
Article:
Xu, Z., Papallas, R., Modisett, J. et al. (2025) Tracking and Control of Multiple Objects During Nonprehensile Manipulation in Clutter. IEEE Transactions on Robotics, 41. 3929 - 3947. ISSN: 1552-3098

https://doi.org/10.1109/TRO.2025.3577437

Tracking and Control of Multiple Objects during Non-Prehensile Manipulation in Clutter

Zisong Xu[1], Rafael Papallas[1,2], Jaina Modisett[1], Markus Billeter[1], and Mehmet R. Dogar[1]

***Abstract*—This paper introduces a method for 6D pose tracking and control of multiple objects during non-prehensile manipulation by a robot. The tracking system estimates objects' poses by integrating physics predictions, derived from robotic joint state information, with visual inputs from an RGB-D camera. Specifically, the methodology is based on particle filtering, which fuses control information from the robot as an input for each particle movement and with real-time camera observations to track the pose of objects. Comparative analyses reveal that this physics-based approach substantially improves pose tracking accuracy over baseline methods that rely solely on visual data, particularly during manipulation in clutter, where occlusions are a frequent problem. The tracking system is integrated with a model predictive control approach which shows that the probabilistic nature of our tracking system can help robust manipulation planning and control of multiple objects in clutter, even under heavy occlusions. Associated code and data available at: https://github.com/ZisongXu/PBPF.**

***Index Terms*—Non-Prehensile Manipulation, 6D Pose Tracking, Particle Filtering, Multi-Object Tracking and Manipulation.**

I. Introduction

In this paper, we propose a method for tracking and control of objects during non-prehensile manipulation in clutter. We present an example scene in Fig. 1, where a robot is pushing an object (the yellow-green cylinder) to a goal area (highlighted as a red circle) among other cluttering objects. The controller uses estimated poses of the objects at each time-step, in a model-predictive control setting, to decide on the next robot controls. These object poses at each time-step are estimated by our tracking method.

Our tracking method combines the information coming from the robots joints (i.e., executed controls) with the information coming from an RGB-D camera, to estimate the poses of the objects. The past robot control information is used to perform a physics-based prediction of the objects' motion. We find that using such a physics-based approach, even if computationally expensive, improves the pose tracking performance significantly when compared with using only the camera image. The use of physics-based predictions also provides a crucial advantage: We can continue tracking the pose of the objects even when the camera is significantly obstructed from viewing certain objects (by other objects or by the robot hand). This is an important, and frequently occurring, problem during object manipulation, particularly in clutter. For example in Fig. 1, the top row shows the actual images from the viewpoint of the camera used. As can be seen, there is heavy occlusion of the cylinder, which is the most important object for the task. In the bottom row, the purple particles show our system's estimation of the possible poses of the objects. Even at the moments when the cylinder is heavily occluded and the camera provides little or no information about its pose, the purple cylinders provide a good guess of where the object is, which is used by our controller to push the object to the goal area. This is achieved, not only because the physics-based predictions tell us how the robot is moving the cylinder object, but also because the other visible objects provide physical constraints to the pose of the occluded object (i.e., good pose estimates of the visible objects indirectly improve the pose estimate of the occluded ones).

Traditionally, estimating the pose of an object from a single camera image has been the dominant approach in robotic manipulation [3]–[6]. This is partly due to prehensile (i.e., grasping-based, pick-and-place) manipulation being the most common mode investigated in the literature. In these systems, object pose is estimated once at the beginning, after which the robot makes a plan to reach and grasp the object. Since the object is not contacted (and therefore does not change its pose) until the moment of grasp, such a single pose estimate is usually sufficient. Object pose can be estimated from a camera image, e.g., using deep-learning-based methods [1], [2].

However, non-prehensile manipulation, such as pushing and toppling [7]–[13], requires tracking the object continuously as the object pose changes while it is being manipulated. Non-prehensile manipulation in clutter is useful, for example when a robot may need to reach into a fridge, or a supermarket shelf, to retrieve an object, or grasp an item from a box filled with objects in a warehouse, requiring the robot to push other objects aside. One way to perform tracking under non-prehensile manipulation is to keep using the same approach with prehensile manipulation (i.e., estimating the object's pose from camera images only). The majority of existing non-prehensile manipulation systems use this approach [8], [9], [11], [12]. In the bottom row of Fig. 1, the blue particles show the estimated poses of the objects using a camera-only system [1]. As expected, while the pose estimates are good when the objects are clearly visible to the camera, they degrade significantly when there are occlusions (e.g., the blue cylinders in the second and third snapshots). Fig. 2, top row, shows another example where this is more clear. The images show the estimated pose of a pushed box as an overlaid wireframe.

[1]School of Computer Science, University of Leeds, Leeds, LS2 9JT, UK. E-mail: {sc19zx, r.papallas, sc22jm, m.billeter, m.r.dogar}@leeds.ac.uk

[2]Department of Computer Science, American University of Beirut - Mediterraneo, Paphos, 8046, Cyprus. E-mail: rp00@aubmed.ac.cy

M. Dogar and R. Papallas were supported by the UK Engineering and Physical Sciences Research Council [EP/V052659/1 and EP/R031193/1].

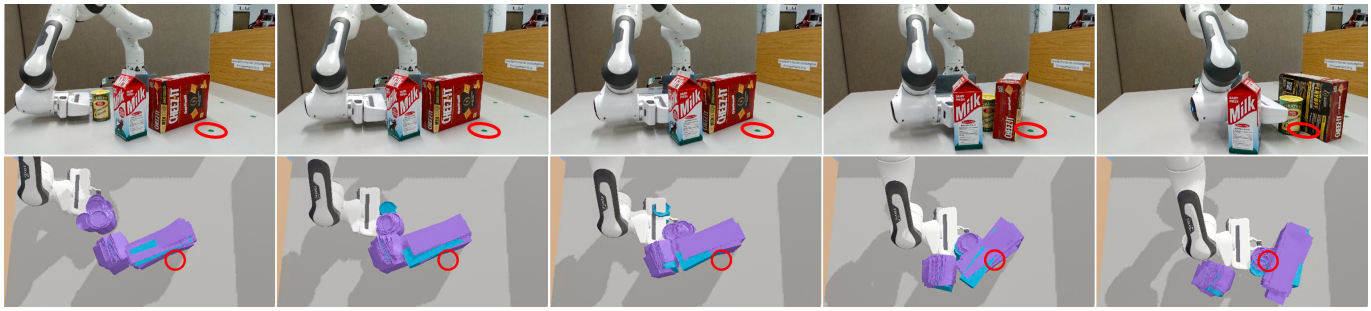


Fig. 1. An experimental scene where a robot is pushing an object (the yellow-green cylinder) to a goal area (red circle) among other cluttering objects. The top row shows the camera image that is used for tracking the objects. Purple particles in the bottom row show the possible poses of the objects as estimated by our method. The blue particles show the estimated pose from a camera-only pose estimation system, DOPE [1]. Please also see the attached videos.

The top row is from another camera-only pose tracking system [2], which loses the object once it becomes occluded.

However, during manipulation, we often have access to the controls executed by the robot. This provides information about how the objects might move from one time-step to the next. The bottom row in Fig. 2 shows the best guess (the particle nearest to the mean of the particle set) of our method, which is able to keep tracking the object using physics-based predictions. Therefore, in this paper, we propose to use robot control information, and to combine it with visual information, to track the objects' poses. We combine these two sources of information in a particle filtering framework [14]. We then use the estimates from the particle filter in a model-predictive-controller to perform goal-directed manipulation in clutter.

Our particle filter uses different observation models for RGB and depth. For depth, we render artificial depth images for each particle and compare them with the real depth image from the camera. For RGB, we find that rendering images is less realistic and informative. Therefore we use a learning-based RGB pose estimation system [1] to make instantaneous estimates of the objects' poses. We then fuse the RGB and depth information into a combined observation model.

Physics-based predictions come at the expense of computational cost, especially when using a particle-based approach. Still, we find that modern GPU and CPU parallelization schemes enable us to run our method at around 4 Hz. While this may not be fast enough for highly dynamic manipulation, we find that it is sufficient for many non-prehensile manipulation tasks. To achieve such rates, and to be able to run our tracking system on a single computer, we parallelize our depth-rendering method using modern GPU pipelines, while we perform the physics predictions using a physics-engine [15] that can be parallelized over multiple CPU cores.

During the development of our system, we realized that while there are benchmark datasets for object pose estimation [16], none are specific to pose estimation during non-prehensile manipulation, and none include information such as time-stamped robot joint states, which are required if one would like to develop and evaluate object pose estimation systems that make use of robot interaction information. There are datasets designed for robotic manipulation [17]–[20] including robot joint state or control information and RGB images from the workspace. However, they do not provide ground truth pose of the manipulated objects over time, making it impossible to evaluate the accuracy of tracking algorithms. We, therefore, release our dataset[1] which includes 50 non-prehensile interaction scenes with multiple YCB [21] and HOPE [22] objects, with each scene including 8000+ timesteps, totaling to 400000+ data points, including robot state information, ground truth object poses, RGB-D images, and timestamps at each data point, along with robot and object 3D models.

Our method has certain assumptions and limitations:

- We assume all objects in a given scene are "known", with a 3D model available. Our method currently cannot

[1]The dataset will be released with the publication of this manuscript.

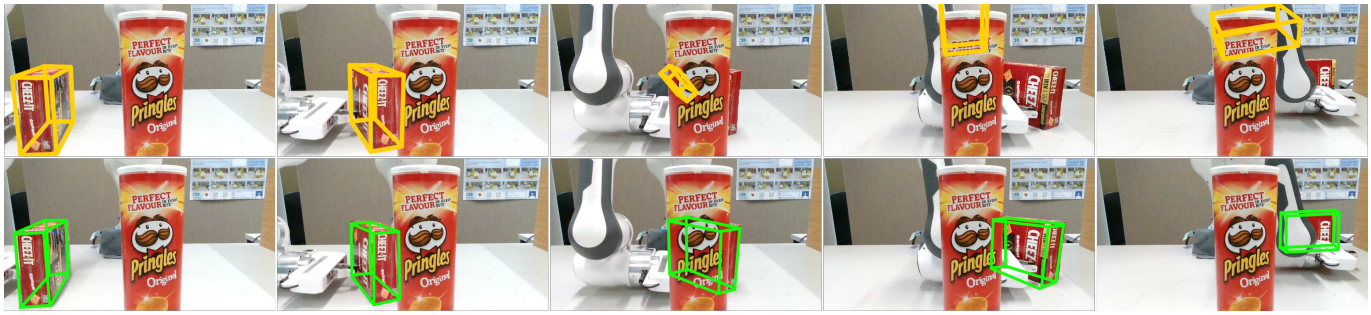


Fig. 2. The estimated pose of an object is overlaid as a wireframe onto the image, during a push by a robot. The Cheezit box is the target object to be tracked, and Pringles is only a visual obstacle, not part of the tracked objects. Top row shows the estimated pose by a state-of-the-art camera-only RGB-D pose estimation system, FoundationPose [2]. The bottom row shows the estimation of our system, which uses physics-based predictions in addition to the RGB-D images. Please also see the multimedia attachment for video versions.

handle unknown objects appearing in the scene.
- Our method assumes all objects are visible at the very beginning of the task (i.e., at $t = 0$).
- Our method assumes the robot is the only agent moving objects. For example, if a human enters the scene and moves the objects, our tracking can get lost.
- While our formulation and implementation is general in terms of the number of objects that the robot can simultaneously interact with, practically, our method has computational limits, due to the increased cost of physics simulation in scenes with more objects. In this paper, we test our method in scenes where the robot simultaneously interacts with up to five objects.

Since our method requires a physics model, the limitations of the particular physics model/engine used (e.g., if the physics model is less accurate for some type of interactions, but more accurate for others) is also carried over into our method. For example, as discussed in our results, we noticed that the physics engine we use is less accurate with high-impact collisions, which results in a degradation of tracking performance of our method in such scenarios.

This work evolved from our previous work [23]. The particular novelties in this version are listed below.

- The particle filtering formulation in our previous work was limited to a single object. Here, we generalize it to multiple objects. When compared to our conference paper, novel aspects of our generalization to multiple objects include (i) extending the particle state to multiple objects, (ii) factorization of the joint observation model into models for each object, (iii) computing visibility between different objects in the same particle. Moreover, our previous work was limited to using RGB images. Here, we also use depth images, which significantly improve our tracking performance. Our depth model, which treats the multi-object scene as a whole, is novel.
- We present a completely new set of new experiments including multiple physically interacting objects.
- The previous work was limited to tracking of object poses during fixed robot motion. Here, we develop and use a model-predictive-controller for goal-directed manipulation, with accompanying experiments.
- Since our method has changed significantly, all experiments in the current version are new. Additionally, we use a more varied and up-to-date set of baseline methods that we compare against.
- We release a new dataset, as described above.

The rest of this paper is structured as follow. Sec. II presents the related work, and Sec. III formulates the general problem. Sec. IV presents our particle-filtering based tracking approach. Sec. V presents the model-predictive-control approach. We present our main experiments and results in Sec. VI. In Sec. VII we perform further experiments to analyze our method's limitations and robustness to parameters. Sec. VIII concludes the paper.

II. RELATED WORK

Robotic manipulation systems often estimate object poses (either in the 2D image or in the 3D world) from a single camera snapshot, as opposed to tracking object poses over time. For example, at the Amazon Robotics Challenge [3], almost all teams used perception systems that estimate object poses (or grip/suction points) from a single snapshot taken before the robot contacts the objects. Similarly, Müller et al. [6], in their winning entry to the Robocup@Home competition, estimated poses of objects on tables from an initial camera image, before using motion planning and grasping pipelines. Sun et al. [5], in their review of robotic systems that took part in recent robotic object manipulation competitions, note that all systems treat perception as a "static" problem, where object poses are estimated without consideration of their dynamics/motion over time, which they identify as a limitation of existing systems. Most existing systems that specifically focus on robotic manipulation in cluttered, multi-object, settings also use a similar approach. Wang and Hauser [24] develop a system for dense packing of multiple objects, which uses static pose estimates from a camera. Shome et al. [25] also propose a system for tight packing of multiple objects, using a similar pose estimation approach [26].

The above systems can afford to estimate the objects' poses once, instead of continuously tracking the objects, because they perform prehensile (i.e., pick-and-place) manipulation. During non-prehensile manipulation, however, the objects' pose continuously change. Therefore, model-predictive-control approaches have been developed that continuously re-optimize/re-plan robot motion, based on the estimated pose of objects at each time-step during manipulation. Hogan and Rodriguez [9] propose such a model-predictive-controller for pushing of a single object, where the object is tracked using residual markers. Similarly, Zhou et al. [8] present an approach for non-prehensile pushing, where objects are tracked using markers. Other systems for pushing in clutter [11], [12] also adopt marker-based tracking. These reflect the challenges of estimating the poses of objects during non-prehensile manipulation in clutter, where not only the objects move and their poses need to be continuously estimated, but also objects and the robotic hand obstruct the view of the camera frequently. Such non-prehensile manipulation, particularly in multi-object settings, requires object pose tracking that can work under significant occlusions, even if the estimates are probabilistic.

There have been significant advances in camera-based RGB(-D) object pose estimation approaches, particularly with the use of neural networks [27], [28]. Tremblay et al. [1] proposed a system called Deep Object Pose Estimation (DOPE) that, given an RGB image, can predict the poses of objects in the image. DOPE requires a pre-training per object. Recently, this system has been improved as Diff-DOPE [29], which can use depth data to improve the DOPE pose estimates significantly. Similarly, Xiang et al. [30] developed a system based on convolutional neural networks that can output the pose of an object in a given image.

While many of the systems above work in a *single-snapshot* manner (i.e., the estimated object pose depends only on the current input image, and not on any input or images from previous timesteps), there are also methods that take a *tracking* approach. These methods track the object pose over time and use frames from previous time steps to refine the

pose of subsequent ones [31]–[33], including FoundationPose [2], which can also track novel objects. Several vision-based tracking systems employ a particle filtering approach similar to ours [34]–[38]. These systems vary in their assumptions about object and camera motion; some presume the object remains stationary [39] while allowing for camera movement [36], and make *constant-velocity* assumptions about motion between consecutive frames. Stoiber et al. [40] proposed a method that uses a multi-modality approach fusing visual appearance and geometric data to track objects. Keypoint features capture local texture details, while a region-based model segments object surfaces for enhanced tracking. However, these systems all use RGB(-D) images only, which, when there are significant occlusions of the tracked object, naturally fail to track it. In our work, we propose to also use the robot control information to make physics-based predictions on how objects move, and merge it with the information from a camera.

There have been other works that combine physics-based models with object pose estimation. Mitash et al. [41] combined visual data with physics-based reasoning for 6D pose estimation in environments with multiple objects. This approach, however, is specifically tailored to static scenes, focusing on scenarios where the physical stability of a pile of objects' pose can be estimated. Schmidt et al. [42] present a depth-based tracking method that also identifies the stable pose of a tracked object given its contacts with the fingers of a robot. Another work [43] focuses on tracking the 6D pose of an object, however focusing on the specific case of when the object makes an impact collision with a surface (e.g. a box being thrown onto a surface). Kandukuri et al. [44] also perform physics-based tracking of an object from RGB-D video. However, the object's movement is solely determined by an initial momentum imparted manually. In contrast, our method incorporates a motion model that integrates robotic motion information continuously, enabling the non-prehensile manipulation of objects by the robot, including collisions and interactions between objects. A learning-based physics prediction for object tracking method, as proposed by T. Mörwald et al. [45], uses random pushes to learn the relationship between applied forces and object motion. Compared to physics engines, it can offer faster computation. However, it requires offline training for different objects, interactions, and environments. For example, the method is presented for a single object only. To make it work in scenes with interactions between multiple objects, one would need to train the system for each combination of different objects. Additionally each interaction mode (e.g., pushing, poking, toppling), and environment (e.g., static obstacles that change how objects move) would require separate training. In contrast, physics engines offer greater flexibility without the need for retraining.

There are also other works that use robot joint state information as constraints on object pose estimation. Martin et al. [46] combine contact-constraints at multiple fingers with a camera image to estimate the pose of an object grasped in the hand. Wuthrich et al. [47] present a method where the object is visually tracked while it is moved by a robot, and the method integrates vision with robot controls, similar to ours. However, this method assumes prehensile manipulation (i.e. object is assumed to be rigidly grasped by the robot), and therefore the object motion prediction is a kinematics-only problem, and does not require physics-based reasoning.

Several works have used Kalman filter-based algorithms for tracking with a variety of motion models such as constant velocity/acceleration or random walk [48]–[51]. However, mismatches between the robot's control inputs and the assumed motion model, such as the non-smooth trajectories generated when the robot pushes a target object, can lead to cumulative prediction errors. In contrast, our method uses robot controls to predict the real motion, instead of imposing a specific but uninformed motion model, such as constant velocity/acceleration.

To the best of our knowledge, ours is the first work that integrates physics-based predictions with camera images to perform robust multi-object tracking during non-prehensile manipulation.

III. PROBLEM FORMULATION

We are interested in non-prehensile manipulation in clutter, where a robot manipulates multiple objects simultaneously. To successfully perform closed-loop control of the objects' poses, the robot requires robust 6D tracking of objects at all times.

We assume access to the sequence of joint-state inputs, u_t, (e.g., in this work, the joint states of a 7-DoF manipulator), and visual observations (z_t) from a camera, which can be RGB images (${}^{RGB}z$) and/or depth images (${}^{D}z$). We assume the camera's pose with respect to the robot is known. We also assume 3D CAD models of the objects, the robot, and the environment (e.g., the table, and static obstacles) are given.

The problem is defined as follows: at any given time-step t, leveraging all prior control inputs $u_0, u_1, \ldots, u_t$ and visual observations $z_0, z_1, \ldots, z_t$, estimate the objects' current poses, denoted as $\{q_t^i\}_{i=1,\ldots,n} \in SE(3)^n$, where n is the number of objects that the robot is interacting with/we want to track, and q_t^i represents the pose of object i at time step t. Since inferring the exact poses of the objects is impossible (especially when they are occluded), we estimate a probability distribution over the object poses, instead of a single pose. Specifically, the problem is to estimate/track and continuously refine the probability distribution of the objects' poses $p(\{q_t^i\}_{i=1,\ldots,n} | z_0, z_1, ..., z_t, u_0, u_1, ..., u_t)$, even when objects are partially or fully occluded. This conditional probability is commonly referred to as the *belief state*, which effectively integrates all prior information to predict the current poses of the objects. We describe a solution to the above tracking problem in Section IV.

With robust object tracking, we can use the object pose estimates as feedback for closed-loop simultaneous control of multiple objects. Methods like Model Predictive Control (MPC) depend on such continuous and reliable feedback. The tracking solution we describe integrates well with an MPC framework. We present such an MPC approach in Section V.

IV. PHYSICS-BASED PARTICLE FILTERING (PBPF)

Our method adopts a Bayesian filtering approach, particularly *particle filtering* [14]. Fig. 3 shows the overall process of our algorithm.

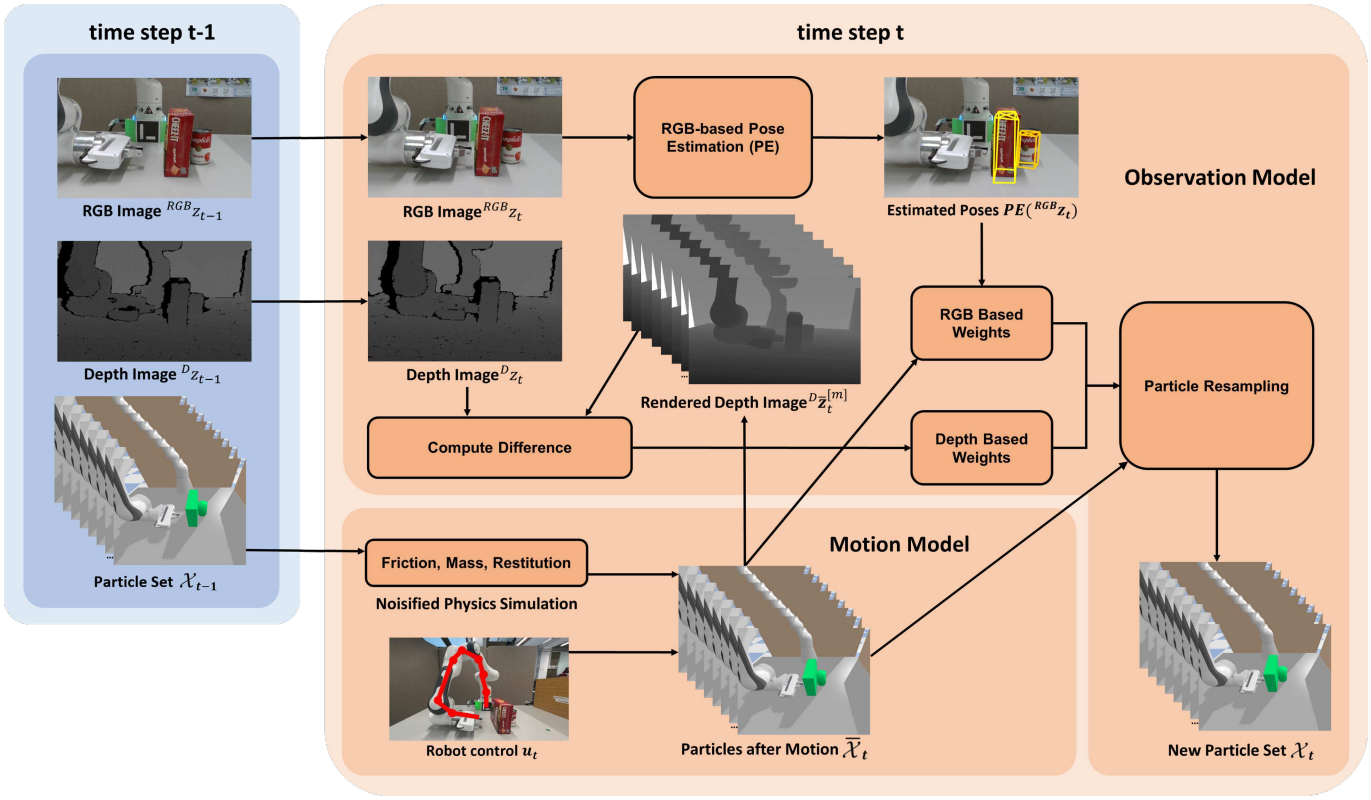


Fig. 3. Our system for objects' pose estimation during robotic non-prehensile manipulation over two time steps, $t-1$ and t. At $t-1$, RGB and depth images describe the scene, and particles represent possible objects' poses. As the system advances to t, these particles are updated by robot control, u_t, through physics simulation. New RGB and depth images are obtained from the camera. The RGB image is used to estimate objects' poses $PE({}^{RGB}z_t)$, and are compared with poses of particles $\bar{x}_t^{[m]}$ to get the difference. The depth image is compared with the rendered depth images (according to the poses of objects in each particle in the simulation). Finally, particle resampling refines the objects poses estimations, and then gets a new particle set.

Particle filtering represents the belief state at any given time t with a set of *particles*:

$$\mathcal{X}_t := x_t^{[1]}, x_t^{[2]}, ..., x_t^{[M]} \tag{1}$$

where each particle $x_t^{[m]} := \{q_t^i\}_{i=1,...,n}^{[m]}$ $(1 \leq m \leq M)$ is an instantiation of the pose state at time t and includes the poses of the objects being tracked, where M is the number of particles. In this paper, we also use the notation $q_t^{i,[m]}$ to refer to the pose of a particular object i in particle $[m]$ at time t. We emphasize that, in our formulation, each particle contains *all* the tracked objects' poses (not individual separate objects), which enables the use of the physics-based interactions/constraints between objects.

The process of particle filtering involves a dual-stage update at each time step t: first, the *motion update* stage (discussed in Sec. IV-A), where particles are moved based on the latest robot control inputs u_t, followed by the *observation update* stage (discussed in Sec. IV-B), where the particles are updated by the observational data z_t.

A. Motion Model

In the first stage of the particle filtering process, the propagation of each particle is addressed. Specifically, for a particle $x_{t-1}^{[m]}$ derived in the previous time step, an intermediate pose state $\bar{x}_t^{[m]} := \{\bar{q}_t^i\}_{i=1,...,n}^{[m]}$ is formulated for the current time step, by the following probabilistic motion model:

$$\bar{x}_t^{[m]} \sim p(x_t \mid x_{t-1}^{[m]}, u_t) \tag{2}$$

where $p(x_t \mid x_{t-1}^{[m]}, u_t)$ represents the conditional probability distribution. This distribution characterizes the evolution of the system state from $x_{t-1}^{[m]}$ to x_t, influenced by the robot control u_t. Direct analytical derivation of this motion model is often impractical. We use a physics engine to approximate this distribution. The physics engine is modelled as f:

$$x_t = f_\theta(x_{t-1}, u_t) \tag{3}$$

which includes a model of the robot, the environment, and objects, and predicts the resulting poses of the objects x_t given their poses in the previous time step, x_{t-1}, and robot control, u_t, by simulating the robot motion inside the engine and finding the resulting motion of the objects. Here, θ represents physics parameters that impact the motion in the physics engine. Examples of such parameters include the friction coefficient at the contacts, contact restitutions, and the mass of the objects.

The physics engine is deterministic (i.e. outputs the same resulting state, if given the same inputs and parameters). It therefore cannot directly be used instead of the probabilistic motion model in Eq. 2. However, we note that our uncertainty about the object's motion is due to (i) our uncertainty about the

exact physics parameters, θ; and (ii) the discrepancy between the physics engine and the real-world physics.

To address point (i), we model these parameters for every object in each particle as random variables sampled from a known normal distribution. For example, for the mass of object i in particle $[m]$ at time t, we sample:

$$\text{mass}_t^{i,[m]} \sim \mathcal{N}(\mu_{\text{mass}_i}, \sigma^2_{\text{mass}_i}) \tag{4}$$

Similarly, we sample a value for each physics parameter of each object in the particle, and $\theta_t^{[m]}$ represents the collection of these sampled values.

We then run the physics engine for that particle using the sampled parameters:

$$\bar{x}_t^{[m]} = f_{\theta_t^{[m]}}(x_{t-1}^{[m]}, u_t) + \epsilon \tag{5}$$

with the addition of a small Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma_f^2)$ to address (ii) above. We assume that $f_{\theta_t^{[m]}}(x_{t-1}^{[m]}, u_t)$, i.e., the output of the physics engine, is always physically feasible, given a physically feasible input. When we add noise ϵ, if it puts the state into a physically impossible configuration (i.e., penetration between bodies), we simply draw a new ϵ value. If we cannot draw an ϵ that does not result in penetration, then we set ϵ to zero for that particular timestep and particle.

In our implementation, we instantiate one physics engine *per particle* and use them to perform the motion update. Since each particle is independent of each other, these updates are parallelizable, which we exploit to enhance computational efficiency. As such, the set of intermediate particles, $\bar{x}_t^{[m]}$ $(1 \leq m \leq M)$, are computed.

In this work, we limit the physics parameters, θ, to the coefficients of friction, the restitution parameters, and the objects' mass[2]. However, uncertainty about other parameters (e.g. object shape, robot hand shape, and ground imperfections) can also be represented and integrated into this framework. In the above, $\mathcal{N}(\mu, \sigma^2)$ represents a normal (Gaussian) distribution[3]. The parameter μ_{mass_i} represents our best guess about the mass of object i, and $\sigma^2_{\text{mass}_i}$ represents our uncertainty (variance). These mean and variance values for mass, friction, and restitution can be estimated offline beforehand for each object type. The parameter σ_f^2 represents our estimated discrepancy between the physics engine and real-world physics. In practice, σ_f needs to be estimated *per physics model*, f, for example by calibrating it using a dataset with ground truth object poses (such as the one we release), and σ_f can then be fixed. The mass and friction mean and variance values need to be provided per object. In this work, we use our best guess about these object values, but other methods can also be used, such as using a vision-based neural network to predict physics properties, or extending the particle filter to also estimate these values.

[2]We assume the inertia tensor of an object to be diagonal and the object to have uniform density within its given 3-D model.

[3]Later in Eq. 12, we will also use the notation $\mathcal{N}(x; \mu, \sigma^2)$, which corresponds to the probability density at x, for mean μ and variance σ^2.

B. Observation Model

During the second stage of the particle filtering process, for each intermediate particle $\bar{x}_t^{[m]}$, we calculate an *importance factor* $w_t^{[m]}$ using the observation z_t:

$$w_t^{[m]} = p(z_t \,|\, \bar{x}_t^{[m]}) \tag{6}$$

Upon calculating the weights for all intermediate particles, these weights are used to *re-sample* a new set of particles $\mathcal{X}_t$, thus concluding the particle filter update. During re-sampling, each intermediate particle $\bar{x}_t^{[m]}$ may be selected (potentially multiple times) for inclusion in the new set $\mathcal{X}_t$, with selection probabilities proportional to $w_t^{[m]}$.

To calculate the weights in Eq. 6 we use the RGB and depth images as observations:

$$w_t^{[m]} = p({}^{RGB}z_t,\ {}^{D}z_t \,|\, \bar{x}_t^{[m]}) \tag{7}$$

We make the simplifying assumption that the RGB and depth images captured by the camera are independent. Consequently, the weight for a particle can be calculated as the product of the conditional probabilities of observing specific RGB and Depth images given the particle $\bar{x}_t^{[m]}$:

$$w_t^{[m]} = p({}^{RGB}z_t \,|\, \bar{x}_t^{[m]}) \cdot p({}^{D}z_t \,|\, \bar{x}_t^{[m]}) \tag{8}$$

C. RGB Images

The expression $p({}^{RGB}z_t \,|\, \bar{x}_t^{[m]})$ in Eq. 8 ideally represents the probability of making the current observation (i.e. getting the current camera RGB image) if objects were at pose $\bar{x}_t^{[m]}$. Since we do not have direct access to such a model, using the Bayes Theorem, we first re-write the observation model:

$$p({}^{RGB}z_t|\bar{x}_t^{[m]}) = \frac{p(\bar{x}_t^{[m]}|{}^{RGB}z_t)\, p({}^{RGB}z_t)}{p(\bar{x}_t^{[m]})} \tag{9}$$

Here, we note that $p({}^{RGB}z_t)$ is the same for every particle since the current observation does not change between particles. Furthermore, we make a simplifying assumption that $p(\bar{x}_t^{[m]})$ are also similar for different particles. This enables us to compute the importance factor using:

$$p({}^{RGB}z_t|\bar{x}_t^{[m]}) \approx p(\bar{x}_t^{[m]}|{}^{RGB}z_t) \tag{10}$$

Given that each particle contains the pose state of all the objects $\bar{x}_t^{[m]} := \{\bar{q}_t^i\}_{i=1,\ldots,n}^{[m]}$, and assuming that the pose of each object is independent of each other, we write:

$$p(\bar{x}_t^{[m]}|{}^{RGB}z_t) = \prod_{i=1}^{n} p(\bar{q}_t^{i,[m]}|{}^{RGB}z_t) \tag{11}$$

Since our particles, $\bar{x}_t^{[m]}$, are feasible (e.g., non-penetrating) outputs of the motion model, none of the simplifications above result in assigning probabilities to infeasible states.

We compute $p(\bar{q}_t^{i,[m]}|{}^{RGB}z_t)$ for each object, based on the results of two computations: a *Distance comparison* and a *Visibility score*.

Distance Comparison: We first use an off-the-shelf RGB-based single-snapshot pose estimation (PE) system to predict the pose of each object according to ${}^{RGB}z_t$ and then use the

distance of $\bar{q}_t^{i,[m]}$ to the predicted object pose to compute a probability value. Using $PE^i({}^{RGB}z_t)$ to represent the output of the pose estimation system for object i, i.e., the predicted pose of the object i given the camera image ${}^{RGB}z_t$, we compute a probability, p_{PE}, for each object:

$$p_{PE}(\bar{q}_t^{i,[m]}|{}^{RGB}z_t) = \mathcal{N}(\bar{q}_t^{i,[m]};\, PE^i({}^{RGB}z_t), \sigma^2_{PE^i}) \quad (12)$$

where the parameter $\sigma^2_{PE^i}$ represents the variance of PE system errors for the object. The variance can be estimated beforehand by collecting pose estimates for each object and comparing them to the ground truth pose. In practice, we use two normal distributions to compute the probability in Eq. 12. One distribution accounts for the positional Euclidean distance between $\bar{q}_t^{i,[m]}$ and $PE^i({}^{RGB}z_t)$. The second distribution represents the rotational distance between $\bar{q}_t^{i,[m]}$ and $PE^i({}^{RGB}z_t)$, computed as the minimum rotation around a single axis required to align the two orientations. The final probability in Eq. 12 is obtained by multiplying the probabilities from these two normal distributions. We use Normal distribution as the most generic model. Alternatively, given a particular PE system, the distribution for that system can be empirically modeled (either as a different parametric distribution or as a statistical/learned model) for a given object.

While $p_{PE}(\bar{q}_t^{i,[m]}|{}^{RGB}z_t)$ can be used as the value of $p(\bar{q}_t^{i,[m]}|{}^{RGB}z_t)$ in Eq. 11, this requires that an output from the PE system for each object, $PE^i({}^{RGB}z_t)$, is present. However, when the object is significantly occluded, a PE system may fail to output any pose estimate at all, i.e., $PE^i({}^{RGB}z_t)$ does not exist. When this happens, in this paper, we say that the PE system failed to "detect" the object. (Conversely, if the PE system outputs an estimate, then we say that the PE system "detected" the object.) To address these situations which occur when the object is occluded, we evaluate the visibility of an object in a particle.

Visibility Score: Note that, given an image ${}^{RGB}z_t$, the absence of object detection is also useful information. It often indicates that the object is either partially or completely occluded, which can provide significant information about its pose. To leverage this information, we use rendered artificial images of each particle from the perspective of the camera, which we call a "simulated camera" below. We then establish the following four conditions:

1) If the object i at pose $\bar{q}_t^{i,[m]}$ in the particle $\bar{x}_t^{[m]}$ is visible to the simulated camera (i.e. camera's view to the object is not occluded by other objects in $\bar{x}_t^{[m]}$, the robot, or the environment), and the PE system detects it in the actual image (i.e., $PE^i({}^{RGB}z_t)$ exists) then $p(\bar{q}_t^{i,[m]}|{}^{RGB}z_t)$ is equal to the probability estimated using the PE system, $p_{PE}(\bar{q}_t^{i,[m]}|{}^{RGB}z_t)$.
2) If the object i at pose $\bar{q}_t^{i,[m]}$ in the particle $\bar{x}_t^{[m]}$ is visible to the simulated camera, but the PE system does not detect it in the actual image (i.e., $PE^i({}^{RGB}z_t)$ does not exist) then $p(\bar{q}_t^{i,[m]}|{}^{RGB}z_t)$ is given a low probability value, $\alpha_l \in [0, 1]$, which is a predefined parameter. It represents the probability that the PE system cannot detect a visible (non-occluded) object.
3) If the object i at pose $\bar{q}_t^{i,[m]}$ in the particle $\bar{x}_t^{[m]}$ is not visible to the simulated camera (i.e. camera's view to the object is mostly occluded by other objects in $\bar{x}_t^{[m]}$, the robot, or the environment), but the PE system detects it in the actual image (i.e., $PE^i({}^{RGB}z_t)$ exists) then $p(\bar{q}_t^{i,[m]}|{}^{RGB}z_t)$ is given a low value, $p_{PE}(\bar{q}_t^{i,[m]}|{}^{RGB}z_t) \cdot \alpha_w$, where α_w ranges between 0 and 1. In rare cases, even mostly occluded objects can be detected by PE systems, and this probability reflects that case.
4) If the object i at pose $\bar{q}_t^{i,[m]}$ in the particle $\bar{x}_t^{[m]}$ is not visible to the simulated camera, and the PE system also does not detect it in the actual image (i.e., $PE^i({}^{RGB}z_t)$ does not exist(then $p(\bar{q}_t^{i,[m]}|{}^{RGB}z_t)$ is given a high probability value $\alpha_h \in [0, 1]$. This represents the probability that the PE system does not detect an object when it is mostly occluded, which is high.

Based on the conditions outlined above, the value of each $p(\bar{q}_t^{i,[m]}|{}^{RGB}z_t)$ in Eq. 11 is determined.

The method we presented above requires deciding whether an object is visible or occluded in a particle. We determine this by computing a visibility score for each object in each particle $Visible^i(x_t^{[m]})$, which represents the proportion of the part of the object i in particle $[m]$ that is not occluded.

To compute $Visible^i(x_t^{[m]})$ our system uses the simulated camera to render one segmented image with all objects in the particle, as well as individual segmented images for each object separately. We show an example in Fig. 4 for a given particle. The top-left image shows the rendered segmented image for all objects in the particle (including the robot and the environment). We count the number of pixels belonging to each object in this image, as shown top-right in Fig. 4. This count reflects the portion/footprint of the object that is not occluded and thus visible to the camera. The bottom three rows on the left show the individual segmented images rendered for each object separately. We also count the number of pixels belonging to the objects in these images, as shown in the bottom three rows in the middle. These counts reflect the total possible footprint of each object at that pose. We define the visibility score of each object as $Visible^i(x_t^{[m]})$ as the ratio of the unoccluded footprint to the total possible footprint, as shown in the bottom three rows on the right of Fig. 4. A visibility score of 1 indicates perfect visibility, where the entire object is visible to the camera. Conversely, a score of 0 indicates that the object is entirely occluded. To simulate the camera, i.e., to render the segmented images above, and to determine the pixel counts, we use GPU hardware queries (Appendix A).

We decide whether an object in a particle is visible to the camera, by comparing $Visible^i(x_t^{[m]})$ to a threshold, V, which is a predefined parameter of our system. If $Visible^i(x_t^{[m]})$ is higher than a V, then we consider the object in the particle to be visible, in the four conditions listed above. If it is lower, we consider it occluded in the particle. We determine V for each object type beforehand, by increasingly occluding it with an obstacle and identifying the degree of occlusion where the PE system starts to fail.

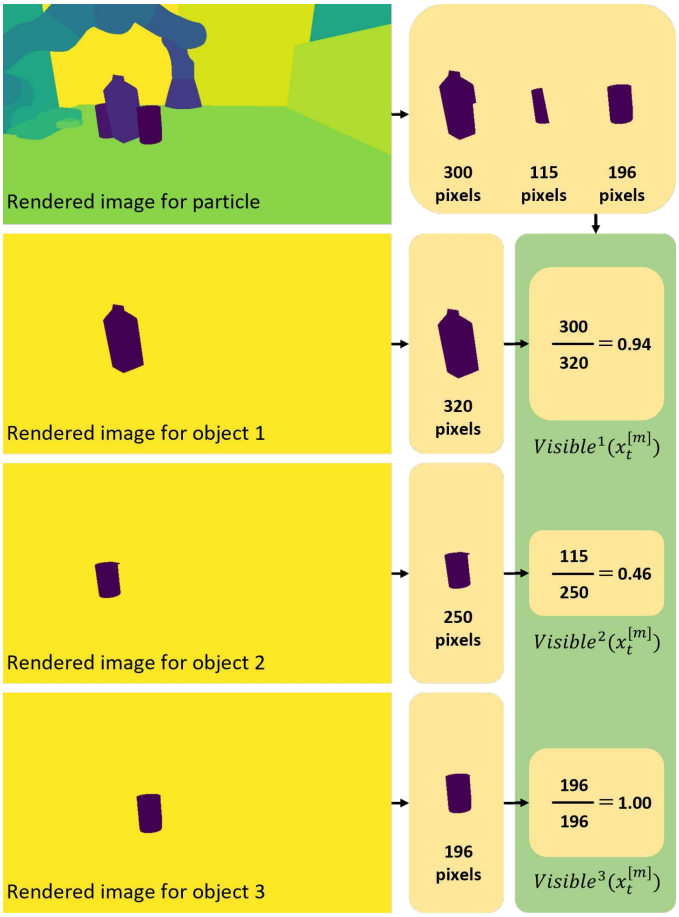


Fig. 4. Visibility score calculation for three objects in an example particle. A segmented/footprint image is rendered for each object jointly in the particle (top-left) and individually (bottom-three-leftmost) at their corresponding poses in the particle. The pixels belonging to each object are counted in each of these images (top-right and bottom-three-middle). A visibility score for each object is calculated as the ratio of the unoccluded footprint to the total possible footprint (bottom-three-rightmost).

D. Depth Images

To compute $p({}^{D}z_t \,|\, \bar{x}_t^{[m]})$ in Eq. 8, our approach again involves using the simulated camera to render a depth image for each particle at every time step. Specifically, after updating each particle by motion model (Sec. IV-A), we render a depth image based on the poses of objects inside the intermediate particle $\bar{x}_t^{[m]}$, and then compare the rendered depth image ${}^{D}\bar{z}_t^{[m]}$ with the real depth image ${}^{D}z_t$ captured by the camera. Each particle needs to render only one depth image encapsulating the pose state information of all objects. Again, we use GPU parallelization to render the depth images (Appendix A), which markedly improves efficiency.

An error score for each particle is obtained by comparing the rendered depth images with the real depth image. We use the error score, $e_t^{[m]}$, using a visible surface discrepancy error function similar to the one proposed by Lee et al. [52]. However, Lee et al. use depth information on a per object basis and therefore require segmentation of the real depth image. Instead, our joint multi-object treatment of the scene enables us to render the full scene directly, instead of relying on potentially inaccurate segmentation of individual objects.

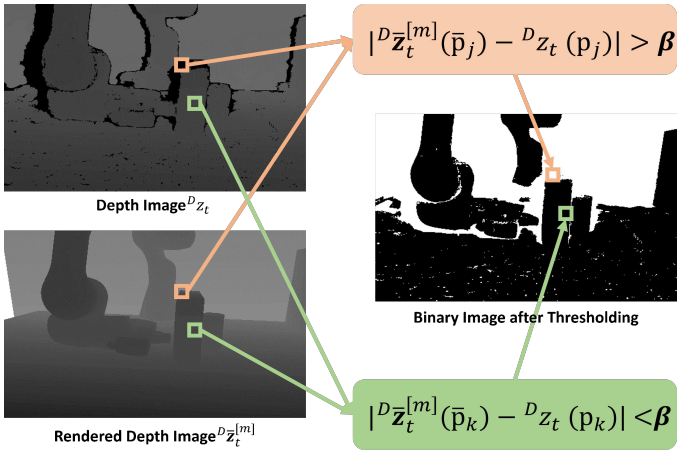


Fig. 5. We subtract the pixel values at corresponding locations between the two depth images. The absolute differences are then compared against a predefined threshold, β. For each pixel, if the absolute difference is less than β, the pixel in the resulting image is assigned a value of 0. Conversely, if the difference exceeds β, the pixel is assigned a value of 1.

We define our error score as:

$$e_t^{[m]} = \underset{\substack{\bar{\mathbf{p}} \in {}^{D}\bar{z}_t^{[m]} \\ \mathbf{p} \in {}^{D}z_t}}{\text{avg}} \begin{cases} 0 & \text{if } |{}^{D}\bar{z}_t^{[m]}(\bar{\mathbf{p}}) - {}^{D}z_t(\mathbf{p})| < \beta \\ 1 & \text{otherwise} \end{cases} \tag{13}$$

where $\bar{\mathbf{p}}$ and $\mathbf{p}$ represent the pixels in ${}^{D}\bar{z}_t^{[m]}$ and ${}^{D}z_t$ respectively, β represents the threshold value used to assess the degree of correspondence between pixel values. If the absolute depth difference between corresponding pixels is less than the threshold β, these pixels are assigned to 0. Otherwise, the pixels are assigned 1, as shown in Fig. 5. This binary classification of pixel distances results in a new set of pixel values. Subsequently, these values are averaged to calculate the $e_t^{[m]}$. Using this method, we compute error scores, $\{e_t^{[m]}\}_{m=1,\ldots,M}$, for all particles at time step t. Upon calculating the $\{e_t^{[m]}\}_{m=1,\ldots,M}$, a normalization operation is performed to get $p({}^{D}z_t \,|\, \bar{x}_t^{[m]})$:

$$p({}^{D}z_t \,|\, \bar{x}_t^{[m]}) = \frac{e_t^{[m]} - \min(e_t^{[1]}, \ldots, e_t^{[M]})}{\sum_{m=1}^{M} \left(e_t^{[m]} - \min(e_t^{[1]}, \ldots, e_t^{[M]}) \right)} \tag{14}$$

The motivation behind our model is to *quickly* compute a probability value without performing object-level segmentation or reasoning on the cluttered real depth images. If object-level segmentations/pose estimates are available on the real depth image, then more informative, and more symmetric, models can be built. However, object level segmentation/pose-estimation on depth images (e.g., ICP-like schemes) are time-consuming, and given our particular focus on highly occluded scenes, are not reliable.

E. Resampling

Multinomial resampling[53] is performed based on the weights of each particle to generate a new set of particles.

F. Computational Cost

To ensure efficient performance, the particle filter is updated (i.e., motion and observation updates are performed) at fixed time intervals, denoted as Δt. The most computationally demanding part of this process is the motion update, which relies on physics-based predictions. Consequently, Δt is selected based on the minimum time interval necessary to complete the physics simulations for all particles within the system.

V. Model Predictive Control (MPC) with Particle Filtering Feedback

In the previous section, we described a method for tracking the 6-D pose of objects. Here, we propose a Model Predictive Control (MPC) framework that uses feedback from the particle filtering algorithm to push an object to a goal region. The tracking system enables the MPC to control object poses even when they are occluded. Figure 9 shows example manipulation tasks, with varying number of objects.

A. MPC Framework

Alg. 1 presents a typical MPC framework. We start by getting an initial state of the system, such as the robot joints and the poses of objects, using a pose estimation system (line 2). In line 3, we generate a straight line trajectory from the end-effector's position to the goal region. We then repeatedly, until successful or timeout (line 4), call the optimizer on line 5, passing in the current state (x_{current}) and a trajectory (τ). We describe in detail how this optimizer works in Section V-B. If the optimization was unsuccessful, we terminate the MPC (lines 6 and 7). If optimization was successful, we execute n_u controls of the optimized trajectory in the real-world in line 8. In line 9, we update the trajectory by removing the executed controls. Finally, we update the simulator with feedback from the real-world in line 10.

We select a particle from the PBPF algorithm to update the simulation state in line 10. Recall that a particle represents the poses of all the objects. We select $x_t^{[m*]}$, the particle closest to the mean of the particle set, as follows:

$$x_t^{[m*]} := \arg\min_m d(x_t^{[m]}, x_t^{[\mu]}) \tag{15}$$

where $x_t^{[\mu]}$ is the mean of the particle set (computed in $\mathbb{SE}(3)$), and $d(x_t^{[m]}, x_t^{[\mu]})$ is the distance of a particle, $x_t^{[m]}$, to the mean. We calculate the distance of a particle by summing up the distances of all the object poses to their corresponding mean pose. The distance for an object combines the Euclidean positional distance (δ_p) and the rotational distance (δ_θ; minimum angle between two orientations): $w_{\delta_p} \cdot \delta_p + w_{\delta_\theta} \cdot \delta_\theta$. We call this MPC variant, MPC-PBPF.

A simple MPC baseline can use feedback from a pose estimation system, like DOPE, in line 10 instead. We implement such an MPC baseline and call it MPC-DOPE. We compare MPC-PBPF, MPC-DOPE, and an open-loop system without feedback in Section VI.

Algorithm 1 Model Predictive Control (MPC) Framework

1: **procedure** MPC
2: $x_{\text{current}} \leftarrow$ initialise simulation state from real-world
3: $\tau \leftarrow$ initialise trajectory
4: **while not** successful **and not** timeout **do**
5: τ, optimizationSuccessful $\leftarrow$ OPTIMIZE(x_{current}, τ)
6: **if not** optimizationSuccessful **then**
7: **return**
8: execute n_u controls from τ in the real-world
9: $\tau \leftarrow$ remove the n_u executed controls from τ
10: $x_{\text{current}} \leftarrow$ update simulation state from real-world

Algorithm 2 Stochastic Trajectory Optimization

1: **function** OPTIMIZE(x_{current}, τ)
2: $S \leftarrow$ rollout τ from x_{current}
3: obtain the cost of rollout using $\mathcal{C}(S)$
4: **while not** successful **and not** convergent **do**
5: sample k noisy trajectories from τ
6: rollout each noisy trajectory
7: obtain cost for each rollout using $\mathcal{C}$
8: $\tau \leftarrow$ best trajectory among the k
9: **return** τ, optimizationSuccessful

B. Stochastic Trajectory Optimizer

Algorithm 2 presents the stochastic optimizer [11], [54] that we use in Algorithm 1 and line 5. We start by rolling out the trajectory, τ (controls in the joint-space), using a physics simulator from the current state, x_{current}, to get a state sequence S (line 2). We compute the cost over that state sequence using a weighted cost function, $\mathcal{C}(S) : S \to \mathbb{R}$, in line 3, as follows:

$$\mathcal{C} = \sum_{i=1}^{|S|} w_1 \cdot d_{\text{g}} + w_2 \cdot d_{\text{o}} + w_3 \cdot d_{\text{z}} + w_4 \cdot \dot{q}_{\text{ee}} + w_5 \cdot c_s \tag{16}$$

where d_{g} is the Euclidean distance from the object to the goal region, d_{o} is the Euclidean distance from the end-effector to the object, d_{z} is the deviation of the end-effector along the z-axis from its initial state (keeping the robot gripper parallel to the table), $\dot{q}_{\text{ee}}$ is the end-effector's linear velocity, and c_s is an indicator function for collision with static obstacles. If successful, we just return the solution without any optimization (line 9). If not, we start by sampling k trajectories from τ by introducing Gaussian noise to the controls (i.e., adding noise to each of the joints over the full trajectory horizon) (line 5). Then, we rollout each of the k trajectories (line 6). We compute a cost for each rollout (line 7). Note that we parallelise lines 5-7. In line 8, we pick the trajectory with the lowest cost. We repeat until we find a successful trajectory or converge to a local minimum. If we hit a local minimum, the task is declared as a failure and we return false for `optimizationSuccessful`. While susceptible to local minima, local trajectory optimizers remain effective for generating contact-based manipulation trajectories [11], [54], balancing computational efficiency with the ability to escape some local minima by carefully tuning noise injection.

TABLE I
EXAMPLE TASKS FOR MULTI-OBJECT 6D POSE TRACKING EXPERIMENTS
AND COMPARISON OF PERFORMANCE OF FOUNDATIONPOSE (FOUN) AND PBPF-RGBD METHODS

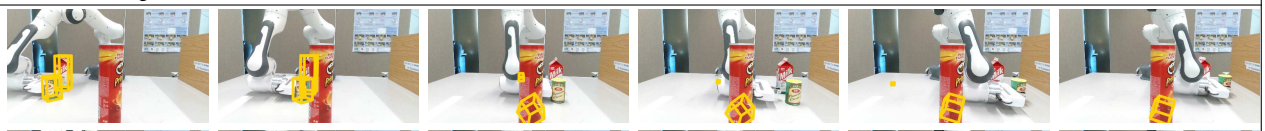
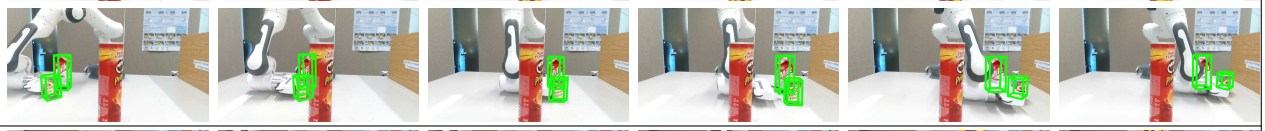
	Method	$time \rightarrow$
One Object Push: Please see Fig. 2		
Two objects push	FOUN	
	PBPF-RGBD	
Three objects push	FOUN	
	PBPF-RGBD	

TABLE II
OBJECTS USED IN TRACKING EXPERIMENTS

YCB Objects [21]		HOPE Objects [22]					
Cracker	**Soup**	**Ketchup**	**Mayo**	**Milk**	**SaladDressing**	**Parmesan**	**Mustard**
							

Please note that the input trajectory, τ, can be an initial rough trajectory (for example, from line 3 of Algorithm 1) or a warm-start with a good trajectory from a previous optimization process (as we interleave planning and execution during MPC).

VI. EXPERIMENTS & RESULTS

In this section, we present two sets of experiments evaluating the performance of our methods.

First, we evaluate the *tracking performance*. In these experiments, the robot executes a pre-determined motion (i.e., without MPC) interacting with the objects. We evaluate how accurate the estimated poses of the objects are (as compared to ground truth poses) during the course of the robot motion. These experiments evaluating the tracking performance are presented in Sec. VI-A to Sec. VI-C.

Second, we evaluate the *control performance*, where an MPC controller (as discussed in Sec. V) pushes an object to a goal region. In these experiments, we evaluate how successfully the object is pushed into a goal region when the MPC controller uses our physics-based pose tracking methods as opposed to other baseline pose estimation methods. These experiments evaluating the control performance are presented in Sec. VI-D to Sec. VI-F.

A. Tracking Tasks and Metrics

To verify the performance of tracking methods, we create different non-prehensile manipulation scenes. In these scenes, the robot performs an open-loop pre-defined motion. Examples can be seen in Fig. 2 and Table I. We create 50 similar scenes:

- **One-object-push:** An example scene is shown in Fig. 2. We create 20 similar scenes with different objects.
- **Two-objects-push:** An example scene is shown in Table I. We create 20 similar scenes with different objects.
- **Three-objects-push:** An example scene is shown in the bottom two rows of Table I. We create 10 similar scenes with different objects.

The above experiments are configured such that, during the manipulation, the objects can be partially or fully obscured by the robotic arm, obstacles, or other objects. This creates significant challenges to the tracking accuracy and effectiveness. We use a variety of objects from two datasets, YCB-dataset [21] and HOPE-dataset [22], as shown in Table II. These objects include symmetric objects, and of varying sizes, demonstrating the applicability of our method to a wide range of objects. In all scenes, tracked objects only interact with the robot and other tracked objects. For the ground truth object poses, we used the OptiTrack system [55], which uses reflective markers placed on the objects for precise pose estimation, as shown in Fig. 6. We use an ar-marker on the robot to find the pose of

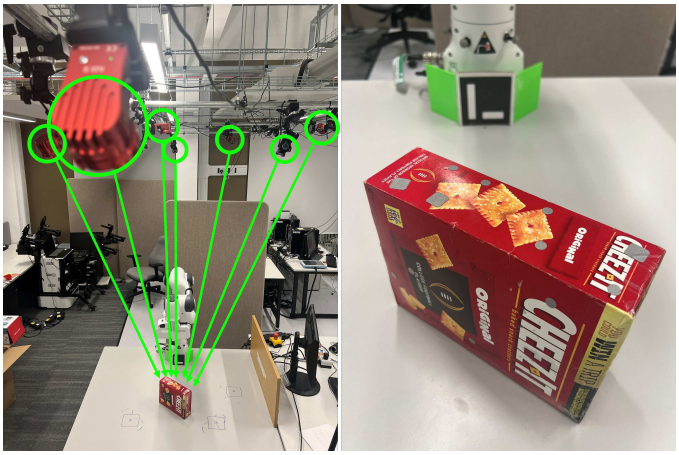


Fig. 6. The OptiTrack system is used to determine the ground truth of the target objects. As shown in the left image, multiple cameras are mounted on the ceiling to track the reflective markers on the objects. The right image shows the reflective markers attached to the target object.

the camera with respect to the robot, i.e., for extrinsic camera calibration.

Each of our runs contain an average of 8630 time-steps, i.e., data points at which we have an RGB-D image, robot joint information, and object pose ground truth information. Over 50 runs, we have a total of 431500 data points. The results we present below are averaged over these data points.

To evaluate the accuracy of pose tracking, we employ two metrics: the Average Distance of Discrepancy (ADD) [30] and the Symmetric Average Distance of Discrepancy (ADD-S). ADD measures the mean Euclidean distance between corresponding points on the tracked object model and the object model at the ground truth pose. ADD-S extends this by considering the mean distance under the best permutation of model points, making it robust to symmetrical objects where orientation discrepancies might otherwise be misleading. We specifically report the *area under curve* (AUC) [30] of ADD and ADD-S. This is the metric most commonly used to evaluate pose estimation systems' accuracy for varying accuracy thresholds [1], [2], [29]–[31].

We release this dataset of 50 experiments (431500 data points) including time-stamped RGB-D data, robot joint values, object pose ground truth, and object/robot/environment 3D models, for others in the community to be able to use not only the RGB-D information (which is often provided in other similar datasets) but also physics-based inference (enabled by timestamped robot joint values and 3D models).

B. Implementation of Tracking Methods

We evaluated four different methods for tracking objects[4].

1) **Diff-DOPE:** Diff-DOPE [29] refines the object's pose at every time step, using prior estimates provided by DOPE [1], a single-snapshot object pose estimation system. We used the official Diff-DOPE implementation[5].

2) **Diff-DOPE (Tracking):** We modified Diff-DOPE such that it uses a different prior estimate at each time step. Instead of using DOPE, we use the pose estimated by Diff-DOPE in the previous time step as the prior estimate. This makes Diff-DOPE capable of using information from previous time-steps, similar to our methods. For the very first frame of a run, we still use DOPE as the prior.

3) **FoundationPose:** FoundationPose [2] uses a combination of deep learning and geometric optimization to track an object. It is one of the leading methods listed on the worldwide BOP leaderboard[6] (at the time of writing). We used the official FoundationPose implementation[7].

4) **PBPF:** Our Method as explained in Sec. IV. Particularly, we implemented three versions of our method: **PBPF-RGB**, **PBPF-D** and **PBPF-RGBD**. The distinction among the three versions lies solely in their observation model. Specifically, PBPF-RGB uses only RGB images (Section IV-C), PBPF-D relies only on depth images (Section IV-D)[8], and PBPF-RGBD integrates both RGB and depth images (Section IV-C and IV-D) as its observation model. **Implementation details:** The implementation details and the parameter values we used in all PBPF versions are: (a) *Motion model parameters.* We used the same coarse friction and restitution parameters for all objects: $\mu_{\text{friction}} = 0.1$ and $\sigma_{\text{friction}} = 0.3$, with the minimum sampled value capped at 0.001. $\mu_{\text{restitution}} = 0.9$ and $\sigma_{\text{restitution}} = 0.2$. We used different mass estimates for objects: μ_{mass} of Cracker, Soup, Milk, Parmesan and Mustard are 0.45 kg, 0.35 kg, 0.04 kg, 0.035 kg and 0.05 kg, respectively. μ_{mass} of Ketchup, Mayo and SaladDressing are the same at 0.06 kg. The σ_{mass} for Cracker and Soup is 0.5, and 0.1 for the remaining objects, with a minimum sampled value capped at 0.02 kg. σ_f: For position 0.005 m, for rotation 0.05 radians. We used the PyBullet physics engine [15] as the physics model, f_θ. We employ a multi-processing approach to initialize and manage each particle within individual PyBullet environments, using all 18 (36 virtual) CPU cores available on the computer. (b) *Observation model parameters.* As the PE system (used to estimate p_{PE} as explained in Sec. IV-C), we use DOPE since it is a single-pass neural network that was fastest among the ones we have tested, with relatively accurate pose estimation for known objects. σ_{PE}: For position 0.1 m and rotation 0.2 radians. Visibility parameters: the threshold V used for comparing $Visible^i(x_t^{[m]})$ to decide object visibility is set to 0.55 for all objects (except 0.45 for the large Cracker object) when the PE system detects the object. If the PE system does not detect the object, the threshold V is set to 0.6 for all objects (0.5 for the Cracker); $\alpha_w = 0.33, \alpha_h = 0.6, \alpha_l = 0.55$ for all objects ($\alpha_h = 0.75, \alpha_l = 0.45$ for Cracker). Depth parameters: $\beta = 0.03$ m. (c) *Update frequency.* $\Delta t = 0.25s$. (d) *Number of particles.* Our tests with different numbers of particles showed accuracy to plateau around 50 particles. On the other hand, due to computational cost, the number of particles needs to be reduced as more objects are tracked to allow the PBPF algorithm to run at the same update frequency. We use the

[4]Experiments were performed on CPU: Intel(R) Xeon(R) W-2295 CPU@3.00GHz; GPU: NVIDIA GeForce RTX 3080; RAM: 192906 Mb

[5]https://github.com/NVlabs/diff-dope

[6]https://bop.felk.cvut.cz/leaderboards

[7]https://github.com/NVlabs/FoundationPose

[8]PBPF-D uses the RGB-based DOPE at $t = 0$ to initialize the objects' poses.

TABLE III
OVERALL AVERAGE TRACKING ACCURACY FOR DIFFERENT METHODS
(↓ REPRESENTS LOWER IS BETTER, ↑ REPRESENTS HIGHER IS BETTER)

Method	ADD ↓	AUC-ADD ↑	ADD-S ↓	AUC-ADDS ↑
Diff-DOPE	0.125	42.8	0.101	53.8
Diff-DOPE (Tracking)	0.187	30.9	0.153	39.9
FoundationPose	0.118	52.3	0.100	59.8
PBPF-D	0.065	58.8	0.049	70.0
PBPF-RGB	0.088	46.0	0.068	59.6
PBPF-RGBD	**0.030**	**70.1**	**0.021**	**79.2**
PBPF-RGBD (Best Particle)	0.016	83.7	0.012	87.6

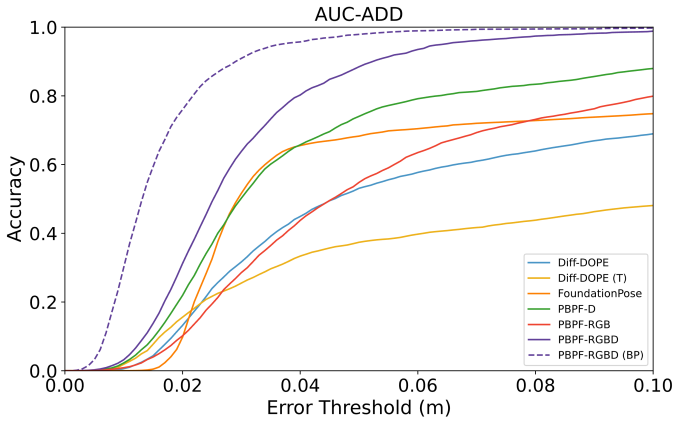


Fig. 7. Accuracy versus Error threshold plots for different methods, used to compute Area Under the Curve (AUC). The horizontal axis represents the increasing threshold of error allowed, ranging from 0.00 to 0.10 meters, and the vertical axis shows the accuracy of the methods, ranging from 0 to 1, for that given threshold. (BP means Best Particle and T means Tracking.)

following particle counts: $M = 70$ for one object, $M = 50$ for two objects, and $M = 40$ for three objects. (e) *Initialization*. We initialize particles at $t = 0$ by sampling from a Gaussian distribution. We use the pose from the PE system at $t = 0$ as the mean pose, and the standard deviations for initialization are 0.03 m and 0.2 radians. During initialization, if objects are penetrating each other or the robot, we move them in the direction of the contact normal until they are not penetrating.

C. Pose Tracking Results

Table III presents the overall average tracking performance of different methods. We evaluated each method 100 times[9] in each of the 50 runs (giving us 100 different estimates for each of the 431500 data points) to accommodate the inherent randomness in some of these methods (e.g., the probabilistic nature of particle filtering). If a method failed to output a pose estimation at a certain time-point, we used its most recent output. PBPF outputs a set of particles, not a single pose estimate. To compute the accuracy values, we used the particle that is closest to the mean of the particles. We also present the AUC curves in Fig. 7 (which show accuracy vs error threshold and are used to compute the area under the curve)

The results presented in the table show that PBPF-RGBD performs significantly better, compared to baselines. This is because our method uses the additional robot control, and the physics-rollout, information while estimating objects' poses.

Among our results, we also show the accuracy of the "best particle" from the particle set (the last row of Table III). The "best particle" is the particle that is closest to the ground truth pose. While it is impossible to know which particle is the best particle when one does not have access to ground truth, we report this, since it can be useful to put a bound on the expected error for conservative/worst-case manipulation planning systems, i.e., systems that plan robust robot motion that take into account all particles in the particle set.

Table IV presents a more detailed view, showing ADD and ADD-S results for different objects, including data from all our 50 scenes (i.e., including when these objects may be in the scene together with multiple other objects). These results show that the PBPF-RGBD algorithm performs better consistently over all objects of various shapes and properties we have tested.

It is also clear from the results that using both RGB and depth information is useful and necessary to achieve the best performance. Table III demonstrates that PBPF-RGBD significantly outperforms PBPF-D (implying the value of RGB information) and PBPF-RGB (implying the value of depth information). We also note that PBPF-D outperforms PBPF-RGB. We think there are two reasons for this. First, PBPF-D in fact uses RGB information, even if only at initialization (at $t = 0$), which puts it at an advantage when compared to PBPF-RGB. Second, in our scenes, where significant occlusions frequently occur, the accuracy of RGB-only pose estimation systems, e.g., DOPE, suffers drastically.

However, RGB is particularly useful when objects have similar shapes. For example, the error results of the Mustard and SaladDressing objects in Table IV reveal that PBPF-RGBD performs notably better than PBPF-D. This is due to the method's difficulty distinguishing between the similarly sized Mustard and SaladDressing objects when only depth images are used, leading to decreased accuracy. This also applies to other target objects with similar size and shape.

We show some visual examples of tracking performance in Fig. 2 and Table I. We use wireframes overlaid on the images to show the estimated poses of different methods (orange for FoundationPose and green for our method). (For our method, we draw the wireframe at the pose of the particle closest to the mean of the particles.) As can be seen in the figures, when all objects are visible, both systems perform

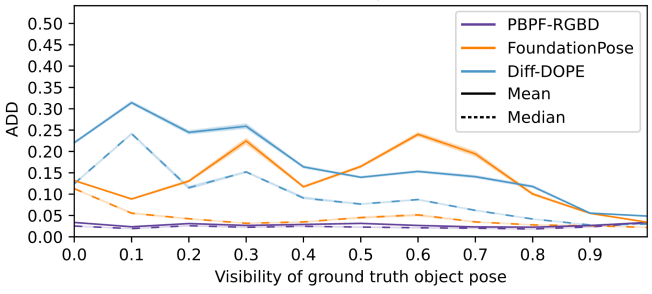


Fig. 8. Estimation error (ADD) vs. visibility of ground truth object poses.

[9]All robot/camera/ground truth data were recorded in a ROS bag file during actual robot manipulation. These bag files were then replayed, respecting timestamps, 100 times for each different method.

TABLE IV
OBJECT-SPECIFIC ACCURACY OF TRACKING FOR DIFFERENT METHODS

Method	Cracker		Soup		Ketchup		Mayo		Milk		Mustard		Parmesan		SaladDressing		Mean	
Metric	ADD ↓	ADD-S ↓	ADD	ADD-S	ADD	ADD-S	ADD	ADD-S	ADD	ADD-S	ADD	ADD-S	ADD	ADD-S	ADD	ADD-S	ADD	ADD-S
Diff-DOPE	0.152	0.109	0.075	0.052	0.076	0.061	0.155	0.137	0.118	0.092	0.177	0.152	0.118	0.100	0.093	0.075	0.125	0.101
Diff-DOPE (T)	0.170	0.105	0.162	0.133	0.207	0.176	0.191	0.167	0.150	0.117	0.191	0.162	0.178	0.151	0.299	0.268	0.187	0.153
FoundationPose	0.049	0.039	0.071	0.054	0.137	0.124	0.080	0.065	0.223	0.208	0.133	0.109	0.115	0.093	0.148	0.121	0.118	0.100
PBPF-D	0.070	0.055	0.030	0.018	0.037	0.026	0.070	0.051	0.037	0.025	0.132	0.104	0.042	0.028	0.122	0.107	0.065	0.049
PBPF-RGB	0.093	0.062	0.065	0.043	0.057	0.043	0.071	0.054	0.084	0.066	0.071	0.051	0.076	0.055	0.240	0.230	0.088	0.068
PBPF-RGBD	**0.032**	**0.025**	**0.029**	**0.017**	**0.028**	**0.020**	**0.028**	**0.020**	**0.027**	**0.019**	**0.035**	**0.024**	**0.031**	**0.020**	**0.032**	**0.021**	**0.030**	**0.021**

well in estimating object poses. However, when there are occlusions (either behind other objects or behind the robot), FoundationPose estimates diverge significantly, whereas PBPF is able to estimate physically plausible poses for objects. Video versions can be seen in the multimedia attachment.

An advantage of our method is that it provides consistent tracking of objects even when there are heavy occlusions. To analyze this better, in Fig. 8, we evaluate each method's performance when the target object is at different levels of visibility to the camera. Here, the visibility values are computed similar to the process shown in Fig. 4, but this time using the ground truth poses of the objects. In Fig. 8, the solid lines show the mean ADD scores for all methods, computed over all objects at all time-steps in all of our 50 scenes. PBPF-RGBD demonstrates impressive accuracy across all visibility levels. Diff-DOPE's performance improves as visibility of the object improves, but stays well above PBPF-RGBD performance. FoundationPose performs well when the object is fully visible (i.e., visibility is 1.0), however suffers significantly, performing even worse than Diff-DOPE, at visibility values around 0.6. This is because, when FoundationPose loses track of the object behind occlusion, its accuracy suffers drastically, even after the object re-appears in the scene and becomes visible. (This can also be observed in Fig. 2, Table I, and the attached video.) To understand FoundationPose performance when we remove the effects of such highly erroneous estimates on the mean, in Fig. 8, we also present the median ADD performance with the dashed line. The median performance of FoundationPose improves as visibility improves, as expected. PBPF-RGBD's mean and median performances are almost identical, showing robustness, due to the stabilizing effect of the physics constraints.

D. Control Tasks and Metrics

In addition to the experiments above, we also performed experiments to show how the tracking of objects can be used for goal-directed manipulation in clutter. To evaluate the effectiveness of our method in manipulation tasks using MPC (Sec.V), we designed four different tasks as shown in Fig. 9:

- **1-Object**: The robot pushes an object to a target area. Two different target areas are used for different instances of this task.
- **2-Objects:** The robot pushes an object to a target area, while there is a second object in the way. Two different target areas are used for different instances of this task.
- **3-Objects:** The robot pushes an object to a target area, while there are two other objects in the way. Two different target areas are used for different instances of this task.
- **Object-Object:** The robot pushes an object to a target area, but indirectly, by pushing another object.

We ran each method 3 times for the same task and the same target area; i.e., in total we ran each method 21 times (6 times on 1-Object tasks, 6 times on 2-Objects tasks, 6 times on 3-Objects tasks, and 3 times on Object-Object tasks).

In executing control tasks, our primary consideration is whether the target object has reached the target area. The target area is defined as a circle with a radius of 5 cm. Upon completion of a run, we assess success by measuring whether the object's centroid lies within this predefined target area. We run each control method for a maximum of 1000 controls, and if the target area is not reached by then, the method reports failure. The parameters for our MPC algorithm are: $w_{\delta_p} = 0.7$, $w_{\delta_\theta} = 0.3$, $n_u = 100$. The parameters for the stochastic trajectory optimiser are: $|\tau| = 1000$, $w_1 = 60000$, $w_2 = 20000$, $w_3 = 45000$, $w_4 = 3000$, $w_5 = 10000$, $k = 40$, we sample noise for each of the 7-DOF from a Normal distribution with mean 0 and $\sigma = 0.3$.

E. Control Methods

We use three different methods to perform the tasks:

- **OPENLOOP:** This method operates without any feedback during the manipulation task except the initial time-step. At the initial time-step, the object poses are estimated using DOPE, and Alg. 2 is used to optimize one trajectory to the goal. This trajectory is then executed open-loop, without any feedback.
- **MPC-DOPE:** This method uses Alg. 1. As state feedback, it uses the pose estimate of the objects provided by the DOPE algorithm at each time-step.
- **MPC-PBPF:** This method uses Alg. 1. As state feedback, it uses the pose estimate of the objects provided by the PBPF-RGBD algorithm at each time-step.

F. Results of Control Experiments

Table V shows the success rates of different methods in different tasks. In all 21 experiments, MPC-PBPF successfully moved the target object to the target area, even when there were significant occlusions of the target and other objects. While OPENLOOP and MPC-DOPE were also successful for

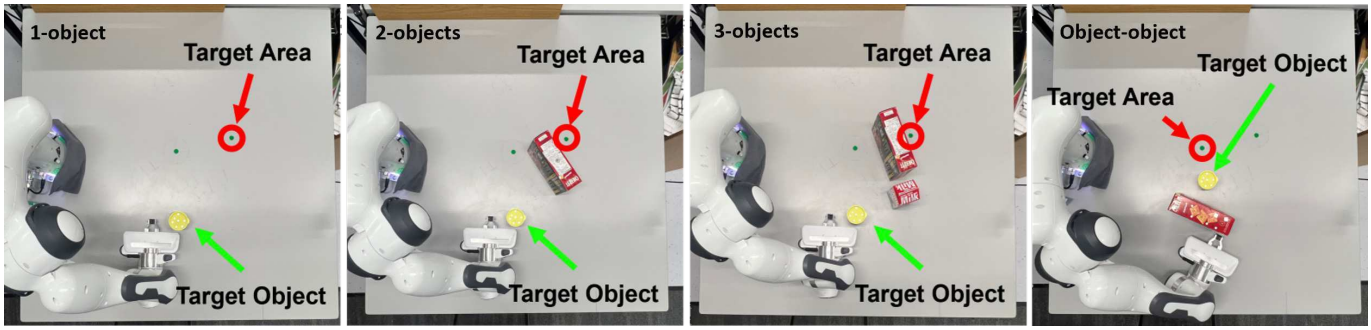


Fig. 9. Examples of four different manipulation tasks, where an MPC controller aims to push a target object to a target area among other objects, either directly (first three tasks on the left), or indirectly by pushing another object (rightmost task).

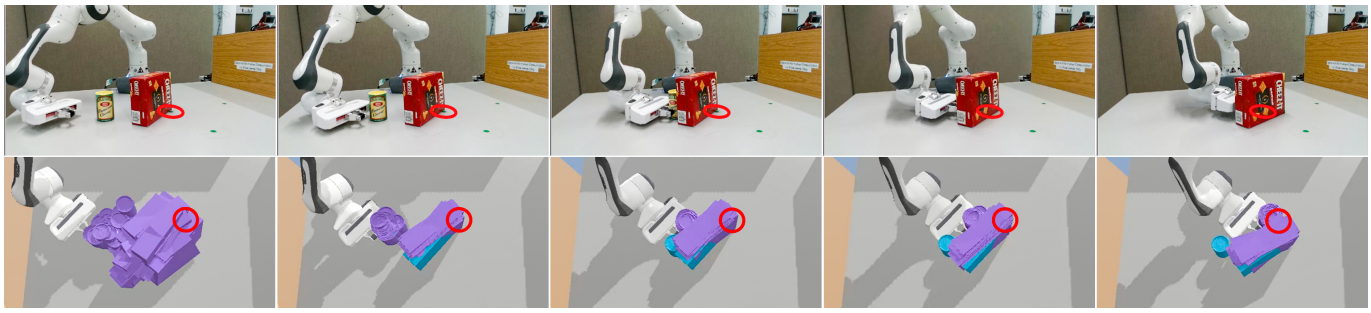


Fig. 10. Example MPC-PBPF execution, where the task is to push the cylindrical Parmesan object to the target area (shown as a red circle), while there is another object in the way (Cheezit object). In the second row, the particles of PBPF-RGBD are shown in purple at each timestep. The blue particles show the poses reported by DOPE at those time-steps.

TABLE V
SUCCESS RATES OF CONTROLLERS ON DIFFERENT MANIPULATION TASKS

	OPENLOOP ↑	MPC-DOPE ↑	MPC-PBPF ↑
1-Object	6/6	6/6	6/6
2-Objects	2/6	1/6	6/6
3-Objects	1/6	0/6	6/6
Object-Object	0/3	0/3	3/3
Total	9/21	7/21	21/21

the easier 1-Object task, they had significantly lower success rates for tasks with more objects, where the lack of accurate knowledge of object poses was detrimental.

Fig. 10 shows MPC-PBPF performance in an instance of 2-Objects task. The second row shows the particles in purple, while it also shows the DOPE detections of the objects in blue. The red circle shows the target region. Even when the target object (Parmesan) is completely occluded, MPC-PBPF can still accurately push the object to the target area. In contrast, when the DOPE method is employed, occlusion of the object leads to incorrect predictions, resulting in manipulation task failures. Similarly, Fig. 1 shows MPC-PBPF performance in an instance of 3-Objects task. Corresponding videos of these experiments can be seen in the multimedia attachment.

VII. Further Analysis & Limitations

In this section, we perform additional experiments and analysis to understand the limitations of our approach.

A. Robustness to uncertainty in object friction and mass

Our method requires mean and variance values for the mass (μ_{mass} and σ^2_{mass}) and friction ($\mu_{friction}$ and $\sigma^2_{friction}$) parameters of the objects. We performed experiments, analyzing how sensitive our method's performance is to the accuracy and uncertainty of these physics parameters of objects. We present these results in Table VI, where the top sub-table (the first three rows of the table) shows the results of experiments where we varied only the μ_{mass} of objects and measured the performance of our method PBPF-RGBD. As shown in the table, for these experiments, as μ_{mass} we used the values 0.01, TM, 0.5, 1.00, and 5.00. (TM stands for the "true mass" of an object, as reported in Sec. VI-B.) Since our complete dataset includes scenes with objects of a wide range of true mass values (some ten times heavier than the others), to make the effect of varying mass values clear, in these experiments we only used the subset of the dataset that includes scenes with objects of similar true mass between 0.03 kg and 0.06 kg, corresponding to Parmesan, Mustard, Ketchup, Mayo, SaladDressing. (Hence, the TM results are slightly different than the results we report for our method in Table III using the complete dataset). The results show that, while the method with the true mass, TM, performs best, other μ_{mass} values did not result in drastic drops in the performance. We hypothesize that there are two reasons for this. (1) Our method is capable

of showing some robustness due to the intentional uncertainty introduced into the system using the sampling variances. To test this, we ran our method again with different μ_{mass} values, but this time with the three variance parameters set to zero ($\sigma_{friction}, \sigma_{mass}, \sigma_f = 0$), as shown in the second sub-table. As expected, the overall performance significantly suffered, but also the difference between the TM performance and other mass values was more significant, confirming our intuition. (2) Mass of an object is less consequential in slower interaction tasks, and our dataset heavily includes slower pushing tasks. This agrees with quasi-static analysis of sliding/pushing [56], where it is shown that, when object accelerations are negligible, (i.e., when the energy induced into the object through the robot finger is small enough that it is *immediately* dissipated by the object-ground frictional forces, and hence the pushed object does not lose contact with the finger during pushing) the mass of the object can mostly be ignored in predicting its motion. To test this, we performed new experiments with the robot, where we poked an object with high impact, so that the object accelerates and keeps sliding after contact. (We present more details about this experiment in Sec. VII-D.) The third sub-table (μ_{mass}(Poking)) shows the results, where the effects of mass are more significant, again confirming our insight.

Similarly, we performed experiments using different values for the $\mu_{friction}$ parameter of our method. While measuring the correct coefficient of friction is more difficult than measuring mass, we estimate it to be between 0.1 and 0.25 for our objects, estimated using a protractor and sloped surface. The results in the sub-tables show that the method performs best with correct friction values, and again that setting ($\sigma_{friction}, \sigma_{mass}, \sigma_f = 0$) makes this more pronounced.

We also measured the performance under varying variances of the physics parameters, shown in sub-table "$\sigma_{friction/mass}(\sigma_f = 0)$". For these experiments, we set $\sigma_f = 0$, since it would interfere with assessing the impact of variance values for friction and mass. In this sub-table, a value of $0.1\times$ indicates scaling the original values reported in Sec. VI-B by 0.1, and similarly for other columns. As expected, setting the variance values too low or too high degraded performance, since too low values make the filtering less robust to uncertainty, but too high variance values diminishes the information provided by the physics model.

B. Effect of motion noise, σ_f

As shown in the bottom sub-table of Table VI, we varied the positional and rotational motion noise, σ_f, to investigate the effect of this parameters on the performance of our system. The results show that tuning this parameter is important, as the performance of the system is significantly effected. Note, however, that the motion model noise σ_f reflects the discrepancy between the physics model and the real-world physics, and therefore should be estimated only once for the physics model (e.g., physics engine) and can then be fixed. Therefore, given a dataset with a ground truth (such as the dataset we are providing), if a new physics engine is used with our method, σ_f should be calibrated to the best performing value, and can then be fixed for later use of the method.

TABLE VI
TRACKING ACCURACY FOR DIFFERENT PARAMETRIZATIONS

μ_{mass}	0.01	TM	0.50	1.00	5.00
ADD ↓	0.028	**0.025**	0.028	0.029	0.029
AUC-ADD ↑	71.9	**74.2**	71.6	71.1	71.0
$\mu_{mass}(\sigma_{fri}, \sigma_{mass}, \sigma_f = 0)$	0.01	TM	0.50	1.00	5.00
ADD ↓	0.093	**0.069**	0.101	0.094	0.103
AUC-ADD ↑	37.5	**46.1**	35.8	34.2	32.4
μ_{mass}(Poking)	0.01	TM	0.50	1.00	5.00
ADD ↓	0.058	**0.046**	0.065	0.060	0.060
AUC-ADD ↑	51.8	**55.7**	50.1	48.9	48.8
$\mu_{friction}$	0.01	0.10	0.25	0.75	1.00
ADD ↓	0.043	**0.028**	0.031	0.041	0.052
AUC-ADD ↑	63.4	**72.2**	69.8	65.7	63.2
$\mu_{friction}(\sigma_{fri}, \sigma_{mass}, \sigma_f = 0)$	0.01	0.10	0.25	0.75	1.00
ADD ↓	0.087	**0.067**	0.069	0.088	0.119
AUC-ADD ↑	41.3	**56.2**	50.0	41.5	32.5
$\sigma_{friction/mass}(\sigma_f = 0)$	0.1×	1.0×	2.0×	5.0×	10×
ADD ↓	0.138	**0.053**	0.057	0.056	0.069
AUC-ADD ↑	31.7	**54.2**	52.6	51.2	48.6
σ_f pos:	0.000	0.005	0.010	0.020	0.050
σ_f rot:	0.00	0.05	0.10	0.20	0.50
ADD ↓	0.063	**0.028**	0.080	0.170	0.693
AUC-ADD ↑	53.8	**72.2**	50.0	34.6	9.70

TABLE VII
OVERALL TRACKING ACCURACY FOR OTHER SCENES

1-Object	Diff-DOPE	FoundationPose	PBPF-RGBD
ADD ↓	0.132	0.126	**0.029**
AUC-ADD ↑	43.4	52.1	**71.1**
2-Objects	Diff-DOPE	FoundationPose	PBPF-RGBD
ADD ↓	0.098	0.191	**0.029**
AUC-ADD ↑	46.3	41.5	**70.7**
3-Objects	Diff-DOPE	FoundationPose	PBPF-RGBD
ADD ↓	0.147	0.043	**0.031**
AUC-ADD ↑	38.9	63.0	**69.0**
4-Objects	Diff-DOPE	FoundationPose	PBPF-RGBD
ADD ↓	0.218	0.054	**0.038**
AUC-ADD ↑	25.0	52.7	**61.9**
5-Objects	Diff-DOPE	FoundationPose	PBPF-RGBD
ADD ↓	0.195	0.055	**0.049**
AUC-ADD ↑	23.2	49.2	**59.7**
Poking (object always visible)	Diff-DOPE	FoundationPose	PBPF-RGBD
ADD ↓	0.029	**0.028**	0.048
AUC-ADD ↑	71.4	**72.3**	56.2
Poking (object with occlusion)	Diff-DOPE	FoundationPose	PBPF-RGBD
ADD ↓	0.090	0.191	**0.049**
AUC-ADD ↑	40.9	22.4	**52.9**
Identical Objects	Diff-DOPE	FoundationPose	PBPF-RGBD
ADD ↓	0.137	0.037	**0.026**
AUC-ADD ↑	25.4	63.1	**72.9**

C. Effect of number of objects

An important limitation of our system is the computational cost of physics simulations, which increase with the number of physically interacting objects. In our original dataset, in Sec. VI-A), our experiments were limited to 3 simultaneously interacting objects. Up to three objects, our system did not show a degradation in performance (the first three sub-tables of Table VII). Performing experiments involving four or more objects presented significant challenges, particularly in acquiring accurate ground truth pose information. Furthermore, as

TABLE VIII
TIME (S) CONSUMED BY EACH COMPONENT

	1-obj M=70	2-obj M=50	3-obj M=40	4-obj M=40	5-obj M=40
Physics Update	0.102	0.096	0.106	0.141	0.167
Depth Update	0.101	0.073	0.073	0.089	0.092
RGB Update	0.031	0.029	0.040	0.051	0.055
Total Time	0.239	0.204	0.227	0.295	0.329

more objects are tracked, the update time increases, making it difficult to update our filter frequently. However, we still wanted to evaluate the performance of our method when the number of objects are increased. Therefore, we set up a scene with 4 objects and another with 5 objects, and performed pushing on these objects, carefully making sure that we still have ground truth readings (i.e., objects are visible to the OptiTrack cameras). We show one of these scenes in Fig. 12-top-left and also in the attached video. We show the performances of our method and baselines in the "4-Objects" and "5-Objects" sub-tables of Table VII. For these two scenes, we used only 40 particles, similar to our 3-object scenes. As expected, our method's performance is significantly degraded compared to its performance on our original dataset. Still, PBPF-RGBD performs better compared to baselines, since occlusions (which happen frequently in such crowded scenes) are much more of an issue for the baseline methods. In Table VIII we present the computational time each component of our method takes for scenes with different number of objects, and when different number of particles, M, is used. These results are averaged over all runs with those number of objects in them. (Standard deviations are not shown, but are negligible.) As shown, the main bottleneck is clearly the physics update.

A realistic manipulation scenario is unlikely to involve tens of interacting objects that needs to be tracked. Therefore, in realistic settings, our system can still be beneficial.

D. High-impact interactions

Through our experiments, we also discovered that our system's performance degrades with highly dynamic motion of the objects, for example when an object drops down to the table, as shown in Fig. 11. In such cases where objects went through high-impact contact interactions, the inaccuracy of the physics predictions and the fast motion of the objects degraded the performance. While it proved difficult to collect ground pose truth information in scenes such as Fig. 11, to explore this issue further, we conducted experiments where the robot performs a fast planar hit on the object to create an impact interaction, resulting in the object sliding away from the hand, as shown in Fig. 12 bottom-two rows, and in the attached video. We call this "Poking". The performance of different methods for Poking is shown in Table VII. When the poked object is completely visible, our method performs significantly worse. Here, the inaccurate physics predictions hinder, rather than help, tracking. Still, when we experimented with a scene where the poked object was occluded for part of its slide on the table (bottom row of 12), PBPF-RGBD performed better than the baselines, as shown in the table.

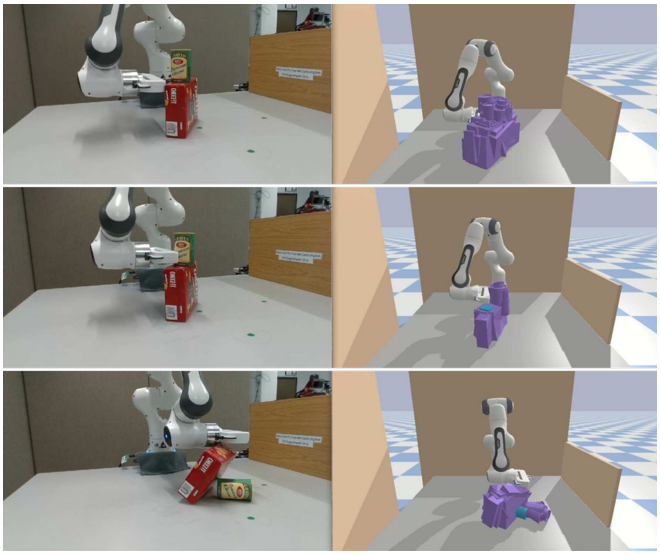


Fig. 11. An example with a high-impact drop of an object, where our system's tracking performance is degraded.

We point out that our system is not limited to planar motion. In Fig. 13, we show an example tracking result of our system, we the robot lifts an object up and then down. The results from the experiment, which involved vertical movements, demonstrated that the system can effectively track such motions, as long as they are not highly dynamic.

E. Tracking scenes with identical objects

We also tested whether PBPF-RGBD can track scenes with identical objects. A scene with two identical objects can be seen in Fig. 12-top-right, and the attached video. The results are presented in the bottom rows of Table VII, which again shows better performance for PBPF-RGBD. To work under this setting, given multiple PE system estimates of the same type of object at the same time-point, PBPF-RGBD needs to identify which particular object (of that type) in a particle these different estimates belong to. To do this, we simply assign the nearest object of the correct type in a particle to the first PE estimate. If an object has already been assigned to a previous PE estimate, we assign the next nearest one.

VIII. CONCLUSION

This work addresses the critical challenge of tracking multi-objects during robotic non-prehensile manipulation under occlusions — scenes where conventional vision-based tracking methods often fail. By combining physics-based predictions within a filtering framework, we perform relatively robust multi-object tracking even when targets are significantly occluded by obstacles, robot arms, or other target objects. Our key contribution is integrating physical prediction (robot joint states) with vision and depth information to resolve problems caused by occlusions. Moreover, we demonstrated the practical utility of our method by integrating it with a Model Predictive Control (MPC) framework. This integration facilitates tasks

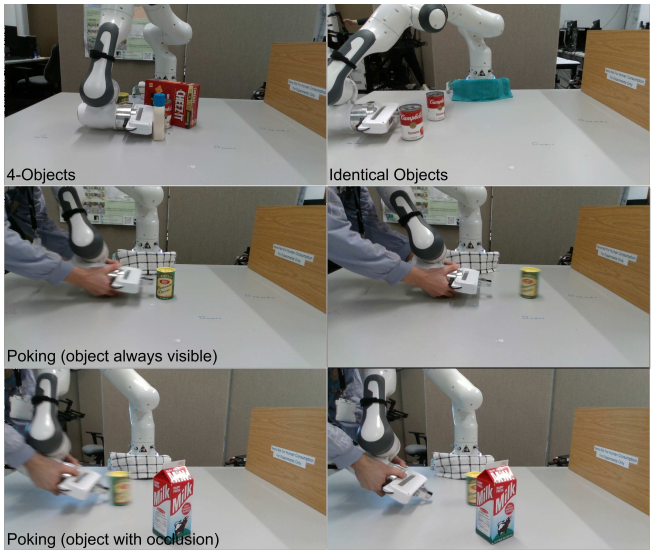


Fig. 12. Other interesting scenes. 1. The top-left shows robot manipulating 4 objects simultaneously. 2. Top-right shows robot manipulating two identical objects simultaneously. 3. The second row images show the robot poking/hitting an object to slide it away. 4. The bottom row also shows the robot poking/hitting an object to slide it away, but with occlusions to the camera. (Please also see attached video for video versions.)

such as pushing objects to target areas; a key capability for applications where robots need to retrieve items from cluttered environments like baskets or shelves. While our experiments confirm the effectiveness of the proposed method, several limitations remain. Notably, the current implementation requires an initial estimate of the objects' 6D poses for particle initialization. The reliance on a physics engine introduces considerable computational costs, which pose challenges in maintaining both high tracking accuracy and speed when scaling to a large number of objects or dealing with high-impact scenarios. Future work will focus on eliminating the dependency on precise initial pose estimates. We also investigate the use of faster physics engines (e.g., [57], [58]) to improve real-time performance without decreasing tracking precision.

APPENDIX
GPU-BASED RENDERING PIPELINE

In this section we give the details of the GPU-based rendering pipeline that we used to compute the Visibility scores for objects in the particle (described in Sec. IV-C) and to render the simulated depth images per particle (described in Sec. IV-D).

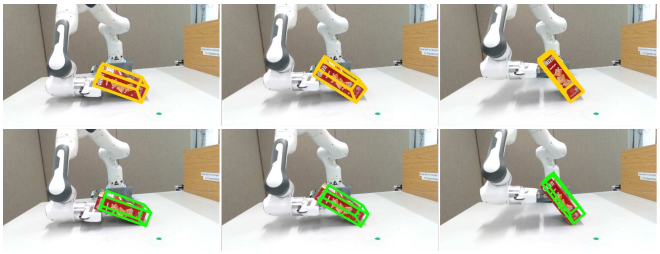


Fig. 13. Robot lifting and object up and then down. Top row: FoundationPose. Bottom row: PBPF-RGBD.

We compute error scores $e_t^{[m]}$ and visibility scores $Visible^i(x_t^{[m]})$ on the GPU, using the Vulkan cross-platform API [59]. We utilize both graphics and compute capabilities. The implementation uses four different steps: we first compute the per-object footprints (illustrated in the top right image in Fig. 4). Next, we render the depth image ${}^D\bar{z}_t^{[m]}$, which is required to compute both the pixel counts (as illustrated in the right in Fig. 4) and the error score $e_t^{[m]}$. The order enables reuse of GPU framebuffer resources, reducing memory requirements.

We use a standard projective graphics pipeline with a single 32-bit floating point depth target. This depth attachment is required for hidden surface removal ("depth testing") during rendering and ultimately holds information equivalent to the depth images ${}^D\bar{z}_t^{[m]}$. We use a Reverse-Z projection [60], [61] to maximize precision in the computed depth.

To count pixels, we utilize *occlusion queries* [62]. Occlusion queries enable us to count, in GPU hardware, the number of drawn pixels for a specific set of draw commands. Samples discarded due to hidden surface removal, i.e., samples that fail a depth test, are not counted. To count the number of pixels for each object individually (bottom three middle images in Fig. 4), we draw object $\bar{q}_t^{i,[m]}$ with the standard projection, but change the depth of the drawn pixels to a constant $d^{i,[m]} < 1$ in the vertex processing stage. We set the the depth test to only accept samples with a strictly smaller depth (`COMPARE_OP_LESS`). Depth is initialized to 1. The first sample to be drawn in a certain pixel location will pass the depth test, causing the result of the occlusion query to be incremented, and update the depth to $d^{i,[m]}$. A following sample in the same location will be discarded by the depth test, leaving the occlusion query unchanged. We draw each object in turn, but with smaller depth values ($d^{j,[m]} < d^{i,[m]}$ for $j > i$). This avoids having to clear the depth buffer for each object.

For counting the pixels of the objects in the joint image (top-left and top-right in Fig. 4), we first draw the particle's objects into a single image. This gives us ${}^D\bar{z}_t^{[m]}$. Next, each object is drawn a second time, with an occlusion query and with the depth test configured to accept samples with equal depth. The occlusion query thus returns the number of visible pixels. In practice, we use `COMPARE_OP_GREATER_OR_EQUAL`; with this we do not have to change the graphics pipeline between drawing ${}^D\bar{z}_t^{[m]}$ and performing the occlusion queries.

Finally, we calculate $e_t^{[m]}$ with a set of compute shaders. The shaders take ${}^D\bar{z}_t^{[m]}$ and a reference image ${}^Dz_t$ as input. We evaluate Equation 13's comparison per pixel and perform a parallel reduction with cooperating work groups [63], [64] to sum the values.

Only query results and the sums from the parallel reductions need to be returned to the CPU-side processing. We specifically avoid copying images from GPU VRAM to CPU RAM, as a PCIe bus is very limited when compared to available VRAM bandwidth.

REFERENCES

[1] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield, "Deep object pose estimation for semantic robotic grasping of household objects," in *Conference on Robot Learning*, PMLR, 2018.

[2] B. Wen, W. Yang, J. Kautz, and S. Birchfield, "Foundationpose: Unified 6d pose estimation and tracking of novel objects," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

[3] N. Correll, K. E. Bekris, D. Berenson, *et al.*, "Analysis and observations from the first amazon picking challenge," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 1, pp. 172–188, 2016.

[4] C. Eppner, S. Höfer, R. Jonschkowski, *et al.*, "Lessons from the amazon picking challenge: Four aspects of building robotic systems.," in *Robotics: science and systems*, vol. 12, 2016.

[5] Y. Sun, J. Falco, M. A. Roa, and B. Calli, "Research challenges and progress in robotic grasping and manipulation competitions," *IEEE robotics and automation letters*, vol. 7, no. 2, pp. 874–881, 2021.

[6] D. Müller, N. Y. Wettengel, and D. Paulus, "Homer@ unikoblenz: Winning team of the robocup virtual@ home open platform league 2021," in *Robot World Cup*, Springer, 2021, pp. 283–290.

[7] F. Ruggiero, V. Lippiello, and B. Siciliano, "Nonprehensile dynamic manipulation: A survey," *RA-L*, 2018.

[8] J. Zhou, Y. Hou, and M. T. Mason, "Pushing revisited: Differential flatness, trajectory planning, and stabilization," *The International Journal of Robotics Research*, vol. 38, no. 12-13, pp. 1477–1489, 2019.

[9] F. R. Hogan and A. Rodriguez, "Reactive planar nonprehensile manipulation with hybrid model predictive control," *The International Journal of Robotics Research*, vol. 39, no. 7, pp. 755–773, 2020.

[10] N. Kitaev, I. Mordatch, S. Patil, and P. Abbeel, "Physics-based trajectory optimization for grasping in cluttered environments," in *ICRA*, 2015.

[11] R. Papallas, A. G. Cohn, and M. R. Dogar, "Online replanning with human-in-the-loop for non-prehensile manipulation in clutter—a trajectory optimization based approach," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5377–5384, 2020.

[12] A. Pasricha, Y.-S. Tung, B. Hayes, and A. Roncone, "Pokerrt: Poking as a skill and failure recovery tactic for planar non-prehensile manipulation," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, 2022.

[13] B. Huang, S. D. Han, J. Yu, and A. Boularias, "Visual foresight trees for object retrieval from clutter with nonprehensile rearrangement," *RA-L*, 2021.

[14] S. Thrun, W. Burgard, and D. Fox, "Probabilistic robotics," 2005.

[15] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning (2016–2019)," *URL http://pybullet. org*,

[16] T. Hodan, M. Sundermeyer, Y. Labbe, *et al.*, "Bop challenge 2023 on detection segmentation and pose estimation of seen and unseen rigid objects," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 5610–5619.

[17] B. Liu, Y. Zhu, C. Gao, *et al.*, "Libero: Benchmarking knowledge transfer for lifelong robot learning," *arXiv preprint arXiv:2306.03310*, 2023.

[18] A. Mandlekar, D. Xu, J. Wong, *et al.*, "What matters in learning from offline human demonstrations for robot manipulation," in *Conference on Robot Learning (CoRL)*, 2021.

[19] O. Mees, L. Hermann, E. Rosete-Beas, and W. Burgard, "Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks," *IEEE Robotics and Automation Letters (RA-L)*, vol. 7, no. 3, pp. 7327–7334, 2022.

[20] A. Mandlekar, S. Nasiriany, B. Wen, *et al.*, "Mimicgen: A data generation system for scalable robot learning using human demonstrations," in *7th Annual Conference on Robot Learning*, 2023.

[21] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Benchmarking in manipulation research: Using the yale-cmu-berkeley object and model set," *IEEE Robotics & Automation Magazine*, vol. 22, no. 3, pp. 36–52, 2015.

[22] S. Tyree, J. Tremblay, T. To, *et al.*, "6-dof pose estimation of household objects for robotic manipulation: An accessible dataset and benchmark," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2022, pp. 13 081–13 088.

[23] Z. Xu, R. Papallas, and M. R. Dogar, "Physics-based object 6d-pose estimation during non-prehensile manipulation," in *International Symposium on Experimental Robotics*, Springer, 2023, pp. 181–191.

[24] F. Wang and K. Hauser, "Dense robotic packing of irregular and novel 3d objects," *IEEE Transactions on Robotics*, vol. 38, no. 2, pp. 1160–1173, 2022.

[25] R. Shome, W. N. Tang, C. Song, *et al.*, "Tight robot packing in the real world: A complete manipulation pipeline with robust primitives," *arXiv preprint arXiv:1903.00984*, 2019.

[26] C. Mitash, A. Boularias, and K. E. Bekris, "Robust 6d object pose estimation with stochastic congruent sets," in *29th British Machine Vision Conference*, 2019.

[27] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab, "Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again," in *IEEE international conference on computer vision*, 2017.

[28] Y. Li, G. Wang, X. Ji, Y. Xiang, and D. Fox, "Deepim: Deep iterative matching for 6d pose estimation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 683–698.

[29] J. Tremblay, B. Wen, V. Blukis, B. Sundaralingam, S. Tyree, and S. Birchfield, "Diff-dope: Differentiable deep object pose estimation," *arXiv preprint arXiv:2310.00463*, 2023.

[30] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "Posecnn: A convolutional neural network for 6d object

pose estimation in cluttered scenes," *Robotics: Science and Systems XIV*, 2018.

[31] B. Wen, C. Mitash, B. Ren, and K. E. Bekris, "Se (3)-tracknet: Data-driven 6d pose tracking by calibrating image residuals in synthetic domains," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2020.

[32] B. Wen and K. Bekris, "Bundletrack: 6d pose tracking for novel objects without instance or category-level 3d models," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2021.

[33] L. Naik, T. M. Iversen, A. Kramberger, J. Wilm, and N. Krüger, "Multi-view object pose distribution tracking for pre-grasp planning on mobile robots," in *International Conference on Robotics and Automation*, 2022.

[34] K. Pauwels and D. Kragic, "Simtrack: A simulation-based framework for scalable real-time object pose detection and tracking," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015.

[35] K. Pauwels, V. Ivan, E. Ros, and S. Vijayakumar, "Real-time object pose recognition and tracking with an imprecisely calibrated moving rgb-d camera," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2014, pp. 2733–2740.

[36] X. Deng, A. Mousavian, Y. Xiang, F. Xia, T. Bretl, and D. Fox, "Poserbpf: A rao–blackwellized particle filter for 6-d object pose tracking," *IEEE Transactions on Robotics*, vol. 37, no. 5, pp. 1328–1342, 2021.

[37] R. Ge and G. Loianno, "Vipose: Real-time visual-inertial 6d object pose tracking," in *IEEE/RSJ International Conf. on Intelligent Robots and Systems*, 2021.

[38] S. Lu, R. Wang, Y. Miao, C. Mitash, and K. Bekris, "Online object model reconstruction and reuse for lifelong improvement of robot manipulation," in *International Conference on Robotics and Automation*, 2022.

[39] C. Choi and H. I. Christensen, "Rgb-d object tracking: A particle filter approach on gpu," in *IEEE/RSJ International Conf. on Intelligent Robots and Systems*, 2013.

[40] M. Stoiber, M. Elsayed, A. E. Reichert, F. Steidle, D. Lee, and R. Triebel, "Fusing visual appearance and geometry for multi-modality 6dof object tracking," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, pp. 1170–1177.

[41] C. Mitash, A. Boularias, and K. Bekris, "Physics-based scene-level reasoning for object pose estimation in clutter," *The International Journal of Robotics Research*, vol. 41, no. 6, pp. 615–636, 2022.

[42] T. Schmidt, K. Hertkorn, R. Newcombe, Z. Marton, M. Suppa, and D. Fox, "Depth-based tracking with physical constraints for robot manipulation," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 119–126.

[43] M. Jongeneel, A. Bernardino, N. van de Wouw, and A. Saccon, "Model-based 6d visual object tracking with impact collision models," in *2022 American Control Conference (ACC)*, IEEE, 2022, pp. 3850–3856.

[44] R. K. Kandukuri, M. Strecke, and J. Stueckler, "Physics-based rigid body object tracking and friction filtering from rgb-d videos," in *2024 International Conference on 3D Vision (3DV)*, IEEE, 2024, pp. 1259–1269.

[45] T. Mörwald, M. Kopicki, R. Stolkin, *et al.*, "Predicting the unobservable visual 3d tracking with a probabilistic motion model," in *IEEE International Conference on Robotics and Automation*, 2011.

[46] M. Pfanne, M. Chalon, F. Stulp, and A. Albu-Schäffer, "Fusing joint measurements and visual features for in-hand object pose estimation," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3497–3504, 2018.

[47] M. Wüthrich, P. Pastor, M. Kalakrishnan, J. Bohg, and S. Schaal, "Probabilistic object tracking using a range camera," in *IROS*, 2013.

[48] K. Eckenhoff, P. Geneva, N. Merrill, and G. Huang, "Schmidt-ekf-based visual-inertial moving object tracking," in *IEEE International Conference on Robotics and Automation*, 2020.

[49] J. Wang and Y. He, "Motion prediction in visual object tracking," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2020.

[50] D. T.-H. Nguyen and V.-H. Nguyen, "Application of visual servo in tracking and grasping moving target," in *2022 International Conference on Control, Robotics and Informatics (ICCRI)*, IEEE, 2022, pp. 83–87.

[51] A. J. Davison, "Active search for real-time vision," in *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, vol. 1, 2005, pp. 66–73.

[52] G. Lee, J.-S. Kim, S. Kim, and K. Kim, "6d object pose estimation using a particle filter with better initialization," *IEEE Access*, vol. 11, 2023.

[53] R. Douc and O. Cappé, "Comparison of resampling schemes for particle filtering," in *4th Intl. Symposium on Image and Signal Processing and Analysis*, 2005.

[54] T. Howell, N. Gileadi, S. Tunyasuvunakool, K. Zakka, T. Erez, and Y. Tassa, "Predictive sampling: Real-time behaviour synthesis with mujoco," *arXiv preprint arXiv:2212.00541*, 2022.

[55] "Motion capture systems." (1996), [Online]. Available: https://optitrack.com (visited on 08/06/2024).

[56] R. D. Howe and M. R. Cutkosky, "Practical force-motion models for sliding manipulation," *The International Journal of Robotics Research*, vol. 15, no. 6, pp. 557–572, 1996.

[57] V. Makoviychuk, L. Wawrzyniak, Y. Guo, *et al.*, "Isaac gym: High performance gpu-based physics simulation for robot learning," *arXiv preprint arXiv:2108.10470*, 2021.

[58] *Genesis: A universal and generative physics engine for robotics and beyond*, Dec. 2024. [Online]. Available: https://github.com/Genesis-Embodied-AI/Genesis.

[59] "Vulkan®." (2024), [Online]. Available: https://vulkan.org (visited on 08/14/2024).

[60] E. Lapidous and G. Jiao, "Optimal depth buffer for low-cost graphics hardware," in *ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, 1999.

[61] "Visualizing depth precision." (2024), [Online]. Available: https://developer.nvidia.com/blog/visualizing-depth-precision/ (visited on 08/14/2024).
[62] "Vulkan® 1.3.292 - a specification." (2024), [Online]. Available: https://registry.khronos.org/vulkan/specs/1.3-extensions/html/vkspec.html%5C#queries-occlusion (visited on 08/14/2024).
[63] W. D. Hillis and G. L. Steele Jr, "Data parallel algorithms," *Communications of the ACM*, vol. 29, no. 12, pp. 1170–1183, 1986.
[64] M. Billeter, O. Olsson, and U. Assarsson, "Efficient stream compaction on wide simd many-core architectures," in *Proceedings of the conference on high performance graphics 2009*, 2009, pp. 159–166.

Zisong Xu received the B.Eng. degree from the China University of Petroleum (Huadong), Qingdao, China in 2019, and M.Eng. degree from the University of Leeds, Leeds, U.K. in 2020. He is currently working toward the Ph.D. degree in robotic manipulation with the University of Leeds, Leeds, U.K. His research interests include robotic manipulation, robotic perception, and computer vision.

Rafael Papallas received his Ph.D. in Robotics and AI from the School of Computer Science at the University of Leeds in 2021. He is currently an Assistant Professor in Computer Science at the American University of Beirut — Mediterraneo in Paphos, Cyprus, and a Visiting Research Fellow at the School of Computer Science, University of Leeds. He works on intelligent robotics and particularly focuses on motion planning for robot arm manipulators. His research interests include motion planning, motion control, perception and human-robot interaction.

Jaina Modisett is a Ph.D. candidate at the University of Leeds, where she received her Master of Science in High-Performance Graphics and Games Engineering in 2023. She is currently working on high-resolution voxel geometry techniques, focusing on the real-time rendering of and interaction with surface representations. Her research interests include real-time rendering, voxel geometry representations, and data structures and algorithms.

Markus Billeter received his Ph.D. in computer science and engineering (computer graphics) from the Chalmers University of Technology (Gothenburg, Sweden) in 2014 after completing a MSc in Engineering Physics (Complex Adaptive Systems) in 2008. He is a lecturer at the School of Computer Science (University of Leeds, UK) and specializes in real-time rendering and efficient GPU data structures and methods.

Mehmet R. Dogar received his PhD from the Robotics Institute at the Carnegie Mellon University in 2013. He is currently a Professor in Robotics and AI at the School of Computer Science, University of Leeds. He was a Visiting Professor at ETH Zurich in 2023 and a post-doctoral researcher at CSAIL, MIT between 2013-2015. Prof. Dogar is an Associate Editor for IEEE Transactions on Robotics and also for the International Journal of Robotics Research. His research focuses on robotic object manipulation.