DROP: Dexterous Reorientation via Online Planning

Albert H. Li[†], Preston Culbertson[‡], Vince Kurtz[‡], Aaron D. Ames[†,‡]

***Abstract*— Achieving human-like dexterity is a longstanding challenge in robotics, in part due to the complexity of planning and control for contact-rich systems. In reinforcement learning (RL), one popular approach has been to use massively-parallelized, domain-randomized simulations to learn a policy *offline* over a vast array of contact conditions, allowing robust sim-to-real transfer. Inspired by recent advances in real-time parallel simulation, this work considers instead the viability of *online planning* methods for contact-rich manipulation by studying the well-known in-hand cube reorientation task. We propose a simple architecture that employs a sampling-based predictive controller and vision-based pose estimator to search for contact-rich control actions online. We conduct thorough experiments to assess the real-world performance of our method, architectural design choices, and key factors for robustness, demonstrating that our simple sampling-based approach achieves performance comparable to prior RL-based works. Supplemental material: https://caltech-amber.github.io/drop.**

I. Introduction

Achieving dexterity comparable to human hands has been a longstanding challenge in robotics research. While even simple robots can produce dynamic, contact-rich behavior [1], [2], general methods for achieving these behaviors are still scarce. Thus far, reinforcement learning (RL) has been the dominant paradigm for many contact-rich tasks due to its ability to generate real-world robust plans. One such well-studied task is in-hand cube reorientation, where a hand must rotate a cube to match as many consecutive goal orientations as possible. Pioneered by OpenAI [3] and further studied by [4], [5], RL policies trained with massively-parallelized, domain-randomized simulations have achieved remarkably robust sim-to-real transfer for cube rotation. But, this *offline* simulation-based approach requires substantial pre-execution computation and is inflexible to changing task specifications.

Leveraging recent advances in real-time parallel simulation (e.g., MJPC) [6], we instead study the *online* approach of sampling-based predictive control (SPC) methods for in-hand manipulation, which continuously replan by simulating parallel rollouts and applying optimal control actions over short time horizons. In contrast with RL, such online planning methods can adjust the task or model without re-training, but may demand expensive online computation. While tools like MJPC have made SPC feasible for many simulated contact-rich tasks, their real-world utility remains largely unproven.

† A. H. Li and A. D. Ames are with the Department of Computing and Mathematical Sciences, California Institute of Technology, Pasadena, CA 91125, USA, {alberthli, ames}@caltech.edu.

‡ P. Culbertson, V. Kurtz, and A. D. Ames are with the Department of Civil and Mechanical Engineering, California Institute of Technology, Pasadena, CA 91125, USA, {pculbert, vkurtz, ames}@caltech.edu.

* This research is supported by Dow.

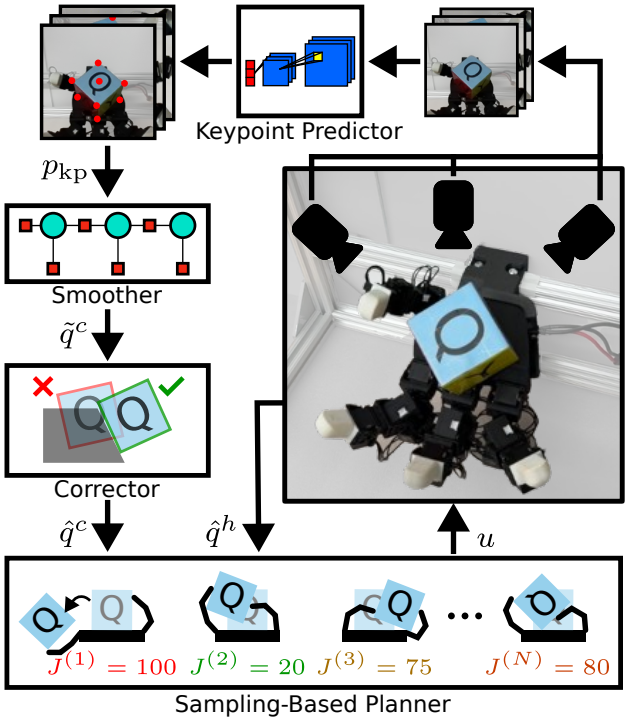

Fig. 1: **The DROP architecture.** DROP consists of (i) a vision-based cube pose estimator (composed of the Keypoint Predictor, Smoother, and Corrector), and (ii) a sampling-based planner that selects control actions by conducting model-based rollouts and iteratively improving the sampling distribution online based on the costs $J^{(i)}$.

Our main contribution is DROP (**D**exterous **R**eorientation via **O**nline **P**lanning), a system architecture (Fig. 1) that consists of (i) a simple sampling-based planner and (ii) a vision-based state estimator comprising a keypoint detection model, a fixed-lag smoother, and a collision-aware state corrector. Our aim is not to design the best possible cube reorientation policy, but to test the viability of simple sampling-based strategies with thorough experiments to assess DROP's real-world performance, architectural design, and key factors for robustness. Still, DROP is the first demonstration of an online planning method for cube reorientation on hardware, and we find that its performance is comparable with prior RL-based methods. Finally, we provide open-source code, estimator weights, and hardware setup to facilitate reproducibility in contact-rich dexterous manipulation[1].

II. Background and Preliminaries

A. Related Work

The first viable methods for in-hand cube reorientation employed ***reinforcement learning (RL)***, starting with Dactyl

[1]Project website: https://caltech-amber.github.io/drop [7].

[3], which popularized domain randomization for sim-to-real transfer of RL policies. Subsequent research built on Dactyl with improved domain randomization [4], tactile-only sensing [8], [9], and "palm-down" rotations for various objects [5], [10]. While typically limited to specific robots or object classes, RL's robust hardware performance has established it as the standard for in-hand manipulation.

Others have explored model-based approaches to contact-rich manipulation via ***contact-implicit trajectory optimization*** (CITO). Originally used in locomotion [11], [12], recent work [6], [13] shows that such planners can generate motions for simple contact-rich tasks like bimanual lifting [14] or real-time planar rolling and sliding [15], [16]. Other approaches have combined CITO with learned signed distance fields [17], global search [18], or smoothed dynamics [19].

Sampling-based planning has a long history in manipulation, starting with methods like RRT, PRM, and STOMP [20]–[22]. Recent advances in SPC [23], [24] have enabled simple contact-rich manipulation like pushing, sliding, and pick and place [25], [26], while new tools for real-time parallel simulation of contact-rich tasks [6], have made SPC viable for simulated contact-rich tasks with remarkably simple controllers. Still, little evidence exists to show the viability of SPC for real-world contact-rich manipulation, and overall, model-based methods (like SPC or CITO) have seldom studied tasks as hard as cube reorientation, which exhibits many simultaneous unpredictable contact modes.

Lastly, high-quality ***object tracking*** is key for most of these methods, especially model-based online search. RL approaches [3], [4] usually use deep neural networks for single-object pose estimation, but recent computer vision advances suggest that more general tracking pipelines like frame-to-frame point trackers [27]–[29] and foundation models for rigid body pose prediction [30] may boost performance.

B. Mathematical Notation

We model the continuous-time cube-in-hand dynamics as

$$\dot{x}(t) = f\left(x(t), u(t)\right), \tag{1}$$

where x is the state and u are control inputs. We do not assume access to a closed-form expression for f, but instead to a simulator (MuJoCo [31]) that generates state trajectories $x(t)$ given an initial state x_0 and a control sequence $u(t)$.

The state x consists of the generalized positions and velocities of both the cube (q^c, v^c) and hand (q^h, v^h):

$$x = [q, v] = [q^c, q^h, v^c, v^h]. \tag{2}$$

In this work, the control actions u are position setpoints for the hand, which are tracked by a lower-level PD controller internal to the model f. Following [6], we represent a control trajectory $u(t)$ on $t \in [0, T]$ with K spline knots,

$$U = [u_1, u_2, \ldots, u_k, \ldots, u_K], \tag{3}$$

which constitute the decision variables for online planning. The control objective is encoded by a cost functional

$$J(U; x_0) = \int_0^T \ell\left(x(t), u(t)\right) dt, \tag{4}$$

where querying $J(U; x_0)$ requires simulating f using an open-loop control signal $u(t)$ from initial condition x_0.

Throughout the paper, we use the following notation to denote "subtraction" of two quaternions $a, b \in \mathbb{H}$:

$$a \ominus b := \operatorname{Log}\left(a \circ b^{-1}\right) \in \mathfrak{so}(3), \tag{5}$$

and similarly, when $a, b \in SE(3)$, $a \ominus b \in \mathfrak{se}(3)$.

III. The DROP Algorithm

DROP consists of (i) a sampling-based planner and (ii) a keypoint-based cube pose estimator (see Fig. 1). The planner continually updates the control spline U via SPC. We use MJPC [6] to handle both parallel rollouts and policy updates.

A. Sampling-Based Planning and Control

Algorithm 1 describes a generic SPC procedure. At time $t \in \mathbb{R}$, the planner receives a state estimate $\hat{x}(t)$ and rolls out a batch of N open-loop control sequences $U^{(i)}$ drawn from some parametric distribution $\pi_\theta\left(U\right)$. Each control trajectory is simulated (in parallel) to obtain a cost $J^{(i)}$, and all costs are jointly used to update the sampling parameters θ. The control input $u(t)$ can be obtained for any time t from the spline parameters U. This allows planning to proceed asynchronously, with the parameters θ updated as quickly as computational limits allow.

Algorithm 1: Sampling-based Predictive Control

Input: θ, N, planner-specific parameters.
while *planning* **do**
  $x_0 \leftarrow \hat{x}(t)$; `// estimate curr state`
  for $i = 1$ *to* N ; `// multi-threaded`
  do
    $U^{(i)} \sim \pi_\theta(U)$; `// sample controls`
    $J^{(i)} \leftarrow J\left(U^{(i)}; x_0\right)$; `// eval rollout`
  $\theta \leftarrow$ `update_params` $\left(U^{(1:N)}, J^{(1:N)}\right)$;
  $u(t) \leftarrow$ `get_action`(θ, t) ; `// asynchronous`

In our experiments, we test two simple planning strategies that both use a diagonal Gaussian distribution $\pi_\theta\left(U\right) = \mathcal{N}\left(\overline{U}, \Sigma\right)$ with parameters $\theta = \left(\overline{U}, \Sigma\right)$. In Algorithm 1, `get_action`$(\theta, t)$ is a spline interpolation with knots $\overline{U}$.

Predictive sampling (PS) repeatedly updates $\overline{U}$ to be the best sample, fixing $\Sigma = \sigma^2 I$. Despite its simplicity, PS has demonstrated surprisingly effective performance on complex robot manipulation and locomotion tasks in simulation [6].

The **cross-entropy method (CEM)** instead fits both $\overline{U}$ and $\Sigma = \operatorname{diag}(s)$ to the sample mean and variance of the M best *elite samples*, where s is a vector of covariances. CEM is only marginally more complex than PS, but has long been used for general gradient-free optimization [32].

As in prior work [4], the task considered in this paper is to use a dexterous hand to rotate a cube to within 0.4 rad of as many goal orientations in a row as possible without dropping. The goals are uniformly randomly sampled over $SO(3)$. In

contrast with [4], to ensure sufficient task difficulty, each new goal must be at least 90° from the prior one.

Mathematically, the DROP cube reorientation problem is expressed as the optimal control problem

$$\begin{aligned} \min_{u} \int_0^T \Big\{ \lambda_{\mathrm{g}} \cdot \ell_{\mathrm{g}}(r^c(t)) + \lambda_{\mathrm{p}} \cdot \ell_{\mathrm{p}}(p^c(t)) \Big\} dt \\ \text{s.t.} \quad \dot{x}(t) = f(x(t), u(t)), \\ x(0) = x_0, \end{aligned} \tag{6}$$

where $\lambda_{(\cdot)}$ denote weights, the dynamics $f(x, u)$ are only available through simulation, and

$$\ell_{\mathrm{g}}(r^c) := \left\| r^c \ominus r^c_{\mathrm{goal}} \right\|_2^2, \tag{7}$$

$$\ell_{\mathrm{p}}(p^c) := \mathrm{dist}_{\mathcal{S}}(p^c), \tag{8}$$

are running costs. The variables p^c and r^c denote the positional and rotational components of the cube pose $q^c = [p^c, r^c]$, which are extracted from the simulated state $x(t)$. ℓ_{g} penalizes rotational distance from the goal, while ℓ_{p} penalizes the cube leaving a "safe" region $\mathcal{S}$ in Cartesian space.

In this work, we prioritize drop reduction by setting λ_{p} high and choosing a conservative region $\mathcal{S}$ (for details, see [7]). In practice, we use a relatively low λ_{g}, which slows down the planner but also amplifies differences in rotation rate across methods, allowing easier quantitative comparison.

B. The Pose Estimation Pipeline

Our pose estimator consists of three parts: a keypoint predictor, a fixed-lag smoother, and a collision-aware corrector.

Keypoint Prediction. The estimator takes in images $I_c \in \mathbb{R}^{C \times H \times W}$ from n_c cameras. We first train a vision model g_φ that predicts 8 fixed keypoints on the cube corresponding to its corners, $p_{\mathrm{kp}} = g_\varphi(I_c)$. The keypoint prediction task is supervised from a training dataset of 686,000 images of a simulated cube rendered by `Blender`, which includes ground-truth pixel locations for all keypoints, even those that are outside the frame or occluded. We generate randomized background scenes using `kubric` [33], which spawns the cube along with other random assets in a `pybullet` simulation. Similar datasets are commonly used to train "track-any-point" models [27]–[29] which exhibit strong sim-to-real transfer for similar keypoint tracking tasks.

Crucially, we then augment these images with random affine transforms; visual effects like color, shadow, and contrast; and randomized backgrounds. We also found that pruning images where the cube was nearly occluded, or too close for a reliable pose estimate, was essential for good performance. Figure 2 shows some of the resulting images.

To train g_φ, we fine-tuned an ImageNet-pretrained `resnet18` using `AdamW` with an MSE loss. The only model adjustments were the number of input channels (depending on RGB or RGBD inputs) and the dimension of the output layer. For finer details, see [7, Extended Version].

Pose Smoothing. We use GTSAM [34], a factor graph-based state estimation package, to convert keypoints into a cube pose estimate via fixed-lag smoothing. Given a fixed pinhole camera model with known camera poses and cube size, we derive keypoint measurement factors that relate keypoints p_{kp} to a cube pose q^c. This allows GTSAM to fuse keypoint predictions from an arbitrary number of cameras in real-time to yield a smoothed cube pose estimate $\tilde{q}^c$. See our open-source implementation for exact details [7].

Fig. 2: **Image augmentations.** To train the keypoint predictor, we augmented simulated images of the cube with random crops, affine transformations, spliced backgrounds, random deletions, and visual adjustments in color, contrast, brightness, and reflectivity.

In both simulation and hardware, we estimate velocities by numerically differentiating position estimates (drawn from the smoother for the cube and joint encoders for the hand) and applying an exponential moving average filter with parameter $\alpha = 0.1$ to compute a smooth velocity estimate $\tilde{v}$.

Collision-Aware Correction. While the smoother's cube pose estimate $\tilde{q}^c$ is usually accurate within $1\mathrm{cm}$, it may not always be physically compatible with the hand configuration $\tilde{q}^h$ from the encoders, as the smoother has no knowledge of collision dynamics. Thus, $\tilde{q} = [\tilde{q}^c, \tilde{q}^h]$ often corresponds to non-negligible interpenetration between cube and hand, which (i) leads to inaccurate plans that destabilize the closed-loop system, and (ii) decreases the planning rate, as stiff MuJoCo models with high interpenetration are poorly-conditioned, requiring more solver iterations.

To remedy this, we adapt the method of [35] by using a *corrector*, which maintains an asynchronous internal model

$$\dot{\hat{x}} = \hat{f}(\hat{x}(t), u(t)) \tag{9}$$

with state $\hat{x} = [\hat{q}^c, \hat{q}^h, \hat{v}^c, \hat{v}^h]$.

The corrector receives an un-adjusted estimate $\tilde{x}$ from the smoother and encoders, and computes a "corrective wrench"

$$w = -C_P(\hat{q}^c \ominus \tilde{q}^c) - C_D(\hat{v}^c - \tilde{v}^c) \tag{10}$$

that is added to (9) as a generalized force. Since the corrector starts in a feasible state, simulating (9) results in a corrected estimate $\hat{x}$ that is always physically feasible, but pulled toward the (possibly infeasible) vision-based estimate $\tilde{x}$. We also add a constant corrective force in the direction of gravity to counteract any smoother estimates that may unrealistically pull the cube upwards due to high corrector gains C_P, overwhelming the natural gravitational forces of (9).

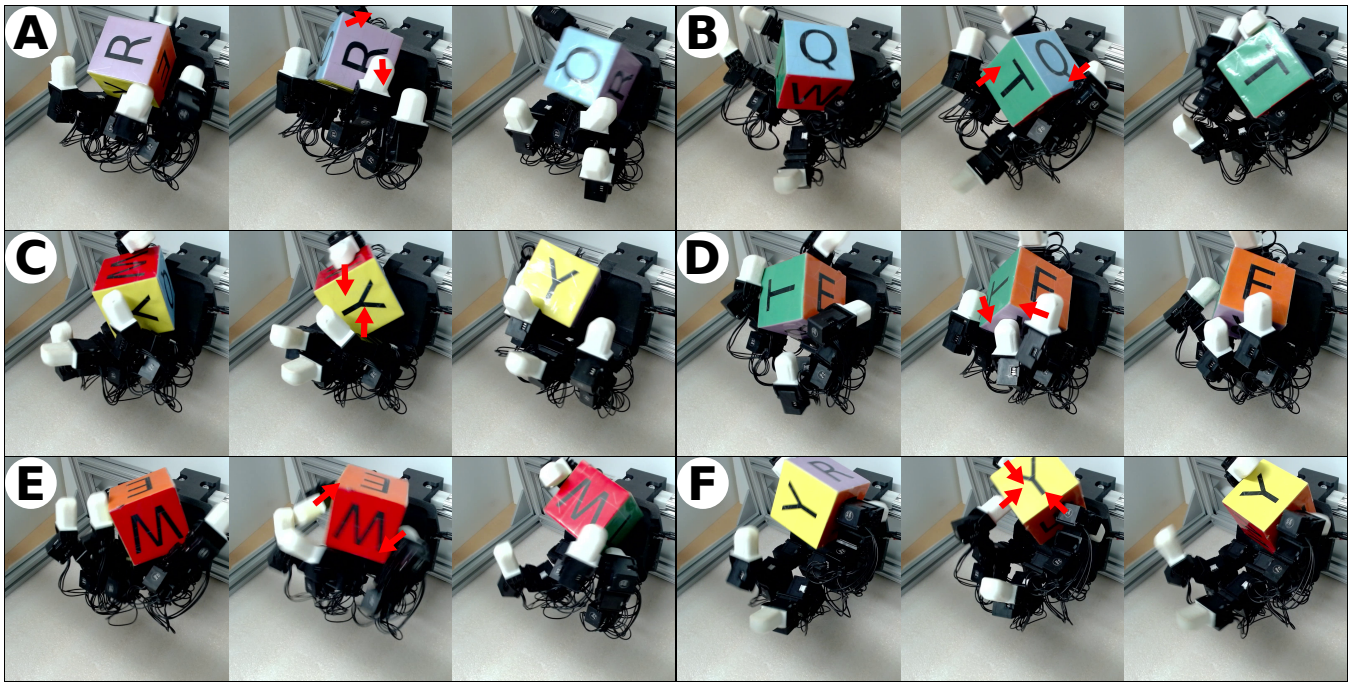

Fig. 3: **Examples of rotations.** CEM can discover many contact-rich plans for cube reorientation. The red arrows show where forces are primarily applied to achieve rotations. **(A)** The middle finger pushes down on a cube edge while the base of the thumb lifts the opposite corner, rotating the Q face up. **(B)** The ring finger and base of the index finger push on opposite corners to rotate the T face up. **(C)** The thumb pulls down on the W face while the base of the ring finger pushes on the opposite corner to rotate the Y face up. **(D)** The index finger pushes down on the edge of the T face while the ring finger swipes left on the E face to rotate it up. **(E)** The ring finger first swipes inwards, then the thumb quickly follows to pull the W face up. **(F)** The thumb and the ring finger pinch and lift the cube, then the index finger pushes on an edge to rotate the Y face up. The cube is calmly lowered onto the palm.

IV. Experiments

For the remainder of the paper, we thoroughly evaluate the DROP architecture via hardware and simulation experiments designed to answer the following questions:

1) Can DROP perform robust and dynamic cube reorientation in hardware (Sec. IV-B)?
2) How do different components of DROP contribute to its overall performance and reliability (Sec. IV-C)?
3) How do modeling and state estimation error affect DROP's performance (Sec. IV-D)?

Overall, we find that DROP can reliably perform cube reorientation in hardware, achieving performance comparable with RL-based methods while exhibiting surprising robustness and dexterity. We emphasize that the goal of our experiments is not to design the best-possible control strategy, but to assess the viability of sampling-based online planning for the cube reorientation task via thorough evaluations of the DROP architecture. Thus, we leave systematic comparisons of DROP with other methods for future work.

A. Hardware Setup and Experimental Details

For all experiments, we perform in-hand rotation of a 3D-printed cube with 7cm side lengths and a fixed-base LEAP hand [36] with palm angled downwards at 20° so that the cube slides off without intervention (see Fig. 1 and 3). We use a 128-thread Ryzen Threadripper Pro 5995wx CPU to plan rollouts in parallel. To capture images, we use three ZED Mini cameras with RGBD channels and perform keypoint estimation on 256x256 center-cropped images at VGA resolution. The estimator runs at about 90Hz, while the planner frequency fluctuates between 25-50Hz.

For our trials, we compare three planners: PS and CEM (the two sampling-based methods discussed in Sec. III-A), as well as iLQR [37], a gradient-based method. PS and CEM use $N = 120$ rollouts while iLQR devotes 120 threads to parallel line search. CEM uses $M = 4$ elite samples. When tuning hyperparameters (such as cost weights), we prioritize robustness at the expense of rotation speed. For all experiments, we used identical costs, code, and hyperparameters for performing simulation rollouts, state estimation, and communicating with hardware. Unlike previous work [3], [4], we did not observe any significant degradation of our hardware stack (e.g., overheating) during testing.

B. Main Hardware Results

We begin by presenting results of hardware trials using the full stack shown in Fig. 1 and discussed in Sec. III. We study each planner by running 10 trials of the cube reorientation task and analyzing the associated rotation and timing statistics. For examples of interesting rotations, see Fig. 3, and for quantitative results, see Table IB. Following prior work, we report statistics assuming the run ends if 80s have elapsed without reaching a goal. Since this cutoff is arbitrary and we used a fairly conservative cost that slows rotation rate, we also report results for the same runs while ignoring the timeout period, ending only when the cube is dropped. For iLQR and PS, this did not change the results.

CEM greatly outperforms PS and iLQR. While iLQR is unable to achieve any rotations and PS only achieves single-

	Planner	Num Rots (Sorted)	Mean Rots	Median Rots	Mean Rot/s ↑	Median Rot/s ↑
(A) Prior Work	Dactyl [3]	9, 12, 13, 19, 28, 29, 29, 32, 43, 50	26.4 ± 13.4	28.5	Not Reported	Not Reported
	Dextreme [4]	1, 6, 6, 10, 10, 18, 18, 36, 61, 112	27.8 ± 19.0	14.0	Not Reported	Not Reported
(B) Main Results	iLQR	0, 0, 0, 0, 0, 0, 0, 0, 0, 0	0 ± 0	0	N/A	N/A
	PS	0, 0, 1, 1, 1, 2, 2, 3, 4, 5	1.9 ± 1.58	1.5	0.063 ± 0.070	0.033 (0.021, 0.061)
	CEM	2, 2, 11, 11, 17, 22, 27, 34, 39, 48	21.3 ± 14.8	19.5	0.090 ± 0.081	0.062 (0.038, 0.106)
	CEM (No T/O)	2, 10, 11, 11, 22, 39, 48, 53, 59, 81	33.6 ± 24.9	30.5	0.086 ± 0.085	0.063 (0.038, 0.103)
(C) CEM Ablations (No T/O)	RGB Only	1, 1, 12, 13, 20, 37, 52, 79, 94, 128	43.7 ± 41.4	28.5	0.081 ± 0.106	0.053 (0.030, 0.089)
	No Corrector	3, 4, 6, 9, 11, 14, 17, 20, 40, 97	22.1 ± 27.0	12.5	0.065 ± 0.063	0.046 (0.029, 0.080)
	Half Rollouts	2, 7, 17, 21, 21, 27, 29, 37, 44, 65	27.0 ± 17.4	24.0	0.062 ± 0.070	0.040 (0.022, 0.072)

TABLE I: **Hardware experiments. (A)** Prior works using RL for cube reorientation. Note that [3] performs *axis-aligned* rotations. Best results are shown. **(B)** Among online planners, CEM clearly performs the best. All planners use 120 threads, RGBD images, and the cube pose corrector. We note that when ignoring the 80s timeout imposed in [3], [4], CEM achieves higher mean and median rotation counts than Dactyl and Dextreme. **(C)** Hardware ablations show that using depth images slightly improves rotation rate, but using the corrector and as many threads as possible substantially boosts performance.

digit rotations in hardware, CEM is able to achieve dozens of rotations. Moreover, the rate that it can rotate the cube is also nearly 50% faster than PS, suggesting that it can discover and/or execute contact-rich plans much more effectively.

CEM discovers nontrivial contact sequences. CEM finds contact-rich plans that leverage contacts with all parts of the hand. Many rotations are only feasible when the hand first partially rotates the cube using an initial contact sequence, then completes the rotation by gaiting the cube to a different set of contacts. For example, in Fig. 3D, the cube is first pushed forward and continually supported by the thumb, allowing the index and ring fingers to then swipe in opposite directions to rotate the cube. Similarly, in Fig. 3F, the thumb and ring finger first pinch and lift the entire cube, which allows the index finger to rotate it about the pinched axis before the cube is safely lowered back onto the palm.

Moreover, many discovered plans exploit contact modes that are traditionally challenging to find, like sliding on edges and corners. To execute these motions, the planner employs intuitive strategies like maximizing torque by levering the cube close to a corner or edge. Lastly, our conservative cost function also induced safeguarding behavior, where fingers preemptively blocked the cube from the palm's edges or carefully supported the cube during risky rotations.

The gradient-based iLQR planner is not viable. Corroborating recent work [6], [38], we find that the stiff dynamics of the cube reorientation task prevent gradient-based methods from effectively finding good plans most likely due to poor numerical conditioning, causing jerky, erratic behaviors that do not lead to coherent rotation sequences.

DROP performs comparably to RL. While it is challenging to compare our results to prior RL-based methods like Dactyl [3] or Dextreme [4] due to many factors distinguishing each setup, like hand morphology, cube size, physical properties, camera type, or vision model, Table IA/B shows that our simple CEM planner approaches the performance of RL-based rotation policies (outperforming them when ignoring timeouts) with similar dexterity (see Fig. 3).

C. Hardware Ablations

To understand the impact of key design choices in the DROP architecture, we conducted a series of single-variable ablations with the CEM planner on hardware (Table IC).

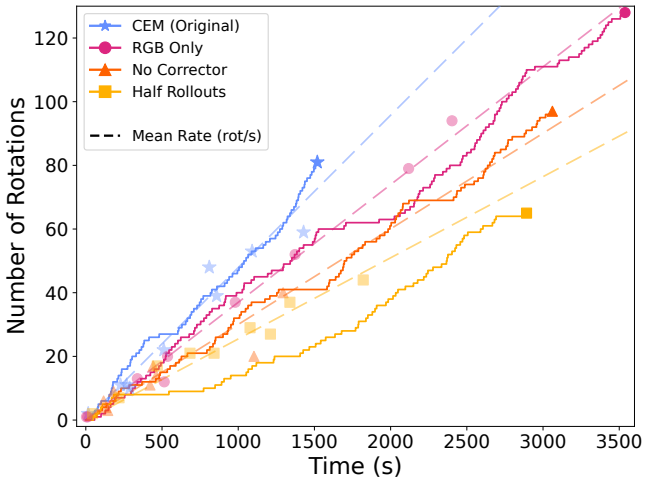

Fig. 4: **CEM ablation rotation rates.** We use rotation rate as a proxy for planner robustness, as slower rates correspond to "stuck" plans or repeatedly failed moves. Markers are all rotations vs. times for CEM and all ablations. The dashed lines show the mean rotation rates: it is clear that all ablations decrease the rate, which justifies our design of the DROP architecture. The solid lines show individual rotations for each method's longest streak. The long, flat regions correspond to the planner getting "stuck" in local minima.

Depth improves performance, but only marginally. We compared keypoint detection models trained on RGB versus RGBD images. While RGBD slightly improved rotation rates, it did not significantly outperform RGB, which still achieved 128 rotations, the longest sequence ever observed for DROP. Despite this noisy result, RGB's median rotations (28.5) were still slightly lower than RGBD's (30.5), and overall, the relative rotation rates suggest that depth provides a minor improvement in state estimation accuracy.

The corrector is key for performance. We tested DROP when ablating the corrector, passing raw smoother estimates $\tilde{q}$ directly to the planner. This significantly reduced performance, with mean rotations decreasing by 33%, median rotations by 60%, and rotation rate by 25%. Without the corrector, the planner often became trapped in local minima and long periods of inactivity (flat regions of rollouts in Fig. 4). This was caused by the exploitation of non-physical forces in rollouts arising from non-physical hand-cube interpenetration, leading to unrealistic predictions.

DROP is sensitive to the number of rollouts. We reduced the number of rollouts from 120 to 60 and elite samples from 4 to 3, resulting in a 20% decrease in mean rotation count

and an over 25% reduction in rotation rate. This highlights the importance of variance reduction via sufficiently-high sample quantity in SPC, and suggesting that improving search efficiency could significantly boost performance.

All ablations increased rotation rate variance. This consistent pattern demonstrates that depth measurements, the corrector, and sufficient rollout quantity all contribute significantly to the planner's reliability and consistency, which justifies the design of the DROP architecture.

D. Simulated Robustness Study

Lastly, we study DROP's robustness to model and estimation errors by conducting controlled trials in simulation, letting us isolate the planner from the estimation pipeline. We compared iLQR, PS, and CEM under various corrupted conditions, simulating system physics with a 2ms timestep while planner threads utilized a separate physics model with a 10ms timestep. Each configuration underwent five trials, ending upon cube drop, 80s timeout, or 150 rotations.

Specifically, we intentionally induce two types of errors that we believe contribute to real-world brittleness: (i) mistuning the planners' internal value of the hand K_p gains, and (ii) corrupting pose estimates from the simulation with a 0.1s lag and additive noise (simulated using a bounded random walk to mimic asymmetric state estimation error). When corrupting K_p, we study two cases: multiplying the true value by 1.25x and 1.5x, as the results were enlightening for comparing different planners. We also study the effect of the estimator and K_p errors together.

CEM is the most robust planner. Table II shows that CEM is the best planner under all error conditions. For example, while PS achieves a higher mean rotation count under perfect conditions than CEM, when K_p is mildly corrupted up to 1.25x, PS immediately achieves fewer mean rotations than CEM while suffering an over 2x decrease in rotation rate. At 1.5x, PS hardly rotates the cube at all, while CEM achieves 32.2 mean rotations, and even with the most aggressive errors, CEM was able to achieve dozens of rotations. CEM's superiority can be attributed to its strategy of recomputing π_θ from multiple rollouts, in contrast to PS's single-rollout approach. This strategy appears to be key for robustness, providing a plausible explanation for CEM's markedly better performance in hardware.

Rot/s is a good proxy for robustness. Based on empirical observation, we find that low rotation rates are typically caused by the failure to execute precisely-planned motions or the inability of the planner to escape local minima, resulting in the cube being "stuck," which can be attributed to model or estimation error. CEM's superior speed in these simulations supports our assessments of its robustness on hardware.

Error types have distinct failure modes. While incorrect K_p values primarily resulted in timeouts, corrupted estimates typically caused cube drops. This suggests that perfect state estimates allow for safe "caging" even with poor actuation models, but estimation errors during precise maneuvers often cause drops, as the planner underestimates rollout risk.

Planner	Change	Mean Rot/s ↑	Mean Rots	Drops	Timeouts
iLQR	None	N/A	0 ± 0	3	2
PS	None	0.130	122 ± 54.8	1	0
	K_p x1.25	0.060	44.4 ± 26.4	1	4
	K_p x1.5	0.042	1.2 ± 1.17	0	5
	Est.	0.032	3.2 ± 1.72	5	0
	Est., K_p x1.25	0.048	3.4 ± 3.14	5	0
	Est., K_p x1.5	0.015	0.2 ± 0.40	3	2
CEM	None	0.222	95.4 ± 34.8	0	4
	K_p x1.25	0.148	52.8 ± 26.0	0	5
	K_p x1.5	0.100	32.2 ± 26.6	0	5
	Est.	0.120	46.8 ± 24.7	5	0
	Est., K_p x1.25	0.083	73.8 ± 26.4	4	1
	Est., K_p x1.5	0.058	24.8 ± 13.3	2	3

TABLE II: **Simulated robustness tests.** We intentionally degrade planners to test their robustness by (i) tuning the hand proportional gain K_p too high, and (ii) corrupting the estimator with noise and lag (denoted "Est."). iLQR failed to achieve any rotations even with perfect information. We observe that PS degrades substantially more than CEM in the presence of model and estimation error, which suggests that CEM may transfer well to hardware.

V. Conclusion and Future Directions

This work presents DROP, a minimalist online planning method for in-hand manipulation via sampling-based predictive control that achieves robust cube rotations in hardware. While promising, there are many avenues for future research.

Better planners. While we found that vanilla CEM already achieved impressive results, many more sophisticated algorithms exist, such as CMA-ES [24], MPPI [23], etc.

Robustness. DROP often plans "risky" actions, possibly due to differences between real-world and simulated physics. Incorporating domain randomization or risk-sensitivity into search-based planners, like successful RL approaches, remains an open challenge, especially due to the extra computation required to simulate randomized physics online.

Object generality. As in prior RL-based works [3], [4], we first focus on robustly reorienting a single, simple object. While DROP can easily adapt to new objects in simulation, our current vision pipeline requires retraining for new objects. Recent advancements in general pixel-space tracking [27], [29] and video-based mask propagation [39] suggest more avenues for pose estimation that could generalize our search-based approach without extensive retraining.

Enhanced, data-driven search. As noted in our ablations, finding good plans via search demands many threads; indeed, our reported results rely on a server-grade CPU. Thus, improving efficiency is key for better performance. Promising directions include sampling from imitation-learned policies [40], learning value functions for rollout evaluation [41], and exploring alternate spline parameterizations [24] or action spaces [42]. Searching for high-level commands to provide to a lower-level RL policy could perhaps yield systems with both the flexibility of search and robustness of RL.

The simplicity of DROP opens many paths for advancing contact-rich manipulation. It is our hope that algorithms like DROP can generalize to more challenging real-world tasks than cube reorientation, permitting tool use, enhanced human-robot collaboration, and more.

REFERENCES

[1] Tatsuya Ishihara, Akio Namiki, Masatoshi Ishikawa, and Makoto Shimojo. Dynamic pen spinning using a high-speed multifingered hand with high-speed tactile sensor. In *2006 6th IEEE-RAS International Conference on Humanoid Robots*, pages 258–263, 2006.

[2] Nikhil Chavan-Dafle, Alberto Rodriguez, Robert Paolini, Bowei Tang, Siddhartha Srinivasa, Michael Erdmann, Matthew T. Mason, Ivan Lundberg, Harald Staab, and Thomas Fuhlbrigge. Extrinsic dexterity: In-hand manipulation with external forces. In *Proceedings of (ICRA) International Conference on Robotics and Automation*, pages 1578 – 1585, May 2014.

[3] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *CoRR*, abs/1808.00177, 2018.

[4] Ankur Handa, Arthur Allshire, Viktor Makoviychuk, Aleksei Petrenko, Ritvik Singh, Jingzhou Liu, Denys Makoviichuk, Karl Van Wyk, Alexander Zhurkevich, Balakumar Sundaralingam, Yashraj Narang, Jean-Francois Lafleche, Dieter Fox, and Gavriel State. Dextreme: Transfer of agile in-hand manipulation from simulation to reality, 2024.

[5] Tao Chen, Megha Tippur, Siyang Wu, Vikash Kumar, Edward Adelson, and Pulkit Agrawal. Visual dexterity: In-hand reorientation of novel and complex object shapes. *Science Robotics*, 8(84), November 2023.

[6] Taylor Howell, Nimrod Gileadi, Saran Tunyasuvunakool, Kevin Zakka, Tom Erez, and Yuval Tassa. Predictive sampling: Real-time behaviour synthesis with mujoco, 2022.

[7] Albert Li. Drop: Dexterous reorientation via online planning. `https://caltech-amber.github.io/drop/`, 2024.

[8] Zhao-Heng Yin, Binghao Huang, Yuzhe Qin, Qifeng Chen, and Xiaolong Wang. Rotating without seeing: Towards in-hand dexterity through touch. *Robotics: Science and Systems*, 2023.

[9] Lennart Röstel, Johannes Pitz, Leon Sievers, and Berthold Bäuml. Estimator-coupled reinforcement learning for robust purely tactile in-hand manipulation. In *2023 IEEE-RAS 22nd International Conference on Humanoid Robots (Humanoids)*, pages 1–8. IEEE, 2023.

[10] Tao Chen, Jie Xu, and Pulkit Agrawal. A system for general in-hand object re-orientation, 2021.

[11] Michael Posa, Cecilia Cantu, and Russ Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *The International Journal of Robotics Research*, 33:69 – 81, 2014.

[12] Neel Doshi, Kaushik Jayaram, Benjamin Goldberg, Zachary Manchester, Robert Wood, and Scott Kuindersma. Contact-implicit optimization of locomotion trajectories for a quadrupedal microrobot. In *Robotics: Science and Systems XIV*, RSS2018. Robotics: Science and Systems Foundation, June 2018.

[13] Simon Le Cleac'h, Taylor Howell, Shuo Yang, Chi-Yen Lee, John Zhang, Arun Bishop, Mac Schwager, and Zachary Manchester. Fast contact-implicit model-predictive control, 2023.

[14] Vince Kurtz, Alejandro Castro, Aykut Özgün Önol, and Hai Lin. Inverse dynamics trajectory optimization for contact-implicit model predictive control, 2023.

[15] Alp Aydinoglu and Michael Posa. Real-time multi-contact model predictive control via admm, 2022.

[16] William Yang and Michael Posa. Dynamic on-palm manipulation via controlled sliding. In *Proceedings of Robotics: Science and Systems*, July 2024.

[17] Wen Yang and Wanxin Jin. Contactsdf: Signed distance functions as multi-contact models for dexterous manipulation, 2024.

[18] Claire Chen, Preston Culbertson, Marion Lepert, Mac Schwager, and Jeannette Bohg. Trajectotree: Trajectory optimization meets tree search for planning multi-contact dexterous manipulation. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8262–8268, 2021.

[19] Tao Pang, H. J. Terry Suh, Lujie Yang, and Russ Tedrake. Global planning for contact-rich manipulation via local smoothing of quasi-dynamic contact models, 2023.

[20] Steven Lavalle and James Kuffner. Rapidly-exploring random trees: Progress and prospects. *Algorithmic and computational robotics: New directions*, 01 2000.

[21] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[22] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *2011 IEEE International Conference on Robotics and Automation*, pages 4569–4574, 2011.

[23] Grady Williams, Paul Drews, Brian Goldfain, James M. Rehg, and Evangelos A. Theodorou. Information-theoretic model predictive control: Theory and applications to autonomous driving. *IEEE Transactions on Robotics*, 34(6):1603–1622, 2018.

[24] Julius Jankowski, Lara Brudermüller, Nick Hawes, and Sylvain Calinon. Vp-sto: Via-point-based stochastic trajectory optimization for reactive robot behavior. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10125–10131, 2023.

[25] Corrado Pezzato, Chadi Salmi, Max Spahn, Elia Trevisan, Javier Alonso-Mora, and Carlos Hernandez Corbato. Sampling-based model predictive control leveraging parallelizable physics simulations, 2023.

[26] Mohak Bhardwaj, Balakumar Sundaralingam, Arsalan Mousavian, Nathan D. Ratliff, Dieter Fox, Fabio Ramos, and Byron Boots. STORM: An integrated framework for fast joint-space model-predictive control for reactive manipulation. In *5th Annual Conference on Robot Learning*, 2021.

[27] Carl Doersch, Pauline Luc, Yi Yang, Dilara Gokay, Skanda Koppula, Ankush Gupta, Joseph Heyward, Ignacio Rocco, Ross Goroshin, João Carreira, and Andrew Zisserman. Bootstap: Bootstrapped training for tracking-any-point, 2024.

[28] Nikita Karaev, Ignacio Rocco, Benjamin Graham, Natalia Neverova, Andrea Vedaldi, and Christian Rupprecht. CoTracker: It is better to track together. 2023.

[29] Yuxi Xiao, Qianqian Wang, Shangzhan Zhang, Nan Xue, Sida Peng, Yujun Shen, and Xiaowei Zhou. Spatialtracker: Tracking any 2d pixels in 3d space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[30] Bowen Wen, Wei Yang, Jan Kautz, and Stan Birchfield. FoundationPose: Unified 6d pose estimation and tracking of novel objects. In *CVPR*, 2024.

[31] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[32] Reuven Rubinstein. The cross-entropy method for combinatorial and continuous optimization, 1999.

[33] Klaus Greff, Francois Belletti, Lucas Beyer, Carl Doersch, Yilun Du, Daniel Duckworth, David J Fleet, Dan Gnanapragasam, Florian Golemo, Charles Herrmann, Thomas Kipf, Abhijit Kundu, Dmitry Lagun, Issam Laradji, Hsueh-Ti (Derek) Liu, Henning Meyer, Yishu Miao, Derek Nowrouzezahrai, Cengiz Oztireli, Etienne Pot, Noha Radwan, Daniel Rebain, Sara Sabour, Mehdi S. M. Sajjadi, Matan Sela, Vincent Sitzmann, Austin Stone, Deqing Sun, Suhani Vora, Ziyu Wang, Tianhao Wu, Kwang Moo Yi, Fangcheng Zhong, and Andrea Tagliasacchi. Kubric: a scalable dataset generator. 2022.

[34] Frank Dellaert and GTSAM Contributors. borglab/gtsam, May 2022.

[35] Jad Abou-Chakra, Krishan Rana, Feras Dayoub, and Niko Sünderhauf. Physically embodied gaussian splatting: A realtime correctable world model for robotics, 2024.

[36] Kenneth Shaw, Ananye Agarwal, and Deepak Pathak. Leap hand: Low-cost, efficient, and anthropomorphic hand for robot learning, 2023.

[37] Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *Proceedings of the First International Conference on Informatics in Control, Automation and Robotics - Volume 1: ICINCO,*, pages 222–229. INSTICC, SciTePress, 2004.

[38] H. J. Terry Suh, Max Simchowitz, Kaiqing Zhang, and Russ Tedrake. Do differentiable simulators give better policy gradients?, 2022.

[39] Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Vasudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross Girshick, Piotr Dollár, and Christoph Feichtenhofer. Sam 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714*, 2024.

[40] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 2024.

[41] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

[42] Roberto Martín-Martín, Michelle Lee, Rachel Gardner, Silvio Savarese, Jeannette Bohg, and Animesh Garg. Variable impedance control in end-effector space. an action space for reinforcement learning in contact rich tasks. In *Proceedings of the International Conference of Intelligent Robots and Systems (IROS)*, 2019.

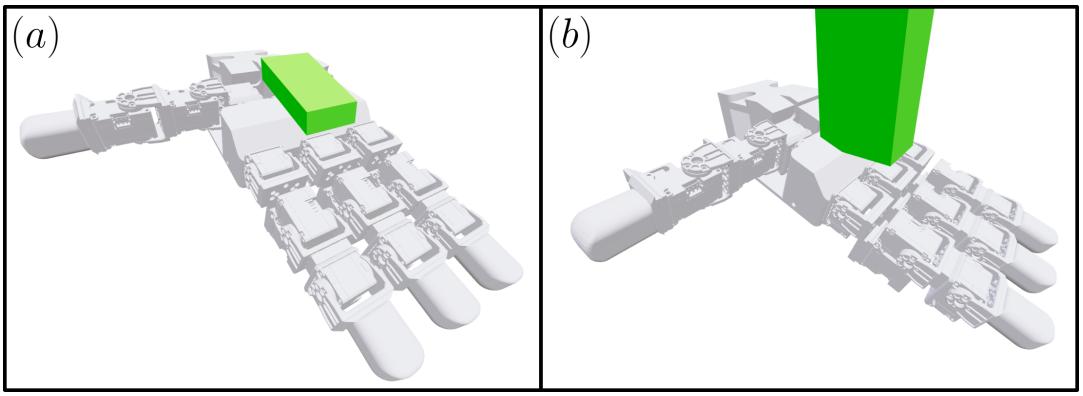

Fig. 5: The safe regions $\mathcal{S}$ (green) for (a) when the cube's xy coordinates lie in the palm, and (b) when the cube's xy coordinates lie outside the palm.

APPENDIX

This appendix aims to make precise some of the technical details that were omitted from the main body of the paper for brevity. For the most fine-grained details, please see the open-source code provided at https://caltech-amber.github.io/drop.

A. Sampling-based Predictive Control Implementation

Warm-starts by time shifting. Recall the generic SPC procedure described in Algorithm 1. Because the cost J at some time depends on the current state $x_0 = \hat{x}(t)$, once time elapses, the optimization problem (6) changes. Instead of solving the problem from scratch, the solution is *warm started* by applying a time shift to the previous solution (e.g., as in [43]). For the last time step of the warm-started solution, we simply copy the second-to-last spline parameter.

To be precise, let h denote an iteration of the mean control spline knots $\overline{U}$. Then, applying the shift operation gives

$$\begin{aligned} \overline{U}[h] &= [u_1[h], u_2[h], \dots, u_K[h]], \\ \text{shift}\left(\overline{U}[h]\right) &= [u_2[h], u_3[h], \dots, u_K[h], u_K[h]], \end{aligned}$$

which the new samples will be centered on (and similarly for any other parameters like sampling variances).

Retaining the best rollout in PS. In predictive sampling (PS), MJPC additionally keeps the current nominal (i.e., the best) control spline as one of the samples when resampling, so that if none of the sampled trajectories improves performance, the nominal policy is not updated. We found that this is critical to the performance of PS.

Minimum standard deviation in CEM. While the sampling variance is fixed in PS, in CEM, a diagonal covariance is repeatedly fit to the top M elite samples. However, in practice, the variance may quickly collapse to very low values, preventing the planner from exploring. Therefore, we specify a minimum standard deviation uniformly over the diagonal entries of Σ, denoted σ_{min}.

Default Parameters. Table III lists the values of planner parameters used in experiments unless otherwise stated.

Method	Parameter	Value
All	Spline Order	0
	Plan Horizon	1.0s
	Plan Timestep	0.01s
	Num Spline Knots	4
PS	Num Samples	120
	Sampling Stdev	0.3
CEM	Num Samples	120
	Num Elites	4
iLQR	Num Parallel Line Search Threads	120

TABLE III: Default planner parameter values used in all experiments.

B. Specifics of the DROP Control Problem

The position cost term. We now explain the exact form of the position cost term $\ell_{\text{p}}(p^c)$ in (8). The function $\text{dist}_{\mathcal{S}}(\cdot)$ is parameterized by the "safe region" $\mathcal{S}$ in which we would like the cube's center to remain. Specifically, we have

$$\begin{aligned} \text{dist}_{\mathcal{S}}(p^c) &:= d(\|p^c\|_{\mathcal{S}}), \\ d(x) &:= 0.05 \cdot \log\left(1 + \exp\left(\frac{250x}{0.05}\right)\right), \end{aligned} \tag{11}$$

where $\|x\|_{\mathcal{S}}$ reports the Euclidean distance between a point x and a set $\mathcal{S}$, and $d(x)$ is a scalar-valued function that applies a very quickly-growing penalty when $x > 0$. In particular, its parameters are chosen such that when $x = 0.01$, $d(x) \approx 1$.

The set $\mathcal{S}$ is defined by two cases. If the cube's x and y coordinates lie within a specified rectangle, then we parameterize $\mathcal{S}$ as a parallelepiped whose x and y faces are axis-aligned, and whose z faces are angled at the same angle as the palm of the hand, $\beta := 20°$. Otherwise, we instead specify a uniform minimum z height denoted z^- and no maximum height. The second case is designed so that if the planner predicts the cube will leave the palm, it prioritizes not dropping it and returning it back to the palm. See Fig. 5 for a visualization of these two cases.

The global origin of the system is located in the center of the interface between the hand mount and the bar of 80/20 to which it is affixed. The side length of the cube is $b := 0.07$. Let (p_x^c, p_y^c, p_z^c) denote the position coordinates of the center of the cube. The dimensions associated with the figure are specified in Table IV.

Cost Weights. The weights in (6) are $\lambda_{\text{g}} = 1.0, \lambda_{\text{p}} = 2.5$.

Time discretization. While (4) is expressed as an integral over the interval $[0, T]$, in practice we numerically compute

Dim	Value
x^-	0.08
x^+	0.14
y^-	-0.02
y^+	0.02
z^-	$\begin{cases} \frac{b}{2\cos(\beta)} - p_x^c \cdot \tan(\beta), & \text{if } p_x^c \in [x^-, x^+],\ p_y^c \in [y^-, y^+] \\ -0.015, & \text{otherwise} \end{cases}$
z^+	$\begin{cases} z^- + 0.035, & \text{if } p_x^c \in [x^-, x^+],\ p_y^c \in [y^-, y^+] \\ +\infty, & \text{otherwise} \end{cases}$

TABLE IV: Dimensions of the set $\mathcal{S}$.

it by discretizing the interval using a fixed time step Δt:

$$\int_0^T \ell\left(x(t), u(t)\right) dt \approx \sum_{j=0}^{H} \ell\Big(x(j\Delta t), u(j\Delta t)\Big) \cdot \Delta t, \quad (12)$$

where $H = T/\Delta t$.

C. Dataset and Training of the Keypoint Predictor

Additional dataset details. We now further explain design decisions for the dataset used to train the vision model g_φ.

As described in Sec. III-B, the training images were generated per the Movi-F dataset convention from Kubric. The images are sampled from frames of a video generated by simulating a random collection of objects (always including the cube) thrown into the scene from various positions and heights, with random camera motion and motion blur. The non-cube objects are drawn randomly from the Google Scanned Objects dataset, and backgrounds are taken randomly from PolyHaven.

These data were split into train and test datasets by first separating videos into train and test videos then by randomizing the order of all images from these videos. This ensured that very similar frames from the same video did not appear in both the train and test sets. Each video supplied 24 images. In these videos, the cube's size was normalized such that its side length was $2.0\mathrm{m}$. Thus, when using the keypoint detector model downstream in conjunction with the smoother, the cube size must first be normalized.

Visual/material properties of the cube were randomized using `kubric` by uniform random sampling (see Table V).

Property	Sampling Range
Roughness	[0.0, 0.3]
Specularity	[0.75, 1.0]
Metallic	[0.25, 0.75]
Index of Refraction	[1.0, 2.0]

TABLE V: Cube material randomizations.

The camera pose was sampled such that it was always pointed at the origin, but its position was uniformly randomly located in some hemispherical shell with inner radius $7.0\mathrm{m}$, outer radius $9.0\mathrm{m}$, and z height exceeding $0.2\mathrm{m}$.

As described in Sec. III-B, images in the dataset where too few or too many pixels corresponded to the cube were removed, as this indicated that the cube was either too occluded or too close to provide an accurate keypoint estimate. Specifically, an image was included in the dataset only if the proportion of cube pixels was in the interval $[0.02, 0.7]$.

Lastly, we note that while the size of the training dataset is fairly large for such a specific task (about 686000 images), it is far fewer than number of images used to train cube pose estimators in other works (e.g., 5 million in [4] and we approximate 76.8 million in [3]). However, it is clear that an improved different approach will need to be used in order to generalize to different objects.

Image Augmentation Details. One of the key factors for robust keypoint prediction in the real world was the application of extensive image augmentations during training. The augmentations were randomly sampled and applied during training using a mix of implementations from the open-source image processing package `kornia` and custom augmentations described below. The `kornia` augmentations and parameters are described in Table VI. For detailed descriptions of the parameters, please see the `kornia` documentation. Any time the parameter `p` appears, it denotes the probability that the augmentation is applied.

Augmentation	Parameters

`RandomAffine`	`degrees: 90`
	`translate: (0.1, 0.1)`
	`scale: (0.9, 1.5)`
	`shear: 0.1`

`RandomErasing1`	`p: 0.5`
	`scale: (0.02, 0.1)`
	`ratio: (2.0, 3.0)`
	`same_on_batch: False`

`RandomErasing2`	`p: 0.5`
	`scale: (0.02, 0.05)`
	`ratio: (0.8, 1.2)`
	`same_on_batch: False`

`RandomPlanckianJitter`	`mode: blackbody`

`ColorJiggle`	`brightness: 0.2`
	`contrast: 0.4`
	`saturation: 0.4`
	`hue: 0.025`

`RandomGaussianBlur`	`kernel_size: (5, 5)`
	`sigma: (3.0, 8.0)`
	`p: 0.5`

`RandomPlasmaShadow`	All defaults

TABLE VI: All RGB image `kornia` augmentations used during training. Augmentation parameters are defaults if unspecified.

We implemented custom augmentations that applied only to depth images, as well as a custom augmentation for transplanting random backgrounds onto training images. We describe these now and summarize the associated parameters in Table VII.

For the depth readings, we (i) uniformly randomly sample a per-image bias (`DepthBias`), (ii) add Gaussian noise (`DepthGaussianNoise`), and (iii) uniformly randomly sample near and far planes centered about some mean value, where everything too close or far is set to 0.0 (`DepthPlane`).

Lastly, we implemented a special augmentation called `CustomTransplantation` (not to be confused with the `RandomTransplantation` augmentation in `kornia`), which requires a batch of RGBD images as well as segmentation masks for the cube. With some probability, each

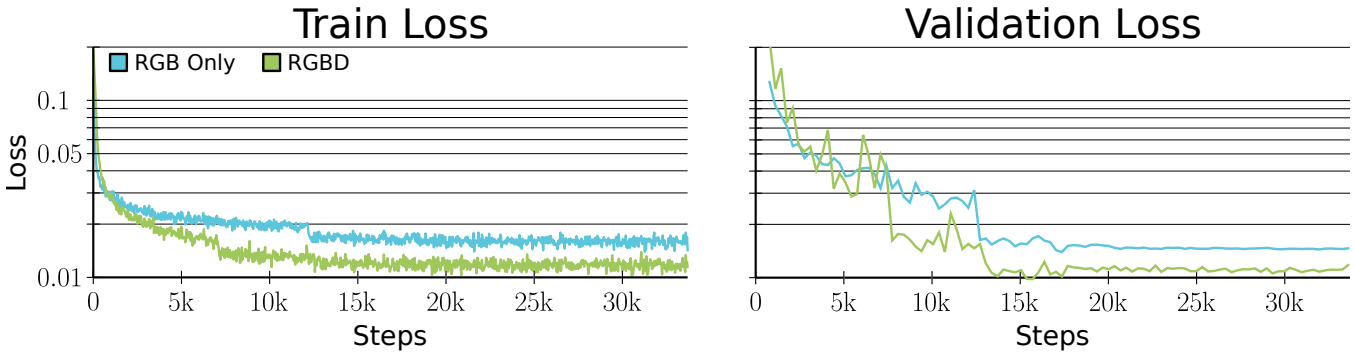

Fig. 6: Training and validation loss curves for the RGBD and RGB-only vision models. Note that the curves are log scale.

image in the batch (the acceptor) has all of its non-cube pixels replaced by pixels from another image in the batch (the donor) using the following procedure. First, all non-cube pixels in the acceptor are identified, which initializes a background mask. Second, all pixels in the donor that are closer to the camera than the corresponding pixels in the acceptor are added to the background mask. Third, the pixels in the donor mask corresponding to the cube in the donor are removed from the mask (to prevent there from being two cubes in a single image). Fourth, the background mask selects pixels from the donor and replaces the corresponding pixels in the acceptor. Finally, if the resulting transplanted image results yields an image that has too large or small of a ratio of visible cube pixels, the transplantation is not applied. This transplantation method allows backgrounds from donors to further obscure the cube in acceptors in random ways, which helps improve robustness.

Augmentation	Parameters
DepthBias	bias_range: (-0.02, 0.02) p: 0.5
DepthGaussianNoise	stdev: 0.005
DepthPlane	near_mean: 0.1 near_range: (-0.05, 0.05) p_near: 0.5 far_mean: 0.5 far_range: (-0.05, 0.05) p_far: 0.5
CustomTransplantation	p: 0.5 ratio_lb: 0.02 ratio_ub: 0.7

TABLE VII: Parameters for our custom augmentations.

Training Hyperparameters. Training hyperparameters are described in Table VIII. We trained the model using a machine with 8 H100 GPUs for about 8 hours, though we remark that using this many GPUs is not required.

Training and Validation Curves. We show the training and validation curves for the RGBD model as well as the RGB-only model used in the ablation study from Sec. IV-C in Fig. 6. Overall, the RGBD model seems to perform slightly better, but not by a large margin. Note that the figure is in log scale.

Hyperparameter	Value
batch_size	256
learning_rate (initial)	1e-3
epochs	100
num_workers	4
AMP	True
Loss Function	MSE
Optimizer	AdamW
Scheduler	ReduceLROnPlateau patience: 5 factor: 0.25 min_lr: 1e-6 grad_scaler: True

TABLE VIII: Training Hyperparameters.

D. Smoother Details

This section provides a cursory overview of factor graphs, explains the setup of the graph used in DROP's estimation problem, and describes implementation details and hyperparameters of the smoother. For the most fine-grained explanation, please see our open-source code: https://caltech-amber.github.io/drop.

A primer on factor graphs. Factor graphs are a type of probabilistic graphical model that models the relationships between unknown random variables via *factors*, which can be interpreted as (unnormalized) likelihood functions of subsets of all of the unknowns parameterized by measurements, which themselves are functions of the same unknowns. The factor graph defines a factorization of some global likelihood function over all unknowns, which gives us a computationally-convenient framework for performing *maximum a posteriori* (MAP) inference to recover an estimate of the unknowns. For further explanation, we refer the reader to [44].

Let X denote the total set of all unknowns and X_l denote some indexed subset of X. Then, we can represent the factors formally as

$$\phi(X) = \prod_l \phi_l(X_l), \tag{13}$$

which models the independence relationships between the unknown variables of interest. Now, let measurements be modeled $z_l = h_l(X_l)$, where h_l is some measurement model. Then, with a Gaussian noise model and the assumption that

the factors take the form

$$\phi_l(X_l) \propto \exp\left\{-\frac{1}{2}\left\|h_l(X_l) - z_l\right\|^2_{\Sigma_l}\right\}, \tag{14}$$

the MAP solution for the unknown variables is equivalent to the following nonlinear least squares problem:

$$X^{\text{MAP}} = \arg\min_X \sum_l \left\|h_l(X_l) - z_l\right\|^2_{\Sigma_l}. \tag{15}$$

In practice, the MAP inference problem is solved by numerical methods like the Gauss-Newton or Levenberg-Marquardt algorithms, which converge to some local minimum by solving successive linearizations of (15).

To reduce sensitivity to outlier measurements (i.e., keypoint detection failures), we used a *robust Huber error model* (see this GTSAM blog post for more information: gtsam.org/2019/09/20/robust-noise-model.html).

Incremental smoothing. We are interested in the setting where we receive a stream of incremental information and we seek to update the MAP estimate over time. In particular, we consider a canonical graphical model of a dynamical system with two types of factors: *dynamics* factors that model the (Markovian) temporal transitions between states ($\phi_t^{\text{dyn}}(X_{t+1}, X_t)$), and *measurement* factors that model the likelihood of observations on those states ($\phi_t^{\text{meas}}(X_t; z_t)$).

Because the graph would rapidly grow in size as time elapses, when incrementally smoothing, it is common to *marginalize* out the effect of older unknowns to retain their information content while removing them from the graph (see [44, Sec. 5.3]). This way, the number of variables remains fixed and the estimation problem stays tractable.

This scheme gives rise to *fixed-lag smoothing*, where some lookback window specifies the number of states in the past to maintain in the graph at each time step. The case where the lookback is 1 corresponds to an iterated extended Kalman filter (iEKF), since marginalization happens after each measurement is received, and the MAP problem is solved through successive linearizations (see [45, Sec 4.3.1]).

The cube pose estimation factor graph. In DROP, we seek to estimate the cube poses (and velocities) over time. For computational speed, we do not model the full dynamics of the cube-hand system (which gives rise to the need for the corrector later). In particular, the stiff contact dynamics would introduce substantial numerical challenges when conducting successive linearizations to solve (15).

Instead, we use a *constant velocity model* to approximate the cube dynamics. Because the linear and angular velocity of the cube is typically small when supported by the palm, this simple model (accompanied by a sufficiently tuned mistrust of it) provides a coarse but sufficiently-accurate model of the cube's motion.

For a given camera with a known global pose in conjunction with a pinhole camera model, the measurement model computes the expected keypoint locations in the camera's image frame (in pixel coordinates). The "true" measurements come from the keypoint predictor network g_φ.

Sketch of the keypoint projection factor. The keypoint factor (one for each keypoint, indexing suppressed for clarity) for a camera has the form

$$\phi_t^{\text{meas}}(q_t^c; \nu^{\text{cam}}, p_{\text{kp}}^{\text{pix}}, p_{\text{kp}}^c, p^{\text{cam}}, r^{\text{cam}}), \tag{16}$$

where q_t^c is the pose of the cube at time t, ν^{cam} is the camera's known intrinsics, $p_{\text{kp}}^{\text{pix}} \in \mathbb{R}^2$ is the measured keypoint locations in pixel coordinates from the predictor g_φ, $p_{\text{kp}}^c \in \mathbb{R}^3$ is the location of the associated keypoint in the cube's local frame, and $(p^{\text{cam}}, r^{\text{cam}}) \in SE(3)$ is the known fixed pose of the camera in the world frame.

Let the function proj : $\mathbb{R}^3 \to \mathbb{R}^2$ project a spatial point expressed in the camera frame to a coordinate in pixel space, which is implemented in GTSAM [34]. The measurement residual is then computed as follows:

$$p_{\text{kp}}^{\text{pix}} - \text{proj}(p_{\text{kp}}^c; \nu^{\text{cam}}, p^{\text{cam}}, r^{\text{cam}}), \tag{17}$$

where the Jacobians associated with frame transforms and the projection function are also computed by GTSAM. For the exact implementation details, please see our open-source implementation [7].

Smoother parameters and other details. We summarize all parameters in Table IX. All quaternions are reported in wxyz order. Noise parameters are given by standard deviations σ corresponding to diagonal components of the noise model covariance matrices. Rotational noise is represented by Gaussian noise in the tangent space. Huber regularization is applied identically to all noise models.

Parameter	Value
Cube Pos Prior	$p = [0.1, 0.0, 0.0]$ $r = [1.0, 0.0, 0.0, 0.0]$ $\sigma_p = [0.5, 0.5, 0.5]$ $\sigma_r = [1.0, 1.0, 1.0]$
Cube Vel Prior	$v = [0.0, 0.0, 0.0]$ $\omega = [0.0, 0.0, 0.0]$ $\sigma_v = [0.01, 0.01, 0.01]$ $\sigma_\omega = [0.2, 0.2, 0.2]$
Dynamics Pos Noise	$\sigma_p = [0.01, 0.01, 0.01]$ $\sigma_r = [0.2, 0.2, 0.2]$
Dynamics Vel Noise	$\sigma_v = [0.01, 0.01, 0.01]$ $\sigma_\omega = [0.2, 0.2, 0.2]$
Measurement Noise (in pixels)	$\sigma_z = [3.0, 3.0]$
Meas. Robustness Parameter k_{huber}	1.345
Lookback Window	1

TABLE IX: Smoother parameters.

E. Corrector Implementation

As describe in Sec. III-B, the corrector applies a virtual corrective wrench to its internal estimate of the cube state as shown in (10). The corrector gain matrices have 6 diagonal entries corresponding to the virtual forces and torques applied to the cube. We use the values

$$\begin{aligned} C_P &= \text{diag}([1000, 1000, 1000, 3, 3, 3]), \\ C_D &= \text{diag}([1, 1, 1, 0.001, 0.001, 0.001]). \end{aligned} \tag{18}$$

The extra gravitational corrective force described in Sec. III-B is simply an extra 10N added along the $-z$ axis of the virtual forces applied to the cube.

Additionally, there are a few parameters of the corrector dynamics $\hat{f}$ that differ from the planner's internal model f, which we show in Table X.

Parameter	Value
Timestep	0.04s
LEAP Hand k_p	3.0
`solimp`	[0.999, 0.999, 0.001, 0.0001, 1]
`solref`	[0.0001, 1]

TABLE X: Parameters of the corrector's internal model $\hat{f}$ that differ from the planner's internal model f.

F. Implementation of Noise Random Walk

In Sec. IV-D, we explained that when corrupting the perception stack, we simulate additive noise using a bounded random walk. The reason for this choice is that in real life, perception errors are highly-correlated with state. For example, if there is a 1cm error in cube keypoint estimates at some cube pose, then we typically observe that the error would remain at similar levels with low noise so long as the cube does not move much from that state. Thus, simply adding 0-mean uncorrelated noise with high variance is not an accurate simulated representation of what we observe in the real world.

Instead, by modeling the noise as a bounded random walk with very low noise, we can approximately model the effects of sustained pose estimation bias. Specifically, let $\nu \in \mathbb{R}^3, \xi \in \mathfrak{so}(3) \cong \mathbb{R}^3$ denote the position and rotational noises respectively. These noise values are iteratively updated as follows, where Δt represents whatever amount of time has elapsed since the last update:

$$\begin{aligned} \nu(t+\Delta t) &= \text{clip}\left(\nu(t) + z_\nu(t), \nu_{\text{lb}}, \nu_{\text{ub}}\right), \\ \xi(t+\Delta t) &= \text{clip}\left(\xi(t) + z_\xi(t), \xi_{\text{lb}}, \xi_{\text{ub}}\right), \end{aligned} \tag{19}$$

where $\nu_{\text{lb}}, \nu_{\text{ub}}, \xi_{\text{lb}}, \xi_{\text{ub}}$ are fixed lower and upper bounds for the random walk, $\text{clip}(\cdot)$ clips the values in the first argument elementwise to the provided lower and upper bounds, and $z_n u \sim \mathcal{N}(0, \Sigma_\nu), z_x i \sim \mathcal{N}(0, \Sigma_\xi)$ are randomly-sampled noises that drive the random walk.

Finally, we update the cube pose $q^c = [p^c, r^c] \in SE(3)$ as follows:

$$\begin{aligned} p^c(t+\Delta t) &= p^c(t) + \nu(t+\Delta t) \\ r^c(t+\Delta t) &= r^c(t) \circ \text{Exp}\left(\xi(t+\Delta t)\right). \end{aligned} \tag{20}$$

The parameters we used are summarized in Table XI.

Parameter	Value
ν_{lb}	[-0.01, -0.01, -0.01]
ν_{ub}	[0.01, 0.01, 0.01]
ξ_{lb}	[-0.1, -0.1, -0.1]
ξ_{ub}	[0.1, 0.1, 0.1]
Σ_ν	diag([0.001, 0.001, 0.001])
Σ_ξ	diag([0.0001, 0.0001, 0.0001])

TABLE XI: Parameters of the noise random walk.

G. Miscellaneous Hardware Implementation Details

While the stack described in Sec. III is fairly simple, there are a few miscellaneous details that affect performance.

LEAP hand gains. The scale of the gains in the simulated model and on hardware are quite different. While in simulation, the planner's LEAP hand gains are $k_p = 1.0$ and $k_v = 0.01$, on hardware, we use $k_p = 75$ and $k_v = 25$. In fact, the hardware gain values are already quite low compared to the values recommended in [36]. This was intentional; by lowering the gains, we lowered the amount of energy imparted on the cube, and allowed the controller to manipulate the cube with a higher degree of control.

The physical cube. The cube itself is made of low-density 3D-printed PLA, weighing 0.108kg. The faces of the cube are simply printed out on regular printer paper and taped onto the cube using reflective, low-friction packing tape. We remark that over time, the frictional properties of the cube may slightly change. Our reported experiments were run using a cube that was used for about 1 week prior, and anecdotally, the friction increased slightly over this period, which helped prevent the cube from slipping out of the hand. In the weeks prior to our final experimental trials, older versions of the cube also exhibited peeling tape, torn faces/edges, and other effects that substantially increased the friction of the cube. For the most repeatable results, we would recommend re-taping a cube from scratch every few weeks.

Manual calibration of camera poses. Before running experiments, we found it important to perform some manual adjustments of each camera's ground-truth pose with respect to the world frame. To perform the calibration, we streamed the estimated cube pose from the smoother and projected the associated analytical keypoint locations back onto each camera's image frame using the keypoint factors derived in App. D. By adjusting the camera position, we were able to manually align the projected keypoints with the corners of the cube.

Manual adjustments to the smoothed cube pose. We found that errors in the camera poses manifested as nearly-constant translational biases in the smoothed cube pose estimate. To remedy this, we simply manually adjusted estimated cube pose by adding an appropriate offset.

H. Acknowledgments

We thank Taylor Howell, Tom Erez, and Yuval Tassa for early discussions regarding sampling-based trajectory optimization for contact-rich tasks. We thank Taylor Howell for continual support on MJPC features that were key for the development of our work.

We thank Gavin Hua for helping to write low-level code for interfacing with the LEAP hand. We thank Zach Olkin for general assistance with setting up the control stack and implementation advice.

We thank Simon Le Cleac'h for insightful discussions and advice on contact-aware state estimation and the corrector.

We thank Sabera Talukder for assisting with the construction of cubes used in our hardware trials.

Lastly, we thank the creators and maintainers of all open source software that made this paper possible, including the following, which were not cited in the main body: `pytorch` [46] and `pybullet` [47].

REFERENCES

[43] Jacob Sacks, Rwik Rana, Kevin Huang, Alex Spitzer, Guanya Shi, and Byron Boots. Deep model predictive optimization, 2023.

[44] Frank Dellaert and Michael Kaess. *Factor Graphs for Robot Perception*, volume 6. Foundations and Trends in Robotics, 2017.

[45] Timothy D Barfoot. *State estimation for robotics*. Cambridge University Press, 2024.

[46] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[47] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2021.