Appendix for iGibson 2.0: Object-Centric Simulation for Robot Learning of Everyday Household Tasks

A.1 The iGibson 2.0 Virtual Reality Interface

In this section, we provide additional information about the implementation of our virtual reality (VR) interface in iGibson 2.0.

Mapping human motion to the virtual embodiment: Humans in VR control a bimanual embodiment in iGibson 2.0 composed of a head, two hands and a torso/body. The hardware for VR control is composed of a headset and two hand controllers, possibly an additional tracker for the human torso that maps directly to the VR body. At each step, the pose of the agent's head is directly set to be the new headset's pose as provided by the VR hardware. This step bypasses physics simulation of the head motion to make sure that the motion of the head in VR corresponds exactly and without delays to that in the real world to avoid any discomfort. In contrast, both the hands and the main body move as the result of physical simulation of a body constraint connecting the simulated hands and body to the new poses provided by the VR hardware. In other words, the VR hardware provides at each step new "desired" poses for the hands and body that "pull" the body parts towards them with forces computed by the simulator. This creates realistic interactions between the hands and body and the objects in the scene, colliding with them and applying forces. The constraint to move the hand has a maximal force of $300\,\mathrm{N}$, simulating the lifting force of humans. The constraint to move the body is $50\,\mathrm{N}$. While the desired new poses for the hands always come from the hand-controllers, the desired pose for the body can be provided by a body tracker on the human's torso or be otherwise estimated based on the headset position. This constraint-based system strikes a balance between accurately following the human motion and realistic interactions with the scene in VR.

Haptic feedback: To approximate the real-world experience, it is important to create haptic feedback to human subjects when interacting with the scene. To that end, in our VR interface collisions of the body and the hands trigger haptic vibrations in the controllers. The body will trigger a strong vibration in both controllers facilitating navigation in the scene. The hands also generate low-strength vibration when they are in contact with an object, to notify users that they are in contact with a virtual object. These mechanisms create a multi-modal stream to the humans (vision and haptics) that help them interact more dexterously and realistically.

Assistive Grasping: Creating realistic, robust and dexterous grasping in virtual reality is challenging. Grasping objects in real-world involves generating multiple frictional contact points and surfaces between the hand and the object: simulating physically this complex process in a realistic manner is non-trivial. Additionally, real-world grasping involves rich multimodal signals that include tactile and haptic information, that is not available in common virtual reality interfaces. To compensate for these differences and generate a natural experience in simulation, we implement an assistive grasping mechanism.

In our VR interface, grasping is performed by pressing the left or right trigger, a one degree of freedom (DoF) actuation. This single DoF is mapped to a closing motion of the avatar's hand where all fingers move synchronously. If the trigger is pressed more than 50% of its range, we activate the *assistive grasping* (AG) mode. The AG mode facilitates grasping by creating an additional fixed or point-to-point joint between the hand and the movable objects inside the hand. We use three criteria to decide what object inside the hand should be assisted for grasping: First, the object has to be inside the hand. We evaluate this we a ray casting mechanism. Rays are shot between the following two sets of points: (thumb tip, thumb middle, palm middle, palm base) and (4 non-thumb finger tips). These 16 rays return a list of all objects that are in the hand. Second, the object has to be close to the palm. This is defined as the distance between the center of mass of the palm link and the object is the smallest among all objects in hand. And third, the object needs to be in contact with the hand and the hand has to be applying force on it. We found that, defined with these three criteria, the AG mechanism is realistic as humans can grasp objects reliably with motions that are close to the ones used in real-world for the same objects. While a user is grasping an object using AG, collision of the hand with that object is also disabled, to avoid any recurring collisions or simulation instability.

The AG breaks the connection between the object and the hand if the grasping trigger goes under 50% pressed, or the center of mass of object being grasped moves further away from the palm than a maximum threshold, D_{AGMax}. We use $D_{AGMax} = 10\,\mathrm{cm}$. This effectively avoids that humans can use AG to grasp objects or pull from them in an unrealistic manner, e.g., grasping a heavy object

such a watermelon with a single hand. Our AG system strikes a good balance between facilitating grasping and interactions of humans in VR and simplifying the manipulation of objects excessively. In this way, humans in VR still need to move and position the hands in realistic ways to grasp and manipulate heavy objects. The AG mode is also critical to grasp small objects with dexterity. In the VR demonstrations created by the human, we can see that the AG mechanism works robustly for all kind of rigid and articulated objects, ranging from fridge handles to knives, from watermelons to strawberries. Please refer to the supplementary video to see examples of AG mechanism.

Hardware compatibility: We have tested our iGibson 2.0 with the three main commercially available headsets at the time, HTC Vive (Pro Eye), Oculus Rift S and Oculus Quest. For the former, we include functionalities to observe and record the eye gaze tracked by the device. To serve both devices, the iGibson renderer needs to output 1296 x 1440 images at 30 Hz, shown to the human to create a immersive 3D experience. The system is capable of rendering at up to 90 Hz, but we had to reduce the frame rate to accommodate additional expensive per-frame computations, including physics and extended states update steps in our simulator.

Logging and replaying demonstrations: All information of the runs can be logged and replay deterministically (same output for the same actions). The logged information includes kinematics and extended iGibson 2.0 states, and VR agent actions. We believe the logged information acquired with the iGibson 2.0 VR interface will facilitate research: the information can be analyzed to understand human strategies, or used with modern robot learning techniques, e.g. with imitation learning, to train embodied AI solutions.

A.2 Extended Object States, Logical Predicates and Generative System in iGibson 2.0

In this section, we provide additional information on iGibson 2.0's new extended physical states, logical predicate system that map physical states to logical states, and the generative system that sample valid simulated physical states based on logical states.

Extended states associated to object categories: In iGibson 2.0, not all extended states need to be maintained for instances of all object categories, e.g., most objects cannot be `Sliced` and only food objects could be `Cooked`. We assume that any object instance added to iGibson 2.0 belongs to a category annotated with properties. The properties indicate what extended states should be updated for object instances of that category. The exhaustive list of all possible object category properties is shown in Table A.2. Some properties will require additional annotation for each object model to estimate logical states.

Object model annotations: Each model needs to be annotated with additional physical and semantic information to simulate correctly interactions and their associated logical states. The exhaustive list of all possible object model properties are included in Table A.3. Some properties directly come from the 3D assets, such as `Shape` and `KinematicStructure`. We compute `Weight` based on query results from Amazon Product API, and compute `CenterOfMass` and `MomentOfInertia` accordingly for each link based on the assumption of uniform density. Note that if an object category is annotated with a certain property, e.g., `stove` is annotated as `HeatSourceSink`, we need to annotate `HeatSourceSinkLink`, a virtual (non-colliding) fixed link that heats or cools objects, for all object models of `stove`. For `SlicingTool` and `CleaningTool`, we additionally annotate `SlicingToolLink` and `CleaningToolLink`, which are colliding fixed links that can slice objects (the blade of a knife) and remove dirt particles from objects (the bottom of a vacuum), respectively.

Updating object state: During simulation, our simulator maintains and updates not only the kinematic states of the objects, such as `Pose` and `InContactObjs`, using our underlying physics engine Bullet [17], but also the non-kinematic states, such as `Temperature` and `WetnessLevel` with custom rules. These update rules are explained in Sec. 3 and summarized in Table A.4.

Logical predicates as discriminative functions: As explained in Sec. 4, we define a set of discriminative functions that map the extended physical states to logical states that are semantically grounded on natural language, such as `Cooked` and `Sliced`. These logical states can used for symbolic planning and checking intermediate success for sub-tasks for reinforcement learning. The details of the discriminative functions of all the logical predicates can be found in Table A.1.

Logical predicates as generative functions: We also define a set of sampling functions that can generate valid physical states that satisfy the given logical states. For example, if the initial conditions of the task require a book placed `OnTopOf` a table or a shelf being `Stained`, our system can

automatically sample concrete physical states that satisfy the requirements: sampling a random position on the table and place the book there, and sample stain particles on random locations on the shelf (see Fig. 5a). The details of the generative functions of all the logical predicates can be found in Table A.5. Please also refer to our supplementary video for more details.

Sampling extended states based on the given logical predicates is relatively simple, e.g., sampling a temperature that corresponds to an object being `Frozen`. However, sampling object poses to fulfill the given kinematic predicates is more involved as it requires sampling values in the Special Euclidean group SE(3) with additional constraints such as placing objects in stable configurations and not causing penetrations between objects. In the following, we describe our algorithm to sample valid poses based on kinematic logical predicates.

Pose Sampling Algorithm: Say we are sampling a valid pose for an object o_1 to be `OnTopOf` object o_2. First, we query the set of stable orientations allowed for object o_1. We assume these orientations are provided per object model, e.g., for a book the orientations to place the book on its cover and last page or upright. Each stable orientation is linked to an axis-aligned bounding box with an associated bounding-box base area. The next step would be to find areas on the surface of object o_2 that can hold the bounding-box area and that are flat, unobstructed, and accessible. To find these areas of o_2 surface we use a ray-casting mechanism conditioned on the specific kinematic logical predicate. For example, for our case of `OnTopOf` we will generate rays starting immediately above o_2 by sampling points from the top face of its axis-aligned bounding box, and marching downwards in the vertical direction. The points where the rays intersect o_2 surface will be used to define planes where we can attempt to sample object o_1 if they fulfill some criteria such as providing stable support. We can repeat the procedure for different stable orientations of o_1. Other logical predicates use a similar generative procedure but with variations in the ray-tracing step. For example, for `InsideOf`, we start our rays at different points inside the o_2 bounding box rather than above it. In addition, for particle-based states such as `Dusty` and `Stained`, we additionally allow casting rays in horizontal directions.

Predicate	Description
`InsideOf`(o_1,o_2)	Object o_1 is inside of object o_2 if we can find two orthogonal axes crossing at o_1 center of mass that intersect o_2 collision mesh in both directions.
`OnTopOf`(o_1,o_2)	Object o_1 is on top of object o_2 if $o_2 \in$ `InSameNegativeVerticalAxisObjs`$(o_1) \wedge o_2 \notin$ `InSamePositiveVerticalAxisObjs`$(o_1) \wedge$ `InContactWith`(o_1, o_2), where `InSamePositive/NegativeVerticalAxisObjs`(o_1) is the list of objects in the same positive/negative vertical axis as o_1 and `InContactWith`(o_1, o_2) is whether the two objects are in physical contact.
`NextTo`(o_1,o_2)	Object o_1 is next to object o_2 if $o_2 \in$ `InSameHorizontalPlaneObjs`$(o_1) \wedge l2(o_1, o_2) < t_{NextTo}$, where `InSameHorizontalPlaneObjs`$(o_1)$ is a list of objects in the same horizontal plane as o_1, $l2$ is the L2 distance between the closest points of the two objects, and t_{NextTo} is a distance threshold that is proportional to the average size of the two objects.
`InContactWith`(o_1,o_2)	Object o_1 is in contact with o_2 if their surfaces are in contact in at least one point, i.e., $o_2 \in$ `InContactObjs`(o_1).
`Under`(o_1,o_2)	Object o_1 is under object o_2 if $o_2 \in$ `InSamePositiveVerticalAxisObjs`$(o_1) \wedge o_2 \notin$ `InSameNegativeVerticalAxisObjs`(o_1).
`OnFloor`(o_1,o_2)	Object o_1 is on the room floor o_2 if `InContactWith`(o_1, o_2) and o_2 is of `Room` type.
`Open`(o)	Any joints (internal articulated degrees of freedom) of object o are open. Only joints that are relevant to consider an object `Open` are used in the predicate computation, e.g. the door of a microwave but not the buttons and controls. To select the relevant joints, object models of categories that can be `Open` undergo an additional annotation that produces a `RelevantJoints` list. A joint is considered open if its joint state q is 5% over the lower limit, i.e. $q > 0.05(q_{UpperLimit} - q_{LowerLimit}) + q_{LowerLimit}$.
`Cooked`(o)	The temperature of object o was over the cooked threshold, T_{cooked}, and under the burnt threshold, T_{burnt}, at least once in the history of the simulation episode, i.e., $T_{cooked} \leq T_o^{max} < T_{burnt}$. We annotate the cooked temperature T_{cooked} for each object category that can be `Cooked`.
`Burnt`(o)	The temperature of object o was over the burnt threshold, T_{burnt}, at least once in the history of the simulation episode, i.e., $T_o^{max} \geq T_{burnt}$. We annotate the burnt temperature T_{burnt} for each object category that can be `Burnt`.
`Frozen`(o)	The temperature of object o is under the freezing threshold, T_{frozen}, i.e., $T_o \leq T_{frozen}$. We assume as default freezing temperature $T_{frozen} = 0^\circ C$, a value that can be adapted per object category and model.
`Soaked`(o)	The wetness level w of the object o is over a threshold, w_{soaked}, i.e., $w \geq w_{soaked}$. The default value for the threshold is $w_{soaked} = 1$, (the object is soaked if it absorbs one or more droplets), a value that can be adapted per object category and model.
`Dusty`(o)	The dustiness level d of the object o is over a threshold, d_{dusty}, i.e., $d > d_{dusty}$. The default value for the threshold is $d_{dusty} = 0.5$, (half of the dust particles have been cleaned), a value that can be adapted per object category and model.
`Stained`(o)	The stain level s of the object o is over a threshold, $s_{stained}$, i.e., $s > s_{stained}$. The default value for the threshold is $s_{stained} = 0.5$, (half of the stain particles have been cleaned), a value that can be adapted per object category and model.
`ToggledOn`(o)	Object o is toggled on or off. It is a direct query of the iGibson 2.0 objects' extended state TS, the toggled state.
`Sliced`(o)	Object o is sliced or not. It is a direct access of the iGibson 2.0 objects' extended state SS, the sliced state.
`InFoVOfAgent`(o)	Object o is in the field of view of the agent, i.e., at least one pixel of the image acquired by the agent's onboard sensors corresponds to the surface of o.
`InHandOfAgent`(o)	Object o is grasped by the agent's hands (i.e. assistive grasping is activated on that object).
`InReachOfAgent`(o)	Object o is within $d_{reach} = 2$ meters away from the agent.
`InSameRoomAsAgent`(o)	Object o is located in the same room as the agent.

Table A.1: **Logical Predicates:** Description of the discriminative functions

Property of an object category	Required extended object states
Can be `Cooked`	`MaxTemperature`, `Temperature`
Can be `Burnt`	`MaxTemperature`, `Temperature`
Can be `Frozen`	`Temperature`
Can be `Soaked`	`WetnessLevel`
Can be `Dusty`	`DustinessLevel`
Can be `Stained`	`StainLevel`
Can be `ToggledOn`	`ToggledState`
Can be `Sliced`	`SlicedState`
Is a `HeatSourceSink`	`ToggledState`
Is a `DropletSource`	`ToggledState`

Table A.2: Extended states associated to properties of object categories

Object Model Property	Must be defined if the object category...	Description
`Shape`		Model of the 3D shape of each link of the object
`Weight`		Weight of the object
`CenterOfMass`		Mean position of the matter in the object
`MomentOfInertia`		Resistance of the object to change its angular velocity
`KinematicStructure`		Structure of links and joints connecting them in the form of URDF (non-articulated objects are composed of one link)
`StableOrientations`		A list of stable orientations assuming the object is placed on a flat surface, computed using a 3D geometry library
`HeatSourceSinkLink`	Is a `HeatSourceSink`	Virtual (non-colliding) fixed link that generates/absorbs heat
`CleaningToolLink`	Is a `CleaningTool`	Fixed link that needs to contact dirt particles for the tool to clean them
`DropletSourceLink`	Is a `DropletSource`	Virtual (non-colliding) fixed link that generates droplets
`DropletSinkLink`	Is a `DropletSink`	Virtual (non-colliding) fixed link that absorbs droplets
`TogglingLink`	Can be `ToggledOn`	Virtual (non-colliding) fixed link that changes the toggled state of the object when contacted
`SlicingLink`	Is a `SlicingTool`	Fixed link that changes the sliced state of another object if it contacts it with enough force
`RelevantJoints`	Can be `Open`	List of joints that are relevant to indicate whether an object is open

Table A.3: Non-updatable object model properties (annotated)

Object State	Description and Update Rules
`Pose`	6 DoF pose (position and orientation) of the object in world reference frame, updated by the underlying physics engine.
`AABB`	Axis-aligned bounding box (coordinates of two opposite corners) of the object in the world reference frame, updated by the underlying physics engine.
`JointStates`	State of all internal DoFs of the (articulated) object for the structure defined by `KinematicStructure`, updated by the underlying physics engine.
`InContactObjs`	List of all objects in physical contact with the object, updated by the underlying physics engine.
`InSamePositiveVerticalAxisObjs`	List of all objects in the positive vertical axis drawn from the object's center of mass, updated by shooting a ray upwards in the positive z-axis and gather the objects hit by the ray.
`InSameNegativeVerticalAxisObjs`	List of all objects in the negative vertical axis drawn from the object's center of mass, updated by shooting a ray downwards in the negative z-axis and gather the objects hit by the ray.
`InSameHorizontalPlaneObjs`	List of all objects in the horizontal plane drawn from the object's center of mass, updated by shooting a number of ray in the x-y plane and gather the objects hit by the rays.
`Temperature`, T	Object's current temperature in $^\circ$C, updated by detecting if the object is affected by any heat source or heat sink.
`MaxTemperature`, T_{max}	Maximum temperature of the object reached historically during this simulation run, updated by keeping track of all the `Temperature` in the history.
`WetnessLevel`, w	Amount of liquid absorbed by the object corresponding to the number of liquid droplets contacted, updated by detecting if the object is in contact with any liquid droplets.
`DustinessLevel`, d	Fraction of the initial amount of dust particles that remain on the object's surface, updated by detecting if the particles are in contact with any `CleaningTool`.
`StainLevel`, s	Fraction of the initial amount of stain particles that remain on the object's surface, updated by detecting if the particles are in contact with any `Soaked` `CleaningTool`.
`ToggledState`, *TS*	Binary state indicating if the object is currently on or off, updated by detecting if the agent is in contact with the `TogglingLink`.
`SlicedState`, *SS*	Binary state indicating whether the object has been sliced (irreversible), updated by detecting if the object is in contact with any `CleaningTool` that exerts a force above a certain threshold F_{sliced}. We assume as default force threshold of $F_{sliced} = 10$ N, a value that can be configured per object category and model.

Table A.4: Object states maintained by iGibson 2.0

Predicate	Sampling Mechanism
`InsideOf(`o_1`,`o_2`)`	Only `InsideOf(`o_1`,`o_2`) = True` can be sampled. o_1 is randomly sampled within o_2 using the mechanism described in Pose Sampling Algorithm in Sec. A.2. o_1 is guaranteed to be supported fully by a surface and free of collisions with any other object except o_2.
`OnTopOf(`o_1`,`o_2`)`	Only `OnTopOf(`o_1`,`o_2`) = True` can be sampled. o_1 is randomly sampled on top of o_2 using the mechanism described in Pose Sampling Algorithm in Sec. A.2. o_1 is guaranteed to be supported fully by a surface and free of collisions with any other object except o_2.
`NextTo(`o_1`,`o_2`)`	Not supported at the moment.
`InContactWith(`o_1`,`o_2`)`	Not supported at the moment.
`Under(`o_1`,`o_2`)`	Only `Under(`o_1`,`o_2`) = True` can be sampled. o_1 is randomly sampled on top of the floor region beneath o_2 using the mechanism described in Pose Sampling Algorithm in Sec. A.2. o_1 is guaranteed to be supported fully by a surface and free of collisions with any other object except the floor.
`OnFloor(`o_1`,`o_2`)`	Only `OnFloor(`o_1`,`o_2`) = True` can be sampled. o_1 is randomly sampled on top of o_2, which is the floor of a certain room, using the scene's room segmentation mask. o_1 is guaranteed to be supported fully by a surface and free of collisions with any other object except o_2.
`Open(o)`	To sample an object o with the predicate `Open(o) = True`, a subset of the object's relevant joints (using the `RelevantJoints` model property) are selected, and each selected joint is moved to a uniformly random position between the openness threshold and the joint's upper limit. To sample an object o with the predicate `Open(o) = False`, all of the object's relevant joints (using the `RelevantJoints` model property) are moved to a uniformly random position between the joint's lower limit and the openness threshold.
`Cooked(o)`	To sample an object o with the predicate `Cooked(o) = True`, the object's `MaxTemperature` is updated to $\max(T_o^{max}, T_{cooked})$. Similarly, to sample an object o with the predicate `Cooked(o) = False`, the object's `MaxTemperature` is updated to $\min(T_o^{max}, T_{cooked} - 1)$.
`Burnt(o)`	To sample an object o with the predicate `Burnt(o) = True`, the object's `MaxTemperature` is updated to $\max(T_o^{max}, T_{burnt})$. Similarly, to sample an object o with the predicate `Cooked(o) = False`, the object's `MaxTemperature` is updated to $\min(T_o^{max}, T_{burnt} - 1)$.
`Frozen(o)`	To sample an object o with the predicate `Frozen(o) = True`, the object's `Temperature` is updated to a uniformly random temperature between $T_{frozen} - 10$ and $T_{frozen} - 50$. To sample an object o with the predicate `Frozen(o) = False`, the object's `Temperature` is updated to $T_{frozen} + 1$.
`Soaked(o)`	To sample an object o with the predicate `Soaked(o) = True`, the object's `WetnessLevel` w is updated to match the `Soaked` threshold of w_{soaked}. To sample an object o with the predicate `Soaked(o) = False`, the object's `WetnessLevel` w is updated to 0.
`Dusty(o)`	To sample an object with `Dusty(o) = True`, a fixed number (currently 20) of dust particles are randomly placed on the surface of o using the mechanism described in Pose Sampling Algorithm in Sec. A.2. To sample an object with `Dusty(o) = False`, all dust particles on the object are removed.
`Stained(o)`	To sample an object with `Stained(o) = True`, a fixed number (currently 20) of stain particles are randomly placed on the surface of o using the mechanism described in Pose Sampling ALgorithm in Sec. A.2. To sample an object with `Stained(o) = False`, all stain particles on the object are removed.
`ToggledOn(o)`	The `ToggledState` of the object is updated to match the required predicate value.
`Sliced(o)`	The `SlicedState` of the object is updated to match the required predicate value. Also, the whole object are replaced with the two halves, that will be placed at the same location and inherit the extended states from the whole object (e.g. `Temperature`).

Table A.5: **Logical Predicates:** Description of the generative functions

A.3 Experimental Setup and Additional Results

In this section we provide the experimental setup for the reinforcement learning and imitation learning experiments described in Sec. 6.

Reinforcement Learning Experiments with Bimanual Humanoid Robot: The observation space include 128×128 RGB-D images from the onboard sensor on the agent's head, and proprioceptive information (hand poses in agent's local frame, and a fraction indicating how much each hand is closed). The action space is 6-dimensional representing the desired linear and angular velocities of the right hand, where the rest of the agent is stationary. For grasping, we adopt the "sticky mitten" simplification from other works [11]: we create a fixed constraint between the hand and the object as soon as they get in contact.

The agent receives a one-time success reward if it satisfies the single predicate (e.g. `Cooked(meat)`). Additionally, we provide distance-based reward shaping for each experiment to encourage the hand to approach activity-relevant objects, e.g. encourage the hand to approach the meat and the meat to approach to stove. Finally, for the Cleaning Stained Shelf task, we provide partial progress reward for each stain particle that has been cleaned. The episode terminates if the agent achieves success or times out (200 timesteps, or equivalently 20 seconds). We train for 10K episodes, evaluate on the same setups, and report the results. The training reward curves can be found in Fig. A.1.

We use Soft Actor-Critic [39] for training. The policy network has two encoders for RGB-D images and proprioceptive information. With RGB-D images as input, we use a 3-layer convolutional neural network to encode the image into a 256 dimensional vector. The proprioceptive information is encoded into a 256 dimensional vector with an MLP. The features are concatenated and pass through additional MLP layers to generate the action.

Reinforcement Learning Experiments with Fetch Robot: We also conducted the same RL experiments with a Fetch robot. The observation space include 128×128 RGB-D images from the onboard sensor on the agent's head, and proprioceptive information (the end effector pose in agent's local frame, joint configurations, and whether the end effector is currently grasping something). The action space is 6-dimensional representing the desired linear and angular velocities of the end effector, where the rest of the agent is stationary. We experimented with both the "sticky mitten" grasping simplification (the same as Bimanual Humanoid) and without such simplification. For the later setup, the Fetch robot has to rely on the friction between the gripper fingers and the objects to grasp them with realistic physics simulation. Its action space also includes one additional DoF for closing the gripper. With this later setup, we hope to minimize the sim2real gap as much as possible. Due to the additional complexity of grasping, we add one more reward shaping terms to encourage the gripper to grasp the task-relevant object and penalize the agent for dropping it. We use different reward scaling. The termination conditions remain the same. We also use the same policy network architecture and training schema as before. The training reward curves can be found in Fig. A.2.

Imitation Learning Experiments with Bimanual Humaniod Robot : The observation space includes the ground truth poses of the task-relevant objects, and proprioceptive information, the same as the RL setup. In the *Bimanual Pick and Place* experiment, task relevant objects include the cauldron, the table and the agent itself. The agent can control both of its hands with the desired linear and angular velocities, which result in 12 degree of freedom. The hand closing action is not learned, but replayed from the human demonstrations.

We collected $6500+$ state-action pairs from 30 human demos and used behavior cloning to predict the action based on the state. The policy network has two MLP encoders for proprioceptive information and ground truth object poses. The features are concatenated and pass through additional MLP layers to generate the action.

The network is trained until validation loss plateaus, and evaluated on a test set of demonstrations. As discussed in the main paper, the entire task is long horizon (>300 steps) and the policy diverges due to covariate shift [41]. We then evaluate if the policy can successfully perform the task if we initialize the simulator a few seconds before task completion of a successful human demo. The success rate with respect to different policy starting time can be found in Fig. 5b. We show a successful sequence after rewinding 2 seconds in Fig. A.3.

A.4 Performance Benchmark of iGibson 2.0

To evaluate whether iGibson 2.0 can be used in computationally expensive embodied AI research, we benchmarked the performance (simulation time) and compared with the previous version. The benchmark setup is the same as in Shen et al. [1], which considered an "idle" setup, in which we place a robot (a TurtleBot model) in the scene and run the physics simulation and extended physical state simulation loop. The benchmark runs on 15 scenes, and statistics are collected. The agent applies zero actions and stays still. We use action time step of $t_a = \frac{1}{30}$s and physics time step of $t_s = \frac{1}{120}$s to be consistent with Shen et al. [1]. Both settings are benchmarked on a computer with Intel 5930k CPU and Nvidia GTX 1080 Ti GPU, in a single process setting, rendering 512×512 RGB-D images.

The simulator speed is shown in Table A.6. Although we added many extended physical states, we still achieved a 25% increase in average performance compared with iGibson 1.0 [1]. In iGibson 2.0, the main source of speed up with respect to the previous version of iGibson is obtained from better usage of the object sleeping mechanism, and lazy update of object poses in the renderer. This allows us to simulate much larger scenes with many more objects with extended physical states tracked, and as a result, more diverse everyday household activities.

Among other simulators, Habitat 2.0 [42] is the one that has the closest capability to iGibson 2.0, with fully interactive furniture and objects. Therefore, we conducted a thorough performance benchmark against Habitat 2.0 in three scenes of different scales. These scenes contain around 20, 60, and 120 objects respectively. We strictly follow the "idle" setup in the Habitat 2.0 paper [42] with 1 GPU, 1

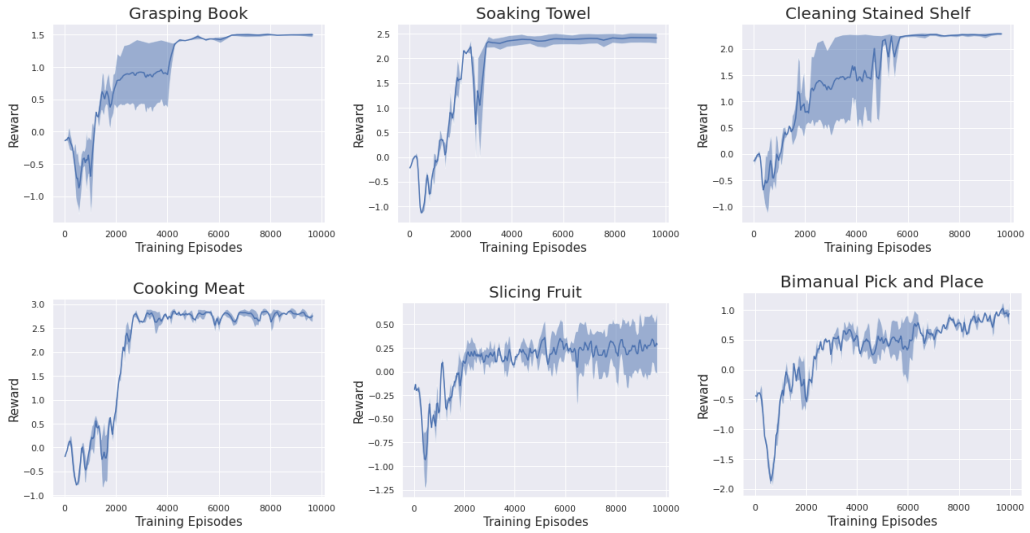

Figure A.1: **Reward curves for RL experiments (Bimanual Humanoid)**: We observe steady training progress across all the tasks and the RL agent achieve perfect reward for task 1)-4). For *Slicing Fruit*, the agent achieves only 15% success rate because the task requires a precise alignment between the knife blade and the fruit. For *Bimanual Pick and Place*, the agent fails to succeed; although it receives partial reward for approaching the cauldron, bimanual manipulation of large heavy objects requires careful coordination between hands and remains challenging learn with RL.

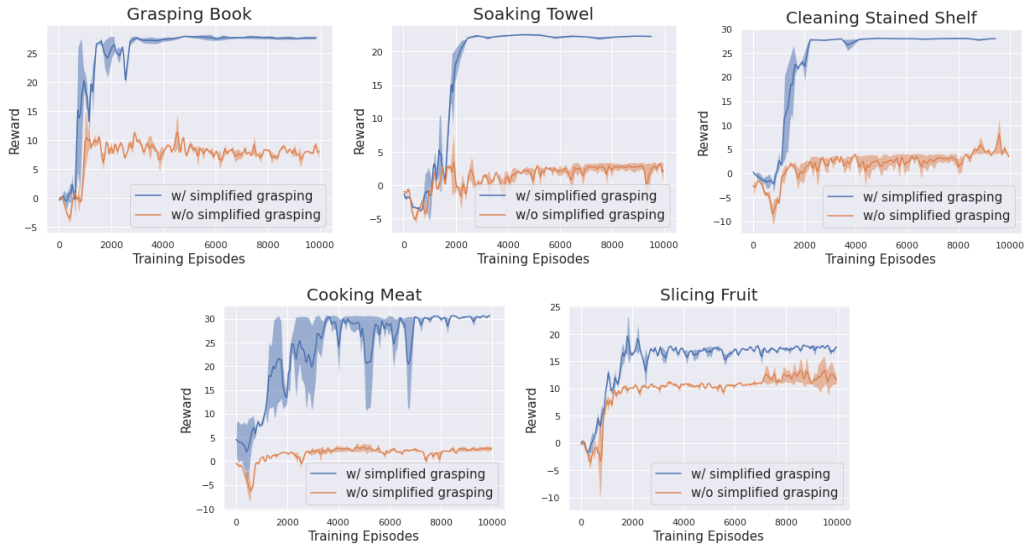

Figure A.2: **Reward curves for the five task trained with RL (Fetch)**: When we adopt the same "sticky mitten" grasping simplifcation as Bimanual Humanoid for Fetch, the RL agent achieves very similar result (blue) as the one shown in Fig. A.1. Once we remove such simplification, the RL agent, however, struggles to solve the tasks, achieving around 25% success rate for *Grasping Book* and *Soaking Towel*, and 0% for the other three. Although the robot learns to approach the task-relevant objects with its end effector, grasping a diverse set of objects (e.g. towel, knife, meat) with a parallel gripper remains a challenging robotics problem.

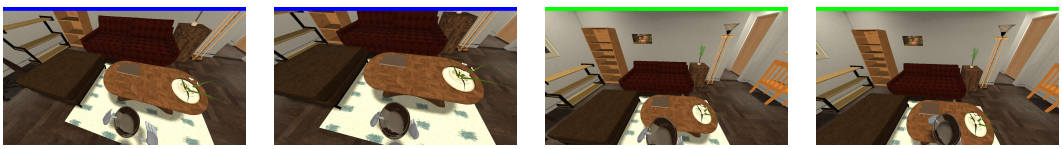

Figure A.3: **A sequence of IL execution of bimnual grasping**: the blue bar indicates that we are replaying the demonstration, and the green bar is indicating the policy is taking over. The sequence shows that we are able to complete the task of placing a heavy object on table by starting at $T - 2$s.

Simulator	Mean	Max	Min
iGibson 1.0	100	150	68
iGibson 2.0	125	217	73

Table A.6: Simulation Speed Comparison for iGibson 1.0 and iGibson 2.0. The unit is steps per second; each step simulates $\frac{1}{30}$s.

Simulator	20 Obj-scene	60 Obj-scene	120 Obj-scene
Habitat 2.0 [42]	**1595**	112	94
iGibson 2.0	425	**202**	**126**

Table A.7: Simulation Speed Comparison for Habitat 2.0 and iGibson 2.0. The unit is steps per second; each step simulates $\frac{1}{30}$s. . iGibson 2.0 has better scaling performance than Habitat 2.0 for scenes with a larger number of objects.

Process. The agent (a Fetch robot) is idle while rendering 128×128 RGB images, with all objects in the scene being physically simulated. The 20 Obj-scene is the same as the one used in Table 2 of the Habitat 2.0 paper [42]: 3 furniture object (a fridge, a table, and a kitchen counter) plus 15 YCB objects [43]. The 60 Obj-scene is the `Rs_int` scene from the iGibson 1.0 dataset [1]. The 120 Obj-scene is the `Beechwood_0_int` scene from the iGibson 1.0 dataset [1] and the `apt_0` scene from Habitat 2.0 [42] and Replica [44]. We tried our best to use comparable objects to build up comparable scenes in Habitat 2.0 and iGibson 2.0 for fair comparison. The benchmark runs on a Ubuntu machine with AMD Ryzen 7 3800X 8-core CPU, NVIDIA GeForce RTX 2080 SUPER GPU, without parallelization. As can be seen in Table A.7, while Habitat 2.0 is well optimized for scenes with a small number of objects, iGibson 2.0 has better scaling performance for scenes with a large number of objects, which is a common scenario for household tasks involving long-horizon mobile manipulation that we propose to study. In addition, Habitat 2.0 doesn't need to keep track of and update the extended physical states (e.g. temperature, wetness level, etc) at every time step like iGibson 2.0 does. Therefore, with physics simulation and rendering alone, iGibson 2.0 can achieve even higher numbers that the ones included in Table A.7.

A.5 Feature Comparisons of Simulators

In Table A.8, we provide a detailed comparison across multiple simulation environments. The table is adapted from Table I of [1]. We include more recent simulation environments as columns and more feature comparisons as rows.

Table A.8: Comparison of Simulation Environments

	iGibson 2.0 (ours)	iGibson 1.0 [1]	Gibson [2]	Habitat [5]	Sapien [4]	AI2Thor [15]	ManipulaTHOR [45]	VirtualH [16]	TDW [7, 13]
Provided Large Scenes Real-World / Designed	15 homes (108 rooms) / –	15 homes (108 rooms) / –	1400 / –	–	–	– / 120 rooms	– / 30 rooms	– / 7	– / 25
Provided Objects	1217 (*)	570	–	–	2346	609	150	308	200
Continuous Extended States	✔	✖	✖	✖	✖	✖	✖	✖	✖
Non-kinematic States	✔	✖	✖	✖	✖	✔	✖	✔	✖
Geometric Sampling Based on Logical States	✔	✖	✖	✖	✖	✔	✖	✖	✖
Human Interface	Mouse, VR	Mouse	-	Mouse	-	Mouse	Mouse	Natural Language	VR
Agent/World Interaction Forces, Predefined Actions	F	F	–	–	F	F & PA	F & PA	F & PA	F
Physics Engine	Pybullet	Pybullet	Pybullet	Bullet	PhysX	Unity	Unity	Unity	Unity & Flex
Supported Task	Nav.&Manip.	Nav.&Manip.	Nav.	Nav.	Nav.&Manip.	Nav.&Manip.	Nav.&Manip.	Nav.&Manip.	Nav.&Manip.
Speed	++	++	+	+++	++(PBR)/-(RTX)	+	+	+	+
Integrated Motion Planner	✔	✔	✖	✖	✖	✖	✖	✖	✔
Specialty	Continuous Extended States, VR	Phys. Int. in Large Scenes	Nav.	Fast, Nav.	Articulation, Ray Tracing	Object States, Task Planning	Mobile Manipulation	Object States, Task Planning	Audio, Fluids

Type of rendering: **PBR**:Physics-Based Rendering, **IBR**:Image-Based Rendering, **RTX**:Ray Tracing

Virtual sensor signals: **RGB**: Color Images, **D**:Depth, **N**:Normals, **SS**:Semantic Segmentation, **LiDAR**:Lidar, **FL**:Flow (optical and/or scene), **S**: Sounds

* included in a parallel submission BEHAVIOR and fully compatible with iGibson 2.0

A.6 Limitations and Future Work

Although iGibson 2.0 has made several significant contributions towards simulating complex, everyday household tasks for robot learning, it is not without limitation. First of all, iGibson 2.0 doesn't support soft bodies / flexible material in a scalable way at the moment, due to the limitation of our underlying physics engine. This prevents us from simulating tasks like folding laundry and making bed in large, interactive scenes. Also, iGibson 2.0 doesn't support accurate human behavior modeling (other than goal-oriented navigation), and thus prevent us from simulating tasks that are inherently rich in human-robot interaction (e.g. elderly care). With the recent advancement of physics engines, and human behavior modeling and motion synthesis, we plan to overcome these limitations in the future. In addition, we also plan to support a more diverse set of extended object states (e.g. `Filled`, `Hung`, `Assembled`, etc) as well as bi-directional transitions for some of our existing states (e.g. `Soaked` and `Stained`/`Dusty`), which can unlock even more household tasks. Finally, we plan to transfer mobile manipulation policies trained in iGibson 2.0 to the real world.