2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Operating with Inaccurate Models by Integrating Control-Level Discrepancy Information into Planning

Ellis Ratner[1], Claire J. Tomlin[1], Maxim Likhachev[2]

***Abstract*— Typical robotic systems rely on models for planning. Therefore, the quality of the robot's behavior is heavily dependent on how accurately the model can predict the outcome of the robot's actions in the environment. A challenge, however, is that no model is perfect; moreover, we often do not know where discrepancies between the model's prediction and the actual outcome occur prior to observing executions in the real-world. One way to address this is to bias the planner away from these discrepancies by inflating the cost of states and actions where we previously observed the model to be inaccurate. Making such decisions about where and how to bias purely at the planning-level, however, neglects valuable information from the control-level, which gives a more fine-grained understanding of where and how the model went wrong during execution. Based on this observation, our key idea is to first infer a statistical model over discrepancies in the control-level's model. Then, we translate this model to the planning-level, where we use it to more informatively bias the planner away from states and actions where the model's predicted outcome is likely to be inaccurate. We demonstrate that our framework enables a robot to complete tasks, despite an inaccurate planning model, with greater efficiency than existing approaches. We do so through an experimental evaluation in simulation and real-robot experiments on NASA's Astrobee free-flyer.**

I. Introduction

Robots rely on models to plan intelligent behaviors and complete tasks. Therefore, robots require models that accurately predict the outcome of actions in the real-world. Unfortunately, however, regardless of how we construct a model — from first-principles, through data-driven methods, etc. — it will never be accurate everywhere. An important challenge, therefore, is how to enable robots to complete tasks despite using inaccurate models.

For example, consider a robot navigating to a goal, as shown in Fig. 1. Here, the robot neglects to model the fan, which then pushes it off of the planned path into the obstacle. The robot's propulsion is insufficient to overcome the fan, and therefore it must find an alternate path to the goal.

Finding a better path to the goal, however, requires changing how the planner reasons in some way to account for the fan. One possibility is to learn a new model, using observations from the robot's actual execution, through model learning [1], model-based reinforcement learning [2], or learning-based control [3]. With limited data and difficult-to-model phenomena such as aeorodynamics, however, this

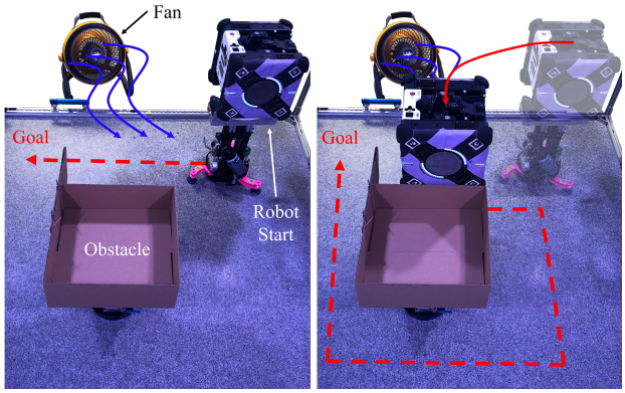

Fig. 1: (Left) The robot plans a path to the goal through a fan's strong air current not captured by its model. (Right) To avoid the fan, the robot must plan an alternate path.

can be very challenging. Moreover, it is often unnecessary for completing the task– in Fig. 1's example, it is possible for the robot to take a longer path around the other side of the obstacle to avoid the fan altogether and still reach the goal. Finding this path does not require a detailed model of the fan's effect on the robot.

Motivated by this, our work focuses on biasing the planner away from regions where the original model has proven to be inaccurate, rather than learning a completely new model. A challenge, however, is *how* to informatively bias the planner, using only the data observed during execution.

To address this, note that any observed performance of a path following controller provides us with valuable information about where the model is likely inaccurate. From the robot's actual path in Fig. 1 (Right, solid line) compared to its planned path (Left, dotted line), we can infer that the model is not correctly predicting action outcomes in this region; furthermore, the inputs that the controller applies while failing to follow the plan provide even more fine-grained information about *how* the predictions are wrong.

Our key idea is to infer a statistical model over discrepancies between the controller's model and the real-world, using this information. We then translate this discrepancy model to the planning-level, by inflating the cost of states and actions with a *control-level penalty*. To compute this penalty for a state and action, we forward-simulate the behavior of the controller when following this action, taking into account the control-level discrepancy model, and compute a probability over whether the final state differs by more than a small constant from the outcome predicted by the planner's model. The control-level penalty is then proportional to this

[1]Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, Berkeley, CA USA. {eratner, tomlin}@berkeley.edu

[2]Robotics Institute, Carnegie Mellon University, Pittsburgh, PA USA maxim@cs.cmu.edu

This research is supported by a NASA Space Technology Research Grant Fellowship.

probability.

In this paper, we contribute:

1) a principled approach for integrating planning and control to enable robots to complete tasks despite using inaccurate models;
2) an experimental analysis of our approach in simulation, which demonstrates a significant improvement in efficiency over baseline approaches;
3) and, a demonstration of our approach on NASA's Astrobee free-flying robot [4] in the lab.

II. Related Work

We focus on how a robot can complete a task despite using a model that is inaccurate in ways unknown prior to execution. Our approach changes the planner's behavior but keeps the model unchanged, in contrast to adjacent research in, for example, model learning [1], model-based reinforcement learning [2], and learning-based control [3].

A. Planning with inaccurate models

To bias the planner away from inaccurate parts of the model, Vemula et al. [5] introduce a penalty, or additional cost, at states and actions where the model's predictions were observed to be inaccurate during execution. [6] extends this to incorporate model-free learning to improve performance on repetitive tasks. Our approach differs in two key aspects. First, we maintain a probability over whether the model will poorly predict an action's outcome, and add an additional cost proportional to this probability (rather than a fixed cost, as in [5]). Second, we use control-level information to better guide where to inflate the planner's cost function.

Pan et al. [7] address a similar problem in task planning. They propose to learn probabilities over actions resulting in unexpected outcomes, over repeated executions of the task.

For some applications, we wish to trade-off model accuracy for speed of planning. For example, modelling deformable objects is computationally challenging; [8, 9] propose to use a simplified model in planning by first training a classifier to predict where discrepancies will occur in the simplified model. They then use this classifier to bias the planner away from states and actions where it predicts a discrepancy. While idea of biasing the planner in this way is similar to our work (and to [5]), we do not assume access to a higher fidelity model of the environment, as is required for training the classifier in [8, 9].

[10] introduces a way to learn a predictor for whether an action taken by a high-level planner will be executable by a lower-level of planning or control. This requires a way to simulate executions of the higher-level actions by the lower-level, which we cannot do before actually executing the robot in the real-world.

B. Integrating planning and control

Many robots, from legged systems [11, 12] to autonomous vehicles [13, 14], use hierarchical planning and control systems, wherein a high-level planner generates a path for a low-level controller to follow. Typically, the planner uses a simpler model to plan faster over a longer horizon, whereas the control uses a more complex model, closer to the actual robot dynamics, over a shorter horizon. A challenge is to ensure that the plan is executable by the controller, despite differences between the models.

To address this, [15–17] perform an offline reachability analysis to characterize differences between the planning and control models. Using these precomputations of this during planning, these approaches can guarantee that the planner will avoid paths that the controller would be unable to follow. An alternate approach is to plan with funnels [18], or motion primitives that the system is guaranteed to follow up to bounded disturbances [19, 20], to ensure a feasible path.

A related approach is to increase the model complexity and replan only when the controller cannot follow the plan during execution [21, 22]. This is similar to our approach in that control-level information is used to change the operation of the planner. However, this requires having a set of models to switch to, which we do not assume in our problem setting.

Compared to these approaches, our work involves a greater amount of feedback from the controller to the planner: information about model discrepancies from control is fed back to bias the planner. Additionally, we focus on discrepancies between the robot's model and the real-world, rather than between the two models themselves. From that perspective, many of these approaches are complementary to our work.

III. Problem Setup

We consider a hierarchical system consisting of a planning-level and a control-level. Our approach integrates planning and control to handle discrepancies between our environment model and the real-world.

A. Planning-level

We use a deterministic graph-based representation of the planning problem, consisting of a state space S, action space A, a successor function $succ(s, a)$, which gives the outcome of taking action a at state s, and a cost function $c(s, a)$, which assigns a nonnegative, finite cost to taking a at s. Furthermore, we assume that all discrepancies between our model and the real-world do not change over time. This assumption precludes, for example, other agents unexpectedly moving through the environment, or static obstacles that the robot discovers to be movable during execution.

Given a start state $s_{\text{start}} \in S$, the planner finds a least-cost path to a goal in $G \subset S$; we denote this path π.

B. Control-level

For control, we represent the environment through a state space X (typically $\mathbb{R}^n$, with n being the state dimension), control space U (typically $\mathbb{R}^m$, with m being the control dimension), and a difference equation $x_{t+1} = f(x_t, u_t, d_t)$, which gives the outcome x_{t+1} of applying the control u_t to the system at state x_t, and t is the integer-valued time index. We assume an additional input $d_t \in D$ (typically $D = \mathbb{R}^l$), which represents the discrepancy between our model of the

environment f and the real-world. We do not know d_t *a priori*.

At each time step, the controller produces a u_t so as to reduce the deviation between the robot's actual state x_t and its desired state x_t^* to the greatest extent possible. We compute the desired state trajectory $\{x_t^*\}$ by postprocessing π. We do not impose any additional constraints on the controller's structure, aside from being able to forward-simulate the controller's output for arbitrary state and desired state trajectory inputs. Specifically, given a state x_t and desired trajectory $\{x_t\}$, we must be able to simulate the output of the controller, u_t.

Finally, we require that there exists a function $\phi : S \to X$, representing the relationship between the planning- and control-level states spaces.

C. Control-level discrepancies

We characterize the discrepancy between the control model f and the real-world through an unknown function of state and control, $d(x, u)$, similar to the disturbance in [23]. We refer to d as the *control-level discrepancy model*.

Specifically, $d(x,u) = \begin{bmatrix} d_1(x,u) & \dots & d_l(x,u) \end{bmatrix}^\top$, where we assume that the components d_i and d_j are independent, for $i \neq j$. We then infer each d_i using Gaussian Process (GP) regression (see, for example [24]). For the remainder of this section, we use the shorthand $z = \begin{bmatrix} x & u \end{bmatrix}^\top$, to represent the stacked vector of state x and control u.

We assume each component i to be a GP with zero mean function and a squared exponential covariance function of the form

$$C(z, z') = v_1^2 \exp\left(-\frac{1}{2}\sum_{i=1}^{n+m}\frac{(z_i - z_i')^2}{w_i^2}\right) + v_0^2\delta_{zz'} \quad (1)$$

where $\{w_i\}, v_0$, and v_1 are hyperparameters, which are typically computed by maximizing the marginal likelihood of the training data (and hence must be recomputed each time more training data is added).

For a new x, u not part of the data set, we denote the predicted mean and variance of d_i as $\mu_i(x,u)$ and $\sigma_i^2(x,u)$, respectively (for $i = 1, 2, \dots, l$). We can express these functions analytically in terms of the covariance function in Eq. 1, and the data set (see [24]).

IV. Approach

Our approach addresses the challenge of inaccurate models by interleaving planning and execution, which we summarize in Alg. 1, and illustrate in Fig. 2. We first plan a least-cost path to goal π from the robot's current state s_{start} (Alg. 1, lines 2-4). We then execute the path at the control-level, and monitor the execution for deviations exceeding a threshold of δ (Alg. 1, lines 6-12). If this threshold is exceeded, we use the robot's execution history to update our control-level discrepancy model (Alg. 1, line 13). We provide more detail on each step in the following sections.

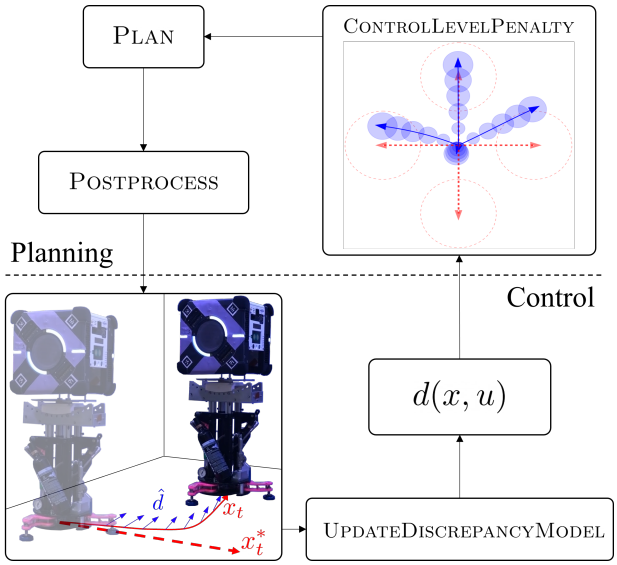

Fig. 2: An illustration of our approach. The planning-level first generates a desired path (dashed line), which the control-level then executes. If the robot's actual path (solid line) deviates, then we update the control-level discrepancy model with the estimated discrepancies $\hat{d}$. The planner uses the new d to bias away from actions likely to result in a discrepancy, using the control-level penalty.

Algorithm 1 Interleaving planning and execution.

1: **while** robot has not reached a goal **do**
2: $s_{\text{start}} \leftarrow$ get robot's (planning) state
3: $\pi \leftarrow$ Plan(s_{start})
4: $\{x_i^*\} \leftarrow$ Postprocess(π)
5: history $\leftarrow \emptyset$
6: **do**
7: $x_t \leftarrow$ get robot's (control) state
8: $u_t \leftarrow$ GetControl($x_t, \{x_i^*\}$)
9: Execute(u_t)
10: $x_{t+1} \leftarrow$ get robot's (control) state
11: history $\leftarrow$ history $\cup \{(x_t, u_t, x_{t+1})\}$
12: **while** $\|x_t - x_t^*\| < \delta$
13: UpdateDiscrepancyModel(history)

A. Planning with control-level penalties

Given the problem in Sec. III-A, we use A* search [25] to plan a least-cost path to goal. However, we use a *penalized* cost function (similar to [5]), of the form

$$c(s,a) + \text{ControlLevelPenalty}(s,a) \quad (2)$$

for taking action a at state s. This additional *control-level penalty* translates the control-level's discrepancy model, which we learn through observing the robot's executions in the real-world, to a planning-level bias. This bias translates discrepancy information from the control-level to the planning-level. This control-level information is more fine-grained and hence typically very informative to the planner, since the controller operates at a much higher frequency than the planner.

Alg. 2 shows how we compute the control-level penalty

for action a at state s. First, FWDSIMCONTROLLER returns the sequence of control inputs $u_1, u_2, \ldots, u_{T-1}$ that the controller would produce to follow a from state x, ignoring any model discrepancies. In Sec. III-B, we remarked that there are very few constraints on the controller design; in fact, the only requirement is that we are able to forward-simulate the controller in this way to generate this sequence of inputs.

Next, we predict the state of the robot after taking the sequence of control inputs $u_1, \ldots, u_{T-1}$ from state x. Specifically, PREDICT predicts the distribution $p(x_T \mid x_1, u_1, \ldots, u_{T-1}) = \mathcal{N}(\mu_{x_T}, \Sigma_{x_T})$ over the final state x_T, given the initial state x, and sequence of controls $u_1, \ldots, u_{T-1}$. To do so, we forward propagate the uncertain discrepancy d through the dynamics, based on our latest estimate. In general, there exist many ways to do this, and we describe our particular implementation in Sec. IV-B.

Given our prediction of x_T, we can evaluate the probability that there will be a discrepancy between what the planning model predicts and the actual outcome of a. This is determined by whether x_T differs from x_T^* by δ or more, as in Alg. 2 line 5. In practice, we estimate this probability, denoted p, empirically via sampling. Finally, we return a penalty proportional to p.

Algorithm 2 Evaluating the control-level penalty.

1: **function** CONTROLLEVELPENALTY((s, a))
2: $x \leftarrow \phi(s)$
3: $u_1, \ldots, u_{T-1} \leftarrow$ FWDSIMCONTROLLER(x, a)
4: $\mu_{x_T}, \Sigma_{x_T} \leftarrow$ PREDICT$(x, u_1, \ldots, u_{T-1})$
5: $p \leftarrow P(\|x_T - x_T^*\| > \delta \mid x, u_1, \ldots, u_{T-1})$
6: **return** $p \cdot c_{\text{penalty}}$

B. Prediction using the discrepancy model

Here, we describe our implementation of PREDICT. Our goal in this section is to derive a principled yet computationally simple way to predict forward in time the distribution over the state under our GP-based discrepancy model. Aside from the approach presented here, however, there exist many other methods for forward simulating using a GP model; see, for example, recent work in [26].

For simplicity, we assume that f is linear. That is:

$$x_{t+1} = Ax_t + Bu_t + B_d d_t \tag{3}$$

where $d_t = d(x_t, u_t)$ is the predicted discrepancy at (x_t, u_t) defined in III-C, and A, B, and B_d are the appropriately-sized matrices. If f is not linear, we can add an additional step to the procedure below, where we linearize f around the current x and u, to find approximate A, B, and B_d.

We assume that the initial state x_1 is known. However, the state at the next time step x_2 is unknown due to the uncertain effect of d. Specifically, $p(x_2 \mid x_1, u_1)$ is Gaussian with the following mean and covariance:

$$\mu_{x_2} = Ax_1 + Bx_1 + B_d \begin{bmatrix} \mu_1(x_1, u_1) \\ \vdots \\ \mu_l(x_1, u_1) \end{bmatrix} \tag{4}$$

$$\Sigma_{x_2} = B_d \begin{bmatrix} \sigma_1^2(x_1, u_1) & & 0 \\ & \ddots & \\ 0 & & \sigma_l^2(x_1, u_1) \end{bmatrix} B_d^\top \tag{5}$$

where μ_i and σ_i^2 are the mean and variance functions described in Sec. III-C.

Then, in general at time step t, given the mean and covariance of x_t, μ_{x_t} and Σ_{x_t}, respectively, we approximate the mean and covariance of x_{t+1} as:

$$\mu_{x_{t+1}} = A\mu_{x_t} + Bu_t + B_d\mu(x_t, u_t) \tag{6}$$

$$\Sigma_{x_{t+1}} = A\Sigma_{x_{t+1}}A^\top + B_d\text{Cov}(d_t, d_t)B_d^\top + A\text{Cov}(x_t, d_t)B_d^\top + B_d\text{Cov}(d_t, x_t)A^\top. \tag{7}$$

To arrive at Eq. 6 and Eq. 7, we use the same 1st- and 2nd-order Taylor series-based approximation approach as described in [27]; the full derivation can be found there.

We implement PREDICT by recursively computing μ_{x_t} and Σ_{x_t} for $t = 1, 2, \ldots, T$ using Eq. 4-7, and returning μ_{x_T}, Σ_{x_T}. We illustrate an example of these mean and covariance predictions in Fig. 2 (upper right).

C. Updating the discrepancy model

When the robot deviates by δ from the planned path, i.e., $\|x_t - x_t^*\| \geq \delta$ (Alg. 1, line 12), we update the control-level discrepancy model with the execution history. To do so, we first construct a data set $\{\hat{d}_t\}$, where

$$\hat{d}_t = x_{t+1} - f(x_t, u_t), \tag{8}$$

is the observed disturbance, for each (x_t, u_t, x_{t+1}) in the execution history. We illustrate these $\hat{d}_t$ by the blue arrows in Fig. 2 (lower left). Finally, we update our GP model $d(x, u)$ defined in Sec. III-C, with the new data $\{\hat{d}_t\}$.

V. EXPERIMENTAL EVALUATION

A. Setup

Planning-level. The planning state is the x- and y-position of the robot. At each state, there are 4 actions: move in the $+x$-, $-x$-, $+y$-, and $-y$-directions by 0.1 m. Each action has cost 0.1. The planner uses an A* search with a standard Manhattan distance heuristic. To postprocess the path, we fit a piecewise polynomial to it, which we then pass to the controller.

Control-level. The control state is the x- and y-position, and yaw angle θ (i.e., angle about the z-axis) of the robot. The robot is holonomic, so the control inputs are simply linear velocities in x and y, and an angular velocity about z.

B. Simulation

We conducted a set of simulation experiments primarily to evaluate the efficiency of our approach at completing tasks, compared to state-of-the-art approaches. For the simulation experiments, we used the Gazebo-based Astrobee simulator [28].

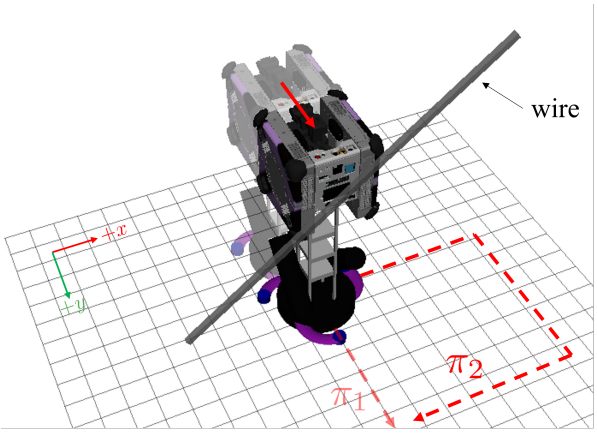

Fig. 3: A wire obstructs the robot's path to goal π_1. An alternate path, such as π_2, is required to reach the goal.

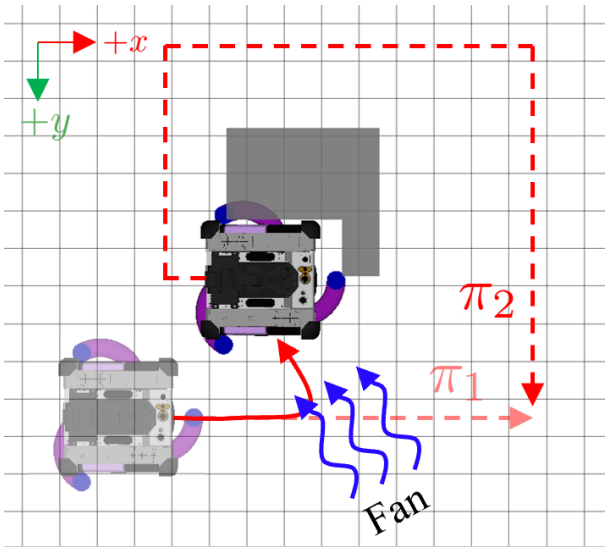

Fig. 4: A simulated fan exerts a force on the robot, indicated by the blue arrows. Unaware of the fan, the robot initially plans path π_1, but is pushed into the obstacle. An alternate path, such as π_2, is required to complete the task.

Methods. We compared our approach to two baselines. First, replanning, which simply replans a path to goal whenever the robot deviates too far from the plan. This is a simple, yet widely used method in hierarchical planning and control systems, described in Sec. II. Second, CMAX [5], which inflates the costs of all states and actions in a small neighborhood around previously observed model discrepancies. While different choices of neighborhood size can affect CMAX's performance, how to choose the best size per domain remains an open question. In our experiments, we fixed this size to $0.075\ m$.

Measures. To measure efficiency, we considered the total time to complete the task (this includes both the planning and execution times), and the total distance traveled. On the planning side, we measured the planning times, and the number of replans.

Scenarios. We ran 50 randomized instances of the following

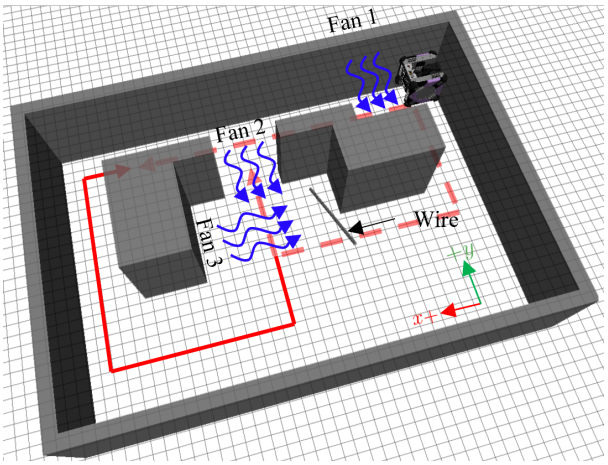

Fig. 5: 3 fans and 1 wire obstruct the robot's path to goal. We show several alternate paths in red.

3 scenarios. The environment in S1 and S2 is $2\ m \times 2\ m$, and $3.6\ m \times 2.4\ m$ in S3.

- S1 (Wire) The robot navigates to the goal, but an unmodelled wire obstructs its path, shown in Fig. 3. The wire is always at the origin, and we randomize its size in x ($[0.05\ m, 0.15\ m]$) and y ($[0.25\ m, 0.75\ m]$). For simplicity, we model the wire as a rigid-body.
- S2 (Fan) The robot navigates to the goal, but there is an unmodelled fan in the environment, as shown in Fig. 4. We simulate the fan as exerting an external force on the robot when it is within a specified region near the fan. There is also an obstacle at the center of the environment. We randomize over the angle ($[10°, 45°]$) and magnitude ($[0.25\ N, 0.35\ N]$) of the fan, and the obstacle's size ($[0.23\ m, 0.53\ m]$ in both x and y).
- S3 (3 Fans + Wire) The robot navigates the environment shown in Fig. 5, which contains 3 fans and 1 wire. We randomize the magnitude and angle of each fan the same as in S2. We randomly place the wire in the freespace in the lower half of the environment shown in Fig. 5, and randomize its length ($[0.02\ m, 0.20\ m]$) and angle ($[-20°, 20°]$), and use the same rigid-body model as in S1.

Analysis. Across all scenarios, the robot completes the task significantly faster than CMAX using our approach, as shown in Fig. 6. Furthermore, the margin by which our approach is more efficient, with respect to both time (Fig. 6) and distance traveled (Table I) increases as the complexity of the scenario increases, from S1 being the simplest to S3 being the most complex.

We found that replanning was unable to complete any task. In all experiments, the planner continues to produce a path that passes through the discrepancy region, and the robot gets stuck. This validates that we chose scenarios where changing the model or planner is necessary to complete the task.

In S1, the control-level information provides little additional insight into the size of the region where the model is inaccurate– the robot is stopped in place by the wire obstructing its path. Consequently, our approach shows the

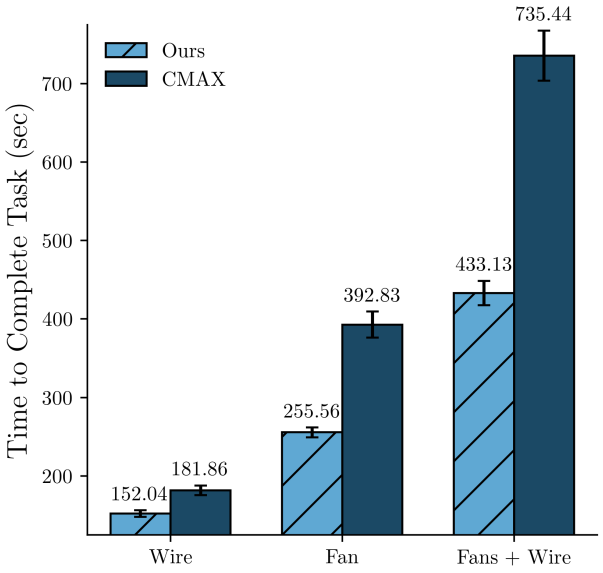

Fig. 6: The mean and standard errors of time to complete the task, for the wire (S1), fan (S2), and 3 fans + 1 wire (S3) scenarios.

	S1	S2	S3
Replan	N/A	N/A	N/A
CMAX	2.46 (0.08)	7.95 (0.35)	13.79 (0.69)
Ours	2.53 (0.07)	**5.41 (0.12)**	**8.62 (1.64)**

TABLE I: Mean distance traveled (m) by the robot in each scenario, with standard error in parentheses.

smallest gain in efficiency with respect to time, and no significant difference compared to CMAX with respect to distance. However, as the complexity of the scenarios increase in S2 and S3, this control-level information becomes more valuable– our approach is able to more quickly determine that the model is wrong in a larger region by observing the performance of the path following controller as the robot attempts to move through the fan's air current. That is why we see a larger advantage for our approach over CMAX in these scenarios, with respect to both time and distance.

One drawback of our approach, however, is increased planning time, as shown in Table II. While our approach leads to lower overall time to complete the task (including fewer number of replans, as shown in Table III), each planning call is more expensive. The additional computation time is largely due to the PREDICT routine, which performs a series of expensive GP-related computations, as described in Sec. IV-B (in some cases, PREDICT took up to 90% of the planning time).

C. Real-robot

To demonstrate our approach on a real-robot, we ran our planner on-board NASA's Astrobee free-flyer; for more details about Astrobee's sensing, localization, propulsion, etc., see [4, 28]. Astrobee is designed for zero-gravity, but we utilized a 2 $m \times 2$ m low-friction surface in a lab at NASA. We show our full results, comparing our approach with replanning and CMAX, in the accompanying video.

	S1	S2	S3
CMAX	**1.96e-6 (1.00e-7)**	**8.65e-5 (1.06e-5)**	**7.10e-4 (2.35e-5)**
Ours	8.35e-2 (4.56e-3)	9.93e-2 (5.12e-3)	9.42e-2 (3.34e-3)

TABLE II: Mean planning time in seconds, with standard error in parentheses.

	S1	S2	S3
CMAX	10.2 (0.4)	16.2 (0.3)	29.8 (2.2)
Ours	**4.6 (0.2)**	**5.0 (0.2)**	**8.6 (1.6)**

TABLE III: Mean number of replans (standard error in parentheses).

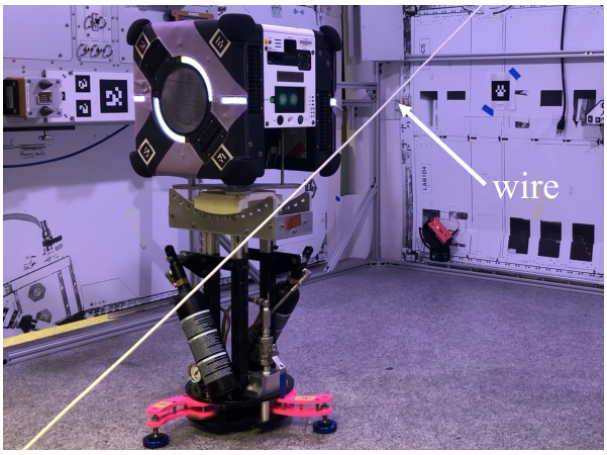

Fig. 7: Astrobee plans a path to the goal, but the path is obstructed by an unseen wire.

Fan. Similar to S1, as shown in Fig. 1. The planner finds a path around the other side of the obstacle (Fig. 1, solid line), to avoid the fan and reach the goal.

Wire. Similar to S2, as shown in Fig. 7. Unlike the simulation, the robot can actually get stuck in the wire, and there is also some flexibility and elasticity in the wire, adding additional challenge. The planner is eventually able to find a path to the goal around the wire.

VI. CONCLUSION AND FUTURE WORK

In this paper, we have presented an approach to enable robots to complete tasks despite planning with an inaccurate model. We utilize control-level information to bias replanning away from where we infer the model to be inaccurate. We found that our approach led to significantly more efficient robot behavior compared to baselines in simulation. Finally, we demonstrated our approach running successfully in lab experiments on NASA's Astrobee free-flyer.

Despite the advantages, our approach introduces additional overhead, leading to increased planning times. We plan to investigate ways to address this through algorithmic improvements and optimizations. Furthermore, when an unmodelled disturbance actually helps the robot, such as a fan that pushes it towards the goal, our approach may not perform well. It is important to investigate such cases in future work. Finally, we plan to investigate more complex applications – such as environments with multiple agents or movable objects.

REFERENCES

[1] D. Nguyen-Tuong and J. Peters. "Model learning for robot control: a survey". *Cognitive processing* 12.4 (2011).

[2] A. S. Polydoros and L. Nalpantidis. "Survey of model-based reinforcement learning: Applications on robotics". *Journal of Intelligent & Robotic Systems* 86.2 (2017).

[3] J. F. Fisac, A. K. Akametalu, M. N. Zeilinger, et al. "A general safety framework for learning-based control in uncertain robotic systems". *IEEE Transactions on Automatic Control* 64.7 (2018).

[4] M. Bualat, J. Barlow, T. Fong, et al. "Astrobee: Developing a free-flying robot for the international space station". *AIAA SPACE 2015 Conference and Exposition*. 2015.

[5] A. Vemula, Y. Oza, J. A. Bagnell, and M. Likhachev. "Planning and execution using inaccurate models with provable guarantees". *Robotics: Science and Systems (RSS)*. 2020.

[6] A. Vemula, J. A. Bagnell, and M. Likhachev. "CMAX++: Leveraging experience in planning and execution using inaccurate models". *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 7. 2021.

[7] T. Pan, A. M. Wells, R. Shome, and L. E. Kavraki. "Failure is an option: Task and Motion Planning with Failing Executions". *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022.

[8] D. McConachie, T. Power, P. Mitrano, and D. Berenson. "Learning when to trust a dynamics model for planning in reduced state spaces". *IEEE Robotics and Automation Letters* 5.2 (2020).

[9] P. Mitrano, D. McConachie, and D. Berenson. "Learning where to trust unreliable models in an unstructured world for deformable object manipulation". *Science Robotics* 6.54 (2021).

[10] M. Noseworthy, C. Moses, I. Brand, et al. "Active learning of abstract plan feasibility". *Robotics: Science and Systems (RSS)*. 2021.

[11] S. Kuindersma, R. Deits, M. Fallon, et al. "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot". *Autonomous robots* 40.3 (2016).

[12] J. Norby and A. M. Johnson. "Fast global motion planning for dynamic legged robots". *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020.

[13] C. Urmson, J. Anhalt, D. Bagnell, et al. "Autonomous driving in urban environments: Boss and the urban challenge". *Journal of field Robotics* 25.8 (2008).

[14] S. Thrun, M. Montemerlo, H. Dahlkamp, et al. "Stanley: The robot that won the DARPA Grand Challenge". *Journal of field Robotics* 23.9 (2006).

[15] M. Chen, S. L. Herbert, H. Hu, et al. "Fastrack: a modular framework for real-time motion planning and guaranteed safe tracking". *IEEE Transactions on Automatic Control* 66.12 (2021).

[16] S. Kousik, S. Vaskov, M. Johnson-Roberson, and R. Vasudevan. "Safe trajectory synthesis for autonomous driving in unforeseen environments". *Dynamic Systems and Control Conference*. Vol. 58271. American Society of Mechanical Engineers. 2017.

[17] S. Singh, M. Chen, S. L. Herbert, et al. "Robust tracking with model mismatch for fast and safe planning: an SOS optimization approach". *International Workshop on the Algorithmic Foundations of Robotics*. Springer. 2018.

[18] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek. "Sequential composition of dynamically dexterous robot behaviors". *The International Journal of Robotics Research* 18.6 (1999).

[19] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts. "LQR-trees: Feedback motion planning via sums-of-squares verification". *The International Journal of Robotics Research* 29.8 (2010).

[20] A. Majumdar and R. Tedrake. "Funnel libraries for real-time robust feedback motion planning". *The International Journal of Robotics Research* 36.8 (2017).

[21] B. K. Styler and R. Simmons. "Robust Efficient Robot Planning through Varying Model Fidelity". *Third International Workshop on Planning and Robotics (PlanRob)*. 2015.

[22] B. Styler and R. Simmons. "Plan-time multi-model switching for motion planning". *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 27. 2017.

[23] A. K. Akametalu, J. F. Fisac, J. H. Gillula, et al. "Reachability-based safe learning with Gaussian processes". *53rd IEEE Conference on Decision and Control*. IEEE. 2014.

[24] C. E. Rasmussen. "Gaussian processes in machine learning". *Summer school on machine learning*. Springer. 2003.

[25] P. E. Hart, N. J. Nilsson, and B. Raphael. "A formal basis for the heuristic determination of minimum cost paths". *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968).

[26] L. Hewing, E. Arcari, L. P. Fröhlich, and M. N. Zeilinger. "On simulation and trajectory prediction with gaussian process dynamics". *Learning for Dynamics and Control*. PMLR. 2020.

[27] A. Girard, C. Rasmussen, and R. Murray-Smith. "Gaussian process priors with uncertain inputs: multiple-step-ahead prediction, delovno poro-£ ilo DCS TR-2002-119". *University of Glasgow, Glasgow* (2002).

[28] L. Fluckiger, K. Browne, B. Coltin, et al. "Astrobee robot software: A modern software system for space". *iSAIRAS (International Symposium on Artificial Intelligence, Robotics and Automation in Space)*. ARC-E-DAA-TN55483. 2018.