Real-Time Physics-Based Object Pose Tracking during Contact-Based Manipulation

Zisong Xu, Rafael Papallas and Mehmet Dogar

***Abstract*— We propose a method to track the 6D pose of an object over time, while the object is under contact-based manipulation by a robot. At any given time during the manipulation of the object, we assume access to the robot joint controls and an image from a camera. We use the robot joint controls to perform a physics-based prediction of how the object might be moving. We then combine this prediction with the observation coming from the camera, to estimate the object pose as accurately as possible. Our results show that making physics-based predictions is worth the computational cost, resulting in more accurate tracking, and estimating object pose even when the object is not clearly visible to the camera.**

I. INTRODUCTION

In this paper, we propose a method to track the pose of an object over time. In our setting, the object is under contact-based manipulation by a robot. For example, in Fig. 1, we show the estimated pose of an object over time, as it is pushed by a robot. Particularly, our method combines the information coming from the robot joints (i.e. controls) with the information coming from a camera, to perform the tracking.

Traditionally, estimating the pose of an object from a single (RGB or depth (D)) camera image has been the dominant approach in robotic manipulation applications [1]. This is partly due to prehensile (i.e. grasping-based, pick-and-place) manipulation being the most common manipulation method investigated in the literature. Object pose can be estimated from a camera image using deep-learning-based (e.g. DOPE [2]) or feature-based [3] methods.

However, in the robotic manipulation community, there is increasing interest in moving beyond prehensile manipulation and performing contact-based actions such as pushing, toppling, and rolling [4]. Often, contact-based manipulation requires tracking the object continuously as the object pose changes while it is being manipulated.

There are other previous works that take a Bayesian filtering approach to object pose estimation. To estimate the pose of the object at the current time-step t, these systems use a new observation (i.e. a new camera image) that arrives at time t, *and* the previous estimated pose of the object at time $t-1$. Relating the previous pose to the current pose requires estimating how the object (or camera) might have moved in between these time-steps [5].

However, during manipulation, we often have access to the controls executed by the robot. This provides information about how the object might have moved from one time-step to the next. Therefore, in this paper, we propose to use robot control information, and to combine it with visual information, to track the object pose. At every time-step, we use the robot control information to make predictions about object motion using a physics engine. Independently, we use the visual information from a camera to perform object pose estimation (particularly, we use DOPE [2]). We then combine these two predictions in a particle filtering framework [6].

Authors are with the School of Computing, University of Leeds, UK. `{sc19zx, r.papallas, m.r.dogar}@leeds.ac.uk`

II. PROBLEM FORMULATION

We use $x_t \in SE(3)$ to represent the full 6D pose of the object at time t. Our objective is to estimate x_t for all times t during manipulation, i.e. to track the object pose as it is being manipulated. At any time t, we assume we have access to:

- the controls u_t, which are the joint-space controls executed by the robot since the last time-step $t-1$;
- the observation z_t, which is an image from a camera looking at the scene.

Therefore, our problem can be formalized as: At any time t, given all controls since the beginning of the manipulation $\{u_0, u_1, \dots, u_t\}$ and all observations $\{z_0, z_1, \dots, z_t\}$, predict an estimate of the object pose, $\tilde{x}_t$.

III. PHYSICS-BASED PARTICLE FILTERING (PBPF)

Instead of treating every time-step as independent from the previous one, we propose to use a Bayesian filtering approach [6]. In this approach, at each time-step, we estimate and update the probability distribution for x_t given all the previous controls and observations, $p(x_t \mid z_0, z_1, ..., z_t, u_0, u_1, ..., u_t)$, which is sometimes also called the *belief state*.

In particle filtering [6], the belief state at time t is represented using a set of *particles* sampled from this distribution:

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

where each particle $x_t^{[m]}$ $(1 \leq m \leq M)$ is a concrete instance of the state at time t. In our setting, each particle represents a possible pose of the manipulated object.

During particle filtering, at each time-step t, the previous set of particles $\mathcal{X}_{t-1}$ are updated using the current controls, u_t, and observation, z_t, to generate a new set of particles $\mathcal{X}_t$. This happens in two stages: the *motion update* (presented in Sec. III-A), and the *observation update* (in Sec. III-B).

A. Motion update

During this first stage, for each particle $x_{t-1}^{[m]}$, we generate a new intermediate particle (shown as $\bar{x}_t^{[m]}$), by sampling:

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

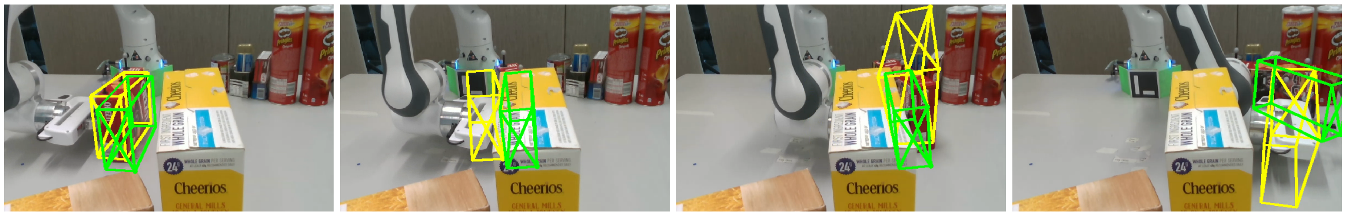
Fig. 1. Robot pushing a Cheezit object (red box) among cluttering objects. Images are from the tracking camera. Estimated object pose of our proposed method (PBPF) is shown as green wireframe box. The estimated object pose from a vision-only pose estimation system (DOPE) is shown as a yellow wireframe box. (Please note wireframe boxes are overlayed on the images as post-processing. While the green box appears on top of the obstacle/robot hand in some images, this is only an artefact of the way we overlay these wireframe boxes. The pose estimates are in fact behind the obstacle/robot hand in these instances, close to the actual object pose. Please also see the accompanying video.)

The probability distribution $p(x_t \,|\, x_{t-1}^{[m]}, u_t)$ is called the *motion model*, and ideally it represents our uncertainty about the object's resulting pose, if the object started at pose $x_{t-1}^{[m]}$ and the robot moved with u_t. While we do not have direct access to this distribution, we estimate via a physics engine.

We assume access to a physics engine, represented as f:

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

The physics engine includes a model of the robot, the environment, and the object, and predicts the resulting pose of the object x_t, given a previous state, x_{t-1}, and robot control, u_t, by simulating the robot motion inside the engine, and finding the resulting motion of the object. Here, θ refers to physical parameters that affect the result of the physics engine, e.g. friction coefficient at the contacts, restitution of the contacts, the mass of the object etc. The physics engine is deterministic (i.e. outputs the same resulting state, if given the same inputs and parameters), and therefore cannot directly be used instead of the probabilistic motion model. However, we note that, our uncertainty about the object motion is due to two sources: (i) our uncertainty about the exact physical parameters of the object, θ; and (ii) the discrepancy between the physics engine and the real world physics. Therefore, to address (i), we approximate the sampling from the motion model $\bar{x}_t^{[m]} \sim p(x_t \,|\, x_{t-1}^{[m]}, u_t)$, by first sampling θ from a distribution representing our uncertainty about the physical parameters of the object:

$$\theta_t^{[m]} \sim \mathcal{N}(\mu_\theta, \sigma_\theta^2) \tag{4}$$

and then running the physics engine with the sampled parameter:

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

with the addition of Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma_f^2)$ to address (ii) above. (In Eqs. 5, 6 and 7 we make a slight abuse of notation by using the addition/subtraction operator over poses. In actual implementation, we use quaternion algebra to add/subtract the rotational components.)

$$\mathrm{d}x = \tilde{x}_{t-1} - \tilde{x}_{t-2} \tag{6}$$

$$\bar{x}_t^{[m]} = x_{t-1}^{[m]} + \mathrm{d}x + \epsilon \tag{7}$$

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

In this work, we limit the physical parameters we sample, θ, to the coefficients of friction, the restitution parameters, and the mass of the object. However, uncertainty about other parameters (e.g. object shape, inertial parameters, robot hand shape, imperfections of the ground) can also be represented and integrated into this framework. In the above, $\mathcal{N}(\mu, \sigma^2)$ represents a normal (Gaussian) distribution[1]. The parameter μ_θ represents our best guess about the parameters θ, and σ_θ^2 represents our uncertainty (variance). The parameter σ_f^2 represents our estimated discrepancy between the physics engine and real world physics.

B. Observation update

During this second stage of particle filtering [6], for each intermediate particle $\bar{x}_t^{[m]}$, we calculate an *importance factor* $w_t^{[m]}$ using the observation z_t:

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

After the importance factor for all the intermediate particles are computed, they are used to *re-sample* the new set of particles $\mathcal{X}_t$, completing the particle filter update. During re-sampling, each intermediate particle $\bar{x}_t^{[m]}$ can be chosen (possibly multiple times) to be added to the new particle set $\mathcal{X}_t$, with a probability proportional to $w_t^{[m]}$.

The expression $p(z_t \,|\, \bar{x}_t^{[m]})$ in Eq. 8 is called the *observation model*. In our setting, it ideally represents the probability of making the current observation (i.e. getting the current camera image) if the object was at pose $\bar{x}_t^{[m]}$. Since we do not have access to such a model directly, we again propose to use an approximation. Using the Bayes Theorem, we first re-write the observation model:

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

Here, we note that $p(z_t)$ is the same for every particle, since the current observation does not change between particles.

[1]Later in Eq. 11, 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.

Furthermore, we make a simplifying assumption that $p(\bar{x}_t^{[m]})$ are also similar for different particles. This assumption enables us to compute the importance factor using:

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

To compute $p(\bar{x}_t^{[m]}|z_t)$, we propose to use a single-snaphot pose estimation system to predict the pose of the object according to z_t, and then to use the distance of $\bar{x}_t^{[m]}$ to this predicted pose to compute a probability value. As the single-snapshot pose estimation system, we again use DOPE, but other pose estimation methods can also be used. Using a notation similar to Eq. 12 where $DOPE(z_t)$ is the object pose predicted by DOPE given the camera image z_t, we compute the importance factor as:

$$p(\bar{x}_t^{[m]}|z_t) = \mathcal{N}(\bar{x}_t^{[m]};\, DOPE(z_t), \sigma^2_{DOPE}) \tag{11}$$

$$\tilde{x}_t = DOPE(z_t) \tag{12}$$

where the parameter σ^2_{DOPE} represents the variance of DOPE errors for the object. This can be estimated beforehand for an object by collecting DOPE estimates for the object and comparing it to a ground truth pose.

C. Calculating $\tilde{x}_t$

The particle filter keeps track of all particles $\mathcal{X}_t$ through the duration of manipulation. However, if a single estimate $\tilde{x}_t$ is required at any time t, then a statistic from the particles can be computed. In this work, we use the mean of all the particles in $\mathcal{X}_t$ to compute $\tilde{x}_t$. To find the mean of 3D rotations , we use the method in Markley et al. [7].

D. Computational cost

We update the particle filter at regular time intervals, Δt. Since the physics-based predictions, i.e. our motion update, is the most computationally expensive part of the filter, we determine Δt as the smallest time duration within which we can perform the physics simulations for all particles.

This also puts a limit on the number of particles that can be used. In general, more particles lead to better tracking accuracy, and one question is whether the physics-based predictions are worth the computational cost. Our second baseline method, presented next, aims to test this.

IV. Experiments and Results

We evaluated and compared the performance of our methods in seven different contact-based manipulation scenes. We used two objects from the YCB dataset [8], the CheezIt box and the Campbell Soup, a cylinder. DOPE had good performance of these objects when they were visible.

Cheezit Scene 1 (shown in Fig. 1) and **Soup Scene 1** (second row of Fig. 2): The robot pushes the object among clutter, where the cluttering object obstructs the view of the camera. The pose of the obstructing object is fixed and known.

Cheezit Scene 2 (first row of Fig. 2) and **Soup Scene 2** (third row of Fig. 2): The robot pushes the object on a clear table. The hand can obstruct the camera view during pushing.

TABLE I. Mean and standard deviation of the positional and rotational errors of different algorithms

		Scene 1				Scene 2			
		pos.error(m)		rot.error(rad)		pos.error(m)		rot.error(rad)	
		μ	σ	μ	σ	μ	σ	μ	σ
Cheezit	PBPF	**0.08**	0.02	**0.48**	0.27	**0.07**	0.03	**0.42**	0.30
	DOPE	0.28	0.21	1.34	1.16	0.29	0.24	1.04	1.02
Soup	PBPF	**0.04**	0.02	**1.13**	0.36	**0.05**	0.03	0.85	0.24
	DOPE	0.21	0.11	1.17	0.37	0.20	0.11	**0.77**	0.22

In Scenes 1 and 2, obstacles in the clutter do not physically interact with the robot and the tracked object.

A. Implementation of Methods

We have implemented our method as below[2].

PBPF: The physics-based particle filtering method (Sec.III). (a) *Motion model parameters.* μ_θ and σ_θ: Mean friction coefficient of 0.1 and standard deviation of 0.3, with minimum capped at 0.001. Mean restitution of 0.9 and standard deviation of 0.2. Mean mass of 0.38 kg and standard deviation of 0.5 with minimum capped at 0.05 kg. σ_f: For position 0.005 m, for rotation 0.05 rad. As the physics model, f_θ, we used the pybullet physics engine [9]. A pybullet environment was initialized for each particle. The pybullet environments for particles were parallelized over the 8 (16 virtual) CPU cores of the computer. (b) *Observation model parameters.* σ_{DOPE}: For position 0.02 m and rotation 0.09 rad. (c) *Update frequency.* $\Delta t = 0.16s$. (d) *Number of particles.* $M = 70$. (e) *Initialization.* We use a Gaussian distribution to initialize 70 particles at $t = 0$. The mean pose is estimated by DOPE at $t = 0$. The standard dev. for initialization are 0.16 m and 0.43 rad.

Ground truth: We used a marker-based OptiTrack system to record "ground truth" poses, x_t^*, during manipulation.

B. Experimental procedure

In each scene, we repeated 10 robot runs, (i.e. a total of 70 real robot executions). During each run, we recorded the robot controls, u_t, and the camera images, z_t. Since particle filtering is a sampling-based method, it can generate different results with the same input. For statistical accuracy, we evaluated PBPF 10 times on the data from each of the 10 runs in a given scene, giving 100 evaluations in each scene.

At each time-step of each evaluation, we computed (as in Sec. II) the positional and rotational errors of PBPF, using the ground truth. When computing errors for DOPE, if DOPE did not output any object pose at a certain time-step (usually because of occlusions), we used the last pose reported by DOPE before that time-step.

C. Results

We present the overall (averaged over all runs and all time-steps) positional and rotational errors in Table. I.

[2]Experiments were performed on CPU: 11th Gen Intel(R) Core(TM) i9-11900@2.50GHz; GPU: NVIDIA GeForce RTX 3090 ; RAM: 128580 Mb

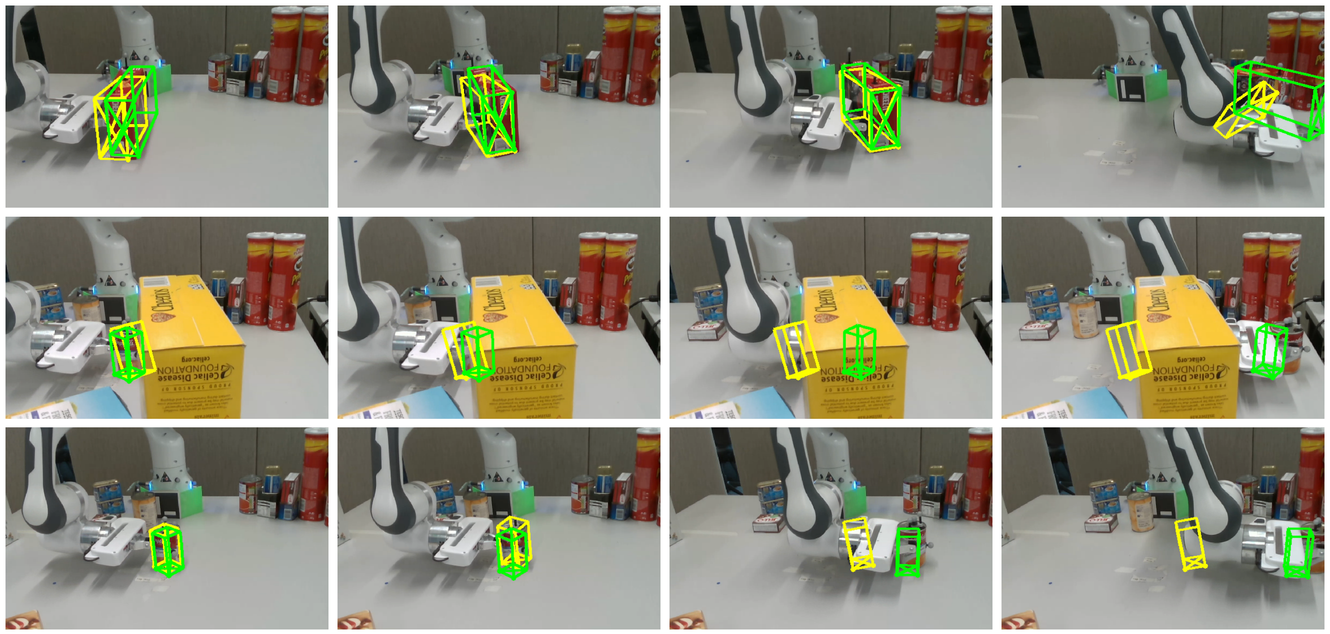

Fig. 2. Experimental Scenes. Shown images are from the tracking camera. Estimated object pose of our proposed method (PBPF) is shown as green wireframe box. The estimated object pose from DOPE is shown as a yellow wireframe box. Videos of all scenes are at github.com/ZisongXu/trackObjectWithPF.

Compared to DOPE, PBPF performs significantly better in Scenes 1 and 2. Please see videos of experiments and results in our repository: github.com/ZisongXu/trackObjectWithPF.

V. Future Work

Our goal is to generalize this method to track multiple objects simultaneously, so that a robot, pushing on a group of contacting objects, can track the pose of all the pushed objects. This is particularly a problem for vision systems, as it is difficult to see individual objects, when multiple objects are pushed as a group/pile. This however requires tracking a much more high-dimensional state.

References

[1] N. Correll, K. E. Bekris, D. Berenson, *et al.*, "Analysis and observations from the first amazon picking challenge," *T-ASE*, 2016.

[2] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield, "Deep object pose estimation for semantic robotic grasping of household objects," *arXiv preprint*, 2018.

[3] A. Collet, M. Martinez, and S. S. Srinivasa, "The moped framework: Object recognition and pose estimation for manipulation," *IJRR*, 2011.

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

[5] C. Choi and H. I. Christensen, "Rgb-d object tracking: A particle filter approach on gpu," in *IROS*, 2013.

[6] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT Press, 2005.

[7] F. L. Markley, Y. Cheng, J. L. Crassidis, and Y. Oshman, "Averaging quaternions," *Journal of Guidance, Control, and Dynamics*, 2007.

[8] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Benchmarking in manipulation research: The ycb object and model set and benchmarking protocols," *arXiv preprint*, 2015.

[9] E. Coumans and Y. Bai, *Pybullet, a python module for physics simulation for games, robotics and machine learning*, `http://pybullet.org`, 2016-2019.