VI. Appendix

A. Object Pose Predictor Architecture

The architecture of our pose predictor is shown in Fig. 8. The pose predictor first extracts a 128-dimensional feature from the history buffer, and then the look-ahead time is appended to this feature, passed together through another fully-connected network to get the final predicted pose. We choose not to use recurrent architectures such as long short-term memory (LSTM) to achieve faster inference time, such that we can compute the whole predicted trajectory for the predictable range in parallel. We also tried a Transformer [47] architecture, but it does not outperform the current version based on fully-connected networks. We reach the same conclusion as stated in [48], which highlights that the Transformer architecture may not outperform in time-series forecasting tasks.

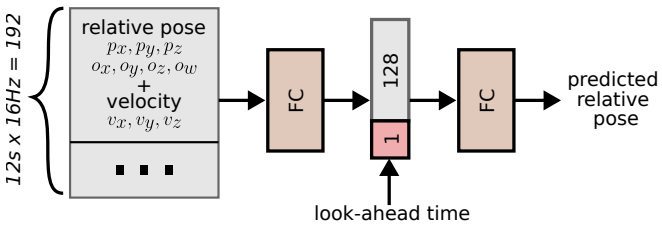

Fig. 8. **Object pose predictor architecture.** The sequence of past poses and velocities is sampled from the past 12 seconds at 16Hz. The first FC (fully-connected) network extracts a 128-dimensional feature, and the second FC network generates the predicted pose from the extracted feature and look-ahead time. All poses are relative to the previous pose.

B. Trajectories and Objects

How the 4 types of trajectories are parameterized and the set of selected objects are shown in Fig. 9. θ specifies the counter-clockwise angle of the trajectory, r is the distance from the trajectory to the robot arm base, l is the length of the trajectory (for rectangular and sinusoidal trajectories, l is the straight distance from start to end), and $d \in \{+1, -1\}$ indicates the direction of the motion, where $+1$ means counter-clockwise and -1 means clockwise. The speed v of the motion is randomly sampled but remains constant through an episode.

We select this set of objects to include both concave and convex shapes and also to ensure that the size of the object fits in the gripper. The sampling range of each trajectory parameter is shown in Table II. We design these ranges such that at least part of the trajectory is reachable for the robot arm, assuming no obstacles.

Trajectory	θ	r (m)	l (m)	d	v (cm/s)
Linear	[0°, 360°]	[0.35, 0.65]	1	$\{+1, -1\}$	[2, 6]
Sinusoidal	[0°, 360°]	[0.35, 0.65]	1	$\{+1, -1\}$	[2, 6]
Rectangular	[0°, 360°]	[0.35, 0.65]	1	$\{+1, -1\}$	[2, 6]
Circular	[0°, 360°]	[0.65, 0.9]	1.5	$\{+1, -1\}$	[5, 10]

TABLE II

PARAMETER SAMPLING RANGE FOR EACH CONVEYOR TRAJECTORY.

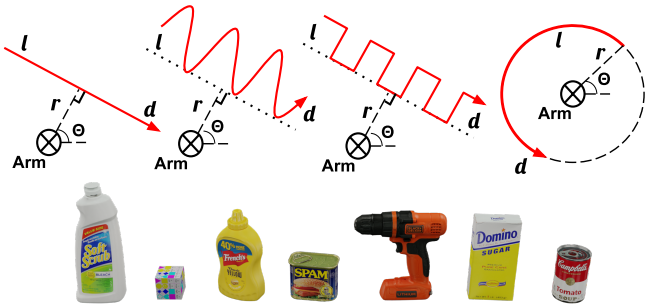

Fig. 9. **Randomized trajectories and selected graspable objects.** Top Row: A bird's eye view of linear, sinusoidal, rectangular, and circular conveyor belt motion generation process. Each random motion is parameterized by angle θ, distance r, length l, and direction d. The cross indicates the position of the robot base. The red line shows the motion of the conveyor belt, with an arrow indicating the direction. The horizontal dashed line at the robot base indicates the positive x-axis of the world frame. Bottom row: 7 objects from the YCB dataset are selected as the graspable target objects for our experiments.

C. Justification on Sim2Real Transfer

We are not able to provide real-world experiments. However, we think that the sim2real gap of this work is minimal, and the performance gain from the meta-controller will be consistent regardless of the sim2real gap. There are several reasons for it. (1) The largest gap when transferring to the real robot is the vision system that estimates the object poses and bounding box dimensions. Recent perception systems such as DOPE [49] and Gen6D [50] can obtain such information fast and reliably. We also add significant Gaussian noises to the poses and bounding box dimensions to account for such perception errors. Given the same magnitude, pure random noise is more challenging than noise from realistic perception systems with fixed observation-to-noise mapping. (2) Given the estimated poses and bounding box dimensions, there is no sim2real gap in other components of our pipeline, such as object pose predictor, grasp planner, motion planner, and the meta-controller. (3) Our simulated environment is under realistic timing. Time spent in all submodules is compensated with the target object motion and we retime the planned arm motion to make sure it matches the real-robot speed.