number of actions comparable with standard MCTS and smaller than UA-MCTS-1 (see rows 1 and 3 of Table IIa). As the model becomes less accurate (e.g., 5-point GP), MCTS-inflated's number of actions is the highest among all methods (see rows 1 and 3 of Table IId). This behavior is explained



(a) Crop raw RGB (b) Add Gaussian blur (c) Convert to HSV (d) Apply thresholding and find blob (e) Compute level from aspect ratio

Fig. 2: Image processing pipeline from RGB image to liquid volume identification.

TABLE II: Experimental results.

(a) 40-point dataset

	Success rate [%]	N. of actions mean (std.dev.)
MCTS	80	1.45(0.59)
UA-MCTS-0	77	1.45(0.59)
MCTS-inflated	**100**	1.70(0.95)
UA-MCTS-1	**100**	2.05(1.02)

(b) 20-point dataset

	Success rate [%]	N. of actions mean (std.dev.)
MCTS	66	1.80(0.87)
UA-MCTS-0	66	1.80(0.87)
MCTS-inflated	93	2.55(1.20)
UA-MCTS-1	**97**	2.50(1.56)

(c) 10-point dataset

	Success rate [%]	N. of actions mean (std.dev.)
MCTS	60	1.65(1.01)
UA-MCTS-0	60	1.65(1.01)
MCTS-inflated	97	2.60(1.07)
UA-MCTS-1	**100**	2.75(1.48)

(d) 5-point dataset

	Success rate [%]	N. of actions mean (std.dev.)
MCTS	64	1.20(0.40)
UA-MCTS-0	64	1.20(0.40)
MCTS-inflated	**97**	2.85(0.85)
UA-MCTS-1	**97**	2.40(1.16)

by considering the propagation function used in MCTS-inflated: by inflating the predicted level, the planner tends to predict higher levels when the variance of the action is large. This does not prevent it from taking inaccurate actions, yet their effect is greatly overestimated. As a consequence, the actual level will be lower than the predicted one and the number of actions required to reach the goal will increase, especially when the average GP's variance is large. UA-MCTS-1 approach is different: it chooses actions with low variance, no matter the average accuracy of the model. As a consequence, the algorithm may take more actions to reach the goal (see last rows of Tables IIa-IId).

An example of action sequences generated by MCTS and UA-MCTS-1 is in Fig. 3, showing how MCTS tends to reach the goal with fewer actions, yet exceeding the goal level, leading to a failure. On the contrary, UA-MCTS-1 is more conservative and takes one action more, yet it correctly reaches the goal. Please refer to the accompanying video for more details.

Fig. 4 clarifies the different policies of the methods. The actions of MCTS-inflated and UA-MCTS-1 are superimposed on the GP's standard deviation across the action space. MCTS-inflated's actions are scattered around the action space independently from the action's variance (similar distributions were obtained for MCTS and UA-MCTS-0 but are not shown here for the sake of brevity). Differently, UA-MCTS-1's actions are always in the low-variance region of the action space (blue area in Fig. 4), demonstrating that the approach tends to discard high-variance actions.

Although both MCTS-inflated and UA-MCTS-1 have comparable success rates, it is worth noting that MCTS-inflated is less general than UA-MCTS-1. Indeed, MCTS-inflated assumes that inflating the effect of an action will hinder a conservative effect during execution. This may be true for one-dimensional problems like the pouring task at hand but does not hold in general (especially for higher state dimensionality). Conversely, UA-MCTS-1 generalizes to higher dimensions, as its strategy directly reasons through transitions' inaccuracy.

Finally, note that all methods were tested with the same allotted computational time (0.5 s). The additional computation owed to the proposed method is negligible, as it only requires the computation of Sigmoid and Softmax function proportionally to the branching factor of the problem. Nonetheless, this additional computation is typically negligible relative to the model propagation.

VII. Conclusions

We proposed an uncertainty-aware Monte Carlo Tree Search approach to reason through the prediction error of learned models in robotics tasks. We use uncertainty estimates such as the variance of Gaussian Processes to bias the search toward low-uncertainty states and actions. We demonstrated the approach in a robotized pouring task with an RGB-based perception system and a robot arm. Our approach proved robust even with GPs obtained from very small datasets (up to 5 points), while baseline approaches showed a decay in the execution success rate as the model became less accurate.

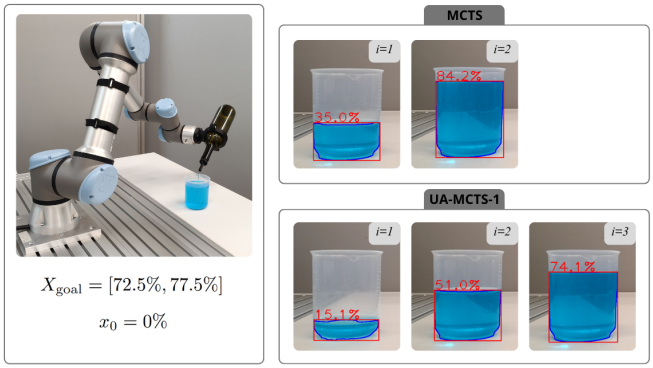


Fig. 3: Example of pouring sequences with uncertainty-aware and -unaware methods.

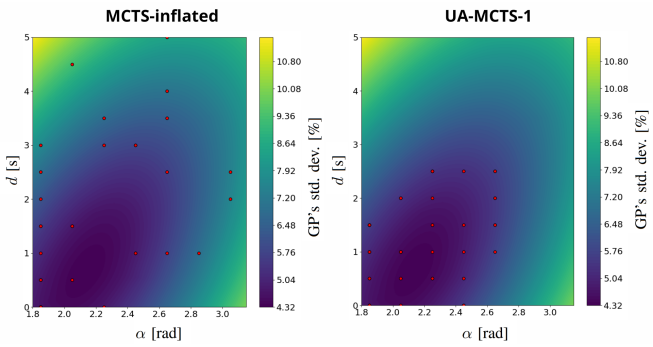


Fig. 4: Actions (α, d) taken by the algorithm superimposed on the standard deviation of the 5-point GP model.

Future works will extend the validation to more complex scenarios, with different container's shapes and considering the robot motion planning and collision avoidance. A quantitative analysis of the hyperparameter tuning effects will also be carried out.

References

[1] M. Lippi, P. Poklukar, M. C. Welle, A. Varava, H. Yin, A. Marino, and D. Kragic, "Enabling visual action planning for object manipulation through latent space roadmap," *IEEE T-RO*, vol. 39, pp. 57–75, 2023.

[2] G. Nicola, E. Villagrossi, and N. Pedrocchi, "Co-manipulation of soft-materials estimating deformation from depth images," *Robotics and Computer-Integrated Manufacturing*, vol. 85, p. 102630, 2024.

[3] P. Mitrano, D. McConachie, and D. Berenson, "Learning where to trust unreliable models in an unstructured world for deformable object manipulation," *Science Robotics*, vol. 6, no. 54, 2021.

[4] X. Lin, C. Qi, Y. Zhang, Z. Huang, K. Fragkiadaki, Y. Li, C. Gan, and D. Held, "Planning with spatial-temporal abstraction from point clouds for deformable object manipulation," in *CoRL*, 2022.

[5] P. Mitrano, A. LaGrassa, O. Kroemer, and D. Berenson, "Focused adaptation of dynamics models for deformable object manipulation," *RSS*, 2022.

[6] J. Liang, X. Cheng, and O. Kroemer, "Learning preconditions of hybrid force-velocity controllers for contact-rich manipulation," in *CoRL*, 2022.

[7] E. Páll, A. Sieverling, and O. Brock, "Contingent contact-based motion planning," in *IEEE/RSJ IROS*, 2018, pp. 6615–6621.

[8] M.-T. Khoury, A. Orthey, and M. Toussaint, "Efficient sampling of transition constraints for motion planning under sliding contacts," in *IEEE CASE*, 2021, pp. 1547–1553.

[9] A. V. Vivas, A. Cherubini, M. Garabini, P. Salaris, and A. Bicchi, "Minimizing energy consumption of elastic robots in repetitive tasks," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2023.

[10] T. Marcucci, M. Garabini, G. M. Gasparri, A. Artoni, M. Gabiccini, and A. Bicchi, "Parametric trajectory libraries for online motion planning with application to soft robots," in *ISRR*, 2020, pp. 1001–17.

[11] J. Liu, Y. Chen, Z. Dong, S. Wang, S. Calinon, M. Li, and F. Chen, "Robot cooking with stir-fry: Bimanual non-prehensile manipulation of semi-fluid objects," *IEEE RA-L*, vol. 7, no. 2, pp. 5159–5166, 2022.

[12] Z. Sun, Z. Wang, J. Liu, M. Li, and F. Chen, "Mixline: A hybrid reinforcement learning framework for long-horizon bimanual coffee stirring task," in *ICIRA*, 2022, pp. 627–636.

[13] A. LaGrassa, M. Lee, and O. Kroemer, "Task-oriented active learning of model preconditions for inaccurate dynamics models," in *IEEE ICRA*, 2024, pp. 16 445–16 445.

[14] A. Bry and N. Roy, "Rapidly-exploring random belief trees for motion planning under uncertainty," in *IEEE ICRA*, 2011, pp. 723–730.

[15] Q. H. Ho, Z. N. Sunberg, and M. Lahijanian, "Gaussian belief trees for chance constrained asymptotically optimal motion planning," in *IEEE ICRA*, 2022, pp. 11 029–11 035.

[16] A. Wu, T. Lew, K. Solovey, E. Schmerling, and M. Pavone, "Robust-rrt: Probabilistically-complete motion planning for uncertain nonlinear systems," in *ISRR*, 2022, pp. 538–554.

[17] M. Faroni and D. Berenson, "Motion planning as online learning: A multi-armed bandit approach to kinodynamic sampling-based planning," *IEEE RA-L*, vol. 8, no. 10, pp. 6651–6658, 2023.

[18] C. Knuth, G. Chou, N. Ozay, and D. Berenson, "Planning with learned dynamics: Probabilistic guarantees on safety and reachability via lipschitz constants," *IEEE RA-L*, vol. 6, pp. 5129–5136, 2021.

[19] A. Vemula, Y. Oza, J. A. Bagnell, and M. Likhachev, "Planning and execution using inaccurate models with provable guarantees," in *RSS*, 2016.

[20] A. Vemula, J. A. Bagnell, and M. Likhachev, "Cmax++: Leveraging experience in planning and execution using inaccurate models," in *AAAI Conference on Artificial Intelligence*, 2021, pp. 6147–6155.

[21] T. Dam, G. Chalvatzaki, J. Peters, and J. Pajarinen, "Monte-carlo robot path planning," *IEEE RA-L*, vol. 7, no. 4, pp. 11 213–11 220, 2022.

[22] W. Li, Y. Liu, Y. Ma, K. Xu, J. Qiu, and Z. Gan, "A self-learning monte carlo tree search algorithm for robot path planning," *Frontiers in Neurorobotics*, vol. 17, p. 1039644, 2023.

[23] B. Kartal, E. Nunes, J. Godoy, and M. Gini, "Monte carlo tree search for multi-robot task allocation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[24] M. Dalmasso, A. Garrell, J. E. Domínguez, P. Jiménez, and A. Sanfeliu, "Human-robot collaborative multi-agent path planning using monte carlo tree search and social reward sources," in *IEEE ICRA*, 2021, pp. 10 133–10 138.

[25] H. Song, J. A. Haustein, W. Yuan, K. Hang, M. Y. Wang, D. Kragic, and J. A. Stork, "Multi-object rearrangement with monte carlo tree search: A case study on planar nonprehensile sorting," in *IEEE/RSJ IROS*, 2020, pp. 9433–9440.

[26] F. Kohankhaki, K. Aghakasiri, H. Zhang, T.-H. Wei, C. Gao, and M. Müller, "Monte carlo tree search in the presence of transition uncertainty," *AAAI Conference on Artificial Intelligence*, vol. 38, no. 18, pp. 20 151–20 158, 2024.

[27] Z. Pan, C. Park, and D. Manocha, "Robot motion planning for pouring liquids," in *ICAPS*, vol. 26, 2016, pp. 518–526.

[28] D. Zhang, W. Fan, J. Lloyd, C. Yang, and N. F. Lepora, "One-shot domain-adaptive imitation learning via progressive learning applied to robotic pouring," *IEEE T-ASE*, vol. 21, no. 1, pp. 541–554, 2024.

[29] E. Babaians, T. Sharma, M. Karimi, S. Sharifzadeh, and E. Steinbach, "Pournet: Robust robotic pouring through curriculum and curiosity-based reinforcement learning," in *IEEE/RSJ IROS*, 2022, pp. 9332–39.

[30] C. Schenck and D. Fox, "Visual closed-loop control for pouring liquids," in *IEEE ICRA*, 2017, pp. 2629–2636.

[31] C. Dong, M. Takizawa, S. Kudoh, and T. Suehiro, "Precision pouring into unknown containers by service robots," in *IEEE/RSJ IROS*, 2019, pp. 5875–5882.

[32] C. Do and W. Burgard, "Accurate pouring with an autonomous robot using an rgb-d camera," in *IAS*, 2019, pp. 210–221.

[33] S. M. LaValle, *Planning algorithms*. Cambridge Univ. Press, 2006.

[34] A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard, "Continuous upper confidence trees," in *International Conference on Learning and Intelligent Optimization*, 2011, pp. 433–445.

[35] M. Faroni and D. Berenson, "Online adaptation of sampling-based motion planning with inaccurate models," in *IEEE ICRA*, 2024, pp. 2382–2388.

[36] D. Coleman, I. A. Sucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: a moveit! case study," *Journal of Software Engineering for Robotics*, vol. 5, p. 3–16, 2014.