Technical Appendix for VECA

Abstract

In this **technical appendix**, we provide four supplementary details of our VECA toolkit: (1) implementation details of our VECA toolkit, (2) usage of our toolkit, (3) a number of toolkit-generated tasks apart from the VECA benchmark, and (4) thorough comparison with related works. Then we specify our comprehensive experimental setups which are omitted in the main paper due to space, e.g., hyperparameters, machine specification, formulation of metrics, and task scenario. The result plot of the fourth experiment, *Usefulness of VECA Toolkit*, is included in Figure 8.

Implementation Details of VECA toolkit

We list several implementation details of our VECA toolkit we did not describe in the main paper. We chose *Unity3D* as an engine to simulate and develop the environment, a state-of-the-art game engine equipped with realistic rendering, physics, and a user-friendly visual editor. On top of that, users can train and test python-coded agents from remote servers using a socket-based *communication interface*. Figure 1 shows the overview of the VECA environment.

Human toddler Avatar & Action. We model a human toddler agent with Blender3D software and integrate it into a Unity3D asset. Specifically, the avatar has 47 bones with 82 degrees of freedom with human joint-like hard constraints, and skin mesh represented with 1648 triangles, which adaptively changes according to the orientation of the bones. To show the physical capability of our humanoid agent, we demonstrate baby-like movements with the humanoid agent: sitting up (Figure 2a) and turning its body (Figure 2b).

However, training an agent from joint-level actions and physical interactions may not be practically plausible for complex tasks. To mitigate such burden, we also support an animation-based agent, which supports stable primitive actions (e.g., walk, rotate, crawl) and interactions (e.g., grab, open, step on) by trading off its physical plausibility. Specifically, we classify objects with their mass: For the light objects, the agent is relatively kinematic and apply the collision force only to the object. For a heavy object, the object is relatively kinematic and pushes the agent out of the collision.

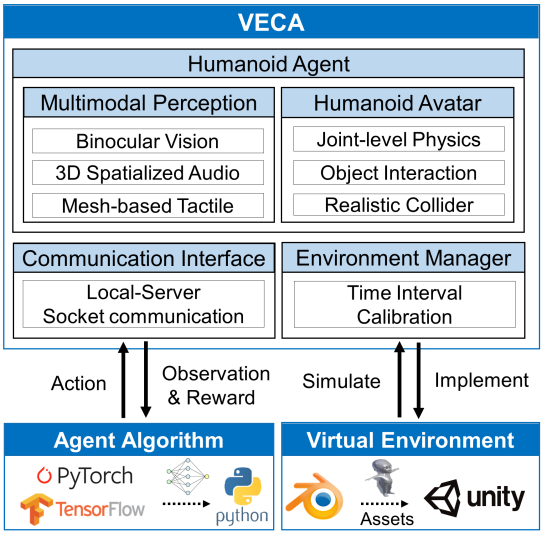


Figure 1: System overview of our VECA Environment. Unity3D engine renders human toddler-like embodied agent and immersive environment. The Environment Manager and communication interface supports the Unity3D application with diverse features. To support remote training, Communication interface enables socket-based communication of training information. Environment Manager stabilizes the time interval of unity3d engine, as well as making the envionment paralleizable.

On top of that, VECA provides an user-friendly interface where the users can implement interactions between agent and object. The object is interactive with the agent when the object is visible (rendered by agent's vision and also closer than 1.5m) and not occluded by transparent objects. When there are multiple interactable objects, the agent could choose among those objects or follow the default order (closest to the center of the viewport).

Audio. A Unity3D uses *AudioListener* class to render audio, and for spatial audio, they utilize the 3D spatialization SDK. Since Unity3D supports only one listener per scene, we found it impossible to design tasks with (i) multiple agents or (ii) multiple parallel environments in a single ap-

plication. With our custom-built audio rendering module, we allow multiple listeners in the scene and supports various effects such as 3D spatialization or reverb. We also enable the developer to listen to the agent's audio data using the audio devices for debugging purposes.

Environment Manager

We design an *Environment Manager* that manages VECA environment instances; it administers the communication of training information, e.g., collected observations and performed actions, between the Unity3D application and the agent algorithm process. The main challenge is that, in Unity3D, the time interval fluctuates between consecutive frames. This is because Unity3D focuses on providing perceptually natural scenes to the user and adapt the frame rate to available computing power. Moreover, communication latency affects the time intervals; for instance, when there is a large communication delay, the simulation time is fast-forwarded to compensate for the delay. The simplest way to solve it is to limit the users to implement the environment in *FixedUpdate()* (which is called in a fixed rate), but this approach makes *Update()* function, which consists the main life cycle of Unity3D, out of sync.

To address the problem, we separate the clock of the virtual environment from the actual time by implementing the time manager class *VECATime*, which replaces the *Time* class in Unity. Specifically, we enable VECA to simulate time-dependent features (e.g., physics, audio clip, animation) with constant time steps. For the physics, VECA adopts the physics simulator provided by Unity3D to simulate physics for the constant time step. For other time-dependent features, VECA searches for all objects with corresponding features in the environment and explicitly controls the simulation time to enforce a constant time interval.

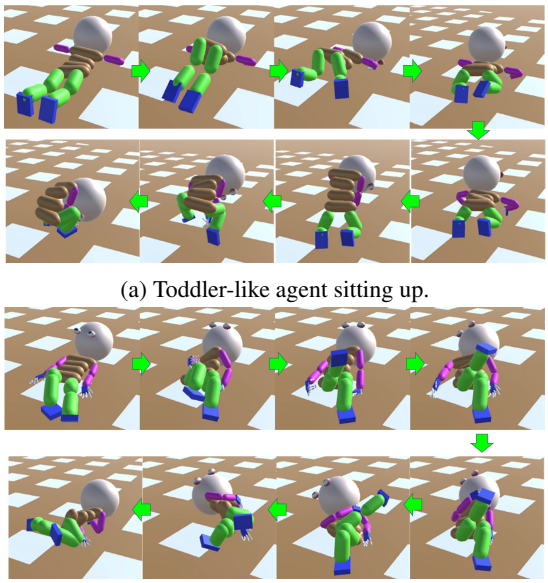
Communication Interface

This component manages the data flow between the agent algorithm (VECA python API) and the VECA environment (Unity3D application). It provides a socket-based network connection to support training on a remote server. Since Unity3D does not render visual information in devices without graphics display, the Unity3D environment should be executed in a local machine with has a monitor displays, to properly train or visualize how the training is done.

Detailed procedure of executing VECA environment is illustrated in Figure 3. (i) Environment manager daemon process is on. (ii) Agent algorithm process signals Environment manager to initiate environments. (iii) Environment manager deploys a single environment process or multiple child environment processes in parallel. (iv) Environment manager and agent algorithm process communicates observations, action and reward through network socket.

How to use VECA toolkit

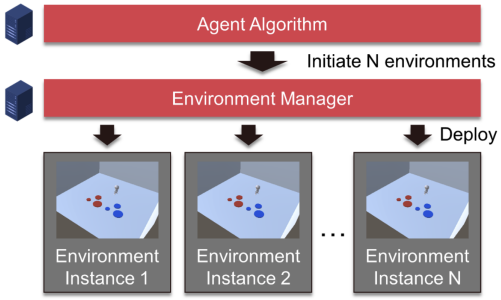
As in Figure 4, VECA toolkit can design customized tasks and object-agent interactions. First, developer needs to create a virtual environment using Unity3D editor, which is full of resources; it has GUI-based visual interfaces, prop



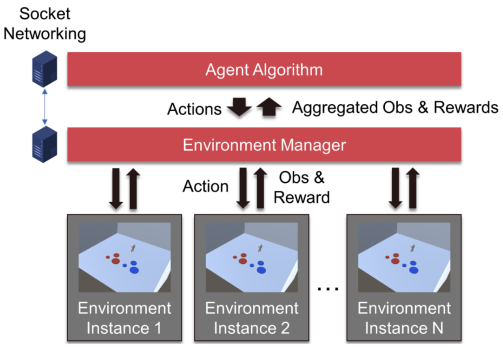
(a) Toddler-like agent sitting up.

(b) Toddler-like agent turning its body.

Figure 2: Example physical movements of a human toddler-like agent.



(a) Initiating communication.



(b) Main communication.

Figure 3: Simplified overview of how the agent algorithm communicates training information with the Environment Manager daemon, e.g., observation, reward, and action.

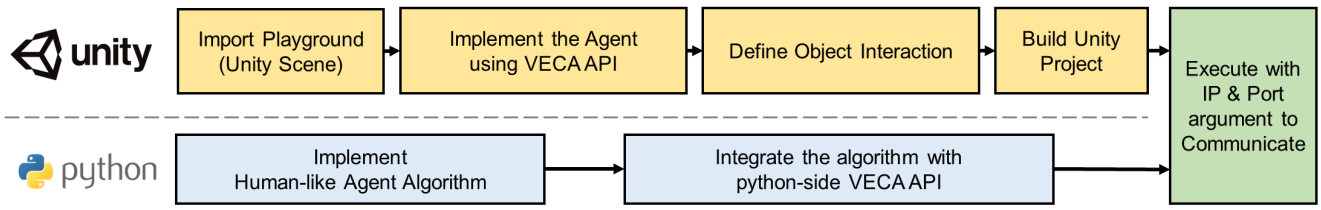


Figure 4: Procedure of using VECA toolkit to generate tasks. First, the user organize an environment using Unity3D Editor features and assets. Second, import VECA agent and define object interaction (either physical or animated). To implement the agent algorithm, the developer can use the OpenAI-Gym like python API to connect with the unity3D application of VECA environment.

assets and APIs. Second, using the VECA toolkit API, developer needs to customize three main functions (similar to ML-Agents (Juliani et al. 2018)): (1) *AgentAction* defining the action capability of a VECA agent, (2) *CollectObservation* defining the sensory input of a VECA agent, and (3) AgentReset to reset the agent. VECA toolkit already contains a series of primitive actions and object interactions that can be leveraged in designing more complicated interaction. Developers can create their own object interactions by implementing over the *VECAObjectInteract* interface. Finally, the developer builds the Unity3D application as a standalone executable. This executable can be used with VECA python API to train the agent algorithms in python. In the Table 6 and 5 below, we list VECA toolkit API functions. With figure 5, we also demonstrate that VECA python API is easy-to-use, and it is similar to widely-used OpenAI Gym's API.

```python
from veca.gym.disktower import Environment

env = Environment(port=args.algo_port, num_envs = args.num_envs)
action_dim = env.action_space
env.reset()

for i in range(100):
    action = np.random.rand(args.num_envs, action_dim) * 2 - 1
    obs, reward, done, infos = env.step(action)
    if done:
        env.reset()

env.close()
```

# of parallel envs.

(a) VECA python API.

```python
import gym
env = gym.make("CartPole-v1")
observation = env.reset()
for _ in range(1000):
  env.render()
  action = env.action_space.sample() # your agent here
  observation, reward, done, info = env.step(action)

  if done:
    observation = env.reset()
env.close()
```

(b) OpenAI Gym API.

Figure 5: Comparison of VECA Environment's API interface and OpenAI Gym's API. VECA's API is easy-to-use, similar to widely-used OpenAI Gym.

VECA Toolkit-generated Embodied AI Tasks

To show that VECA toolkit can create diverse tasks assessing distinct cognitive skills, we develop various additional tasks besides VECA benchmark that represent building blocks of human learning, as in Figure 6. We focused on four essential aspects in early human development: joint-level locomotion and control, understanding contexts of objects, multimodal learning, and multi-agent learning. Note that these categories are neither disjoint nor unique.

Joint-level Control & Dynamics. We provide a set of tasks for *joint-level control and humanoid-object interaction* with diverse objectives. Humanoid agent control has been studied for human assistance and also to understand the underlying human cognition. However, developing a human-level control is extremely challenging (Akkaya et al. 2019). Prior environments are domain-specific and difficult to integrate the human-like aspects. VECA toolkit supports joint-level dynamics down to finger knuckles, physical interactions, and tactile perception. With this capability, we build a set of challenging joint-level control tasks {TurnBaby, RunBaby, CrawlBaby, SitBaby, RotateCube, SoccerOnePlayer}. We also incorporate multimodal learning tasks (GrabObject) and tasks related to understanding objects (PileUpBlock, PileDownBlock).

Understanding Contexts of Objects. We provide various tasks {ColorSort, ShapeSort, ObjectNav, BabyZuma's Revenge, MazeNav, FillFraction} which aims to understand *contexts of objects*. Learning abstract contexts of objects is one of the key features in human learning (Cohen and Cashon 2007; Entwistle 2007), and is getting increasing attention in artificial intelligence (Vercauteren et al. 2019; Mikolov et al. 2013). Thanks to the extensibility of VECA toolkit, users can implement tasks and environments with various useful contexts, including properties (ColorSort, ShapeSort, ObjectNav), functionality (BabyZuma's Revenge), and mathematical meanings (FillFraction).

Multimodal Learning. For multimodal learning, we provide tasks {KickTheBall, ObjectPhysNav, MoveToTarget} which features in learning how to incorporate *multiple sensory inputs*. Multimodal learning plays a big role in human learning(Tacca 2011; Landau, Smith, and Jones 1998), and have also been suggested to be beneficial for training arti-

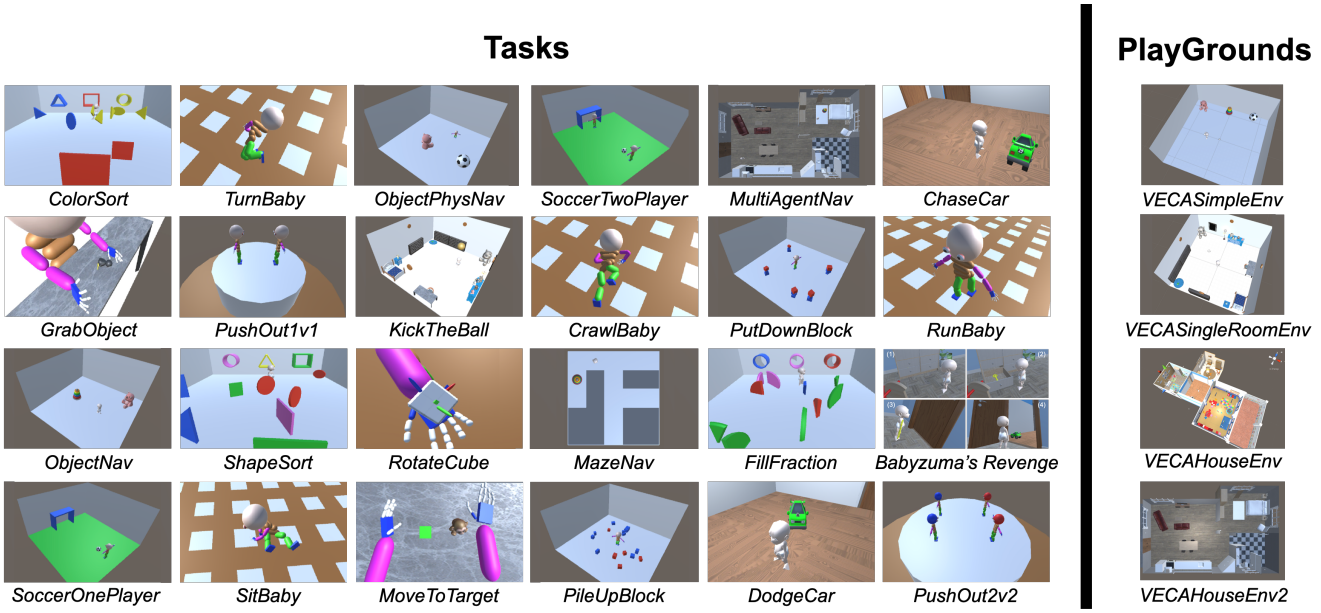


Figure 6: Diverse set of VECA toolkit-generated tasks assessing distinct cognitive aspects. Note that these tasks are additionally given; they are not a part of our VECA benchmark. Difficulty of the task is altered with playgrounds. Tasks cover: joint-level motor control {TurnBaby, RunBaby, RotateCube, e.t.c.}, multimodal learning {KickTheBall, ObjectPhysNav, MoveToTarget}, understanding context of object {ColorSort, ObjectNav, BabyZuma's Revenge, e.t.c.}, and multi-agent RL {SoccerTwoPlayer, MultiAgentNav, PushOut}.

ficial intelligence(De Vries et al. 2017; Ngiam et al. 2011). Using diverse biomemetic perceptions provided by VECA toolkit, users can develop tasks and environments for multimodal learning, such as vision-audio (KickTheBall) and vision-tactile (ObjectPhysNav, MoveToTarget) learning.

Multi-Agent Reinforcement Learning. We provide a set of multi-agent RL tasks in which agents need to *cooperate or compete with others* to solve the tasks. Multi-agent learning is critical in human development (Eckerman and Whatley 1977; Trevarthen 1979) and is a rising topic for artificial general intelligence (Lowe et al. 2017; Foerster et al. 2017). However, it is difficult to generate a multi-agent task and to integrate the multimodal perception and active interactions (including interactions between agents) to the task. With the multi-agent support of VECA toolkit, we build a competitive task (SoccerTwoPlayer), a cooperative task (MultiAgentNav), and a mixed competitive-cooperative task (PushOut). Some of the tasks are mixed; they evaluate not only the multi-agent RL but also the other aspects like multimodality (MultiAgentNav) or joint-level physics (SoccerTwoPlayer).

Playgrounds VECA toolkit provides four example playgrounds (scenes), as shown in Figure 6. Those playgrounds vary in numbers of props, furniture, and its structure. It allows controlling the complexity of tasks such as *ObjectNav, KickTheBall, MultiAgentNav*, i.e., the same tasks could be evaluated with a different complexity.

Additional Comparison with Related Works

In Table 1, we thoroughly compare our VECA toolkit and other RL environments in terms of features they support. We further contrast ours with related works other than the ones in the main paper: (1) crowdsourced datasets and (2) game-based environments.

Crowdsourced datasets. The most common approach to train an agent is to use pre-collected datasets such as *Imagenet* (Deng et al. 2009), *Audioset* (Gemmeke et al. 2017). Unlike VECA tasks, this approach requires developers to collect a large volume of data to build accurate models, which is costly and time-consuming. Like other RL environments, our VECA can train agents that learns through exploration and interaction.

Game-based environments. Recently, several game-based environments (Bellemare et al. 2013; OpenAI 2016; Beattie et al. 2016; Johnson et al. 2016; Kempka et al. 2016; Song et al. 2020) have been proposed and adopted to train and test agents with various learning algorithms (e.g., reinforcement learning (Schulman et al. 2017), imitation learning (Ho and Ermon 2016)). However, they do not apply human-like sensory features, and they do not target the evaluation of cognitive development.

Detailed Experimental Setup

We detail experimental setups for four experiments: baseline results for VECA Benchmark, validity analysis, effect of human-like perception, and usefulness of VECA toolkit.

Environments	3D	Extensible	Physics	EgoVis	Audio	Tactile	Multiagent	Interaction
ALE (Bellemare et al. 2013)	X	X	X	X	X	X	X	X
DeepMind Lab (Beattie et al. 2016)	O	△	X	O	X	X	X	X
OpenAI Universe (OpenAI 2016)	O	O	X	O	X	X	X	X
VizDoom (Kempka et al. 2016)	O	O	X	O	X	X	O	X
Arena (Song et al. 2020)	O	O	O	X	X	X	O	X
Malmo (Johnson et al. 2016)	O	O	X	O	X	X	X	O
Gibson (Xia et al. 2018)	O	X	O	O	X	X	X	X
MINOS (Savva et al. 2017)	O	X	X	O	X	X	X	X
House3D (Wu et al. 2018)	O	△	X	O	X	X	X	X
HoME (Brodeur et al. 2017)	O	△	O	O	O	X	O	X
AI2-THOR (Kolve et al. 2017)	O	O	O	O	X	X	O	O
Soundspaces (Chen et al. 2020)	O	X	O	O	O	X	X	X
VECA	O	O	O	O^+	O^+	O	O	O

Table 1: Supported features of our VECA toolkit and other RL environments. O indicates that it provides the feature, and X means it does not. O^+ indicates that it also reflects the characteristic of human sensation. **3D** : Supports 3D environment. **Extensible** : Can develop new environments and tasks. △ means it is difficult to develop new tasks due to lack of visual editor or heavily specialized API. **Physics** : Supports physics engine (e.g. collision, friction) **EgoVis** : Renders egocentric first-person point of view. **Audio** : Renders audio perception. **Tactile** : Renders tactile perception. **Multiagent** : Can deploy multiple agents in a single environment. **Interaction** : Supports complex interactions. For example in Malmo, hitting a block breaks it, but hitting a monster gives it damage.

We use a random seed 999 for the entire unity3D environment of VECA tasks. Results are obtained with single run, except the experiment 2.

We used Intel(R) Core(TM) i5-9600KF 3.70GHz with 64GB Memory for simulating the environment in Windows 10, and used Xeon Gold 5218 CPU with NVIDIA GeForce GTX 1060 6GB and 256GB RAM on a Ubuntu 16.04 for two experiments, *Effect of Human-like Perception*, and *Usefulness of VECA Toolkit*.

For the other two VECA benchmark experiments, we used Intel(R) Core(TM) i7-6700K with 32GB RAM for VECA environment. We use Xeon Gold 5218 CPU with four NVIDIA TITAN XP 12GBs and 256GB RAM on a Ubuntu 16.04 to train the agent algorithm.

Sensory Input Hyperparameters We sampled binocular RGB vision data in a resolution of 84x84. No blur or grayscaling is applied to the agent's vision. We sampled the audio data at the rate of 22050Hz and converted the audio data to frequency-domain by FFT with the window size of 1024. Mimimum audible distance $d_{TH} = 20$, absolute threshold of tactile sensory value $\delta = 0.05$, decay rate of tactile sensory value $\lambda = 0.0$. Tactile sensory input has a dimension of 3296×1.

Experiment 1: VECA Benchmark Results

Metrics. We further describe three Bayley-4 metrics used in our VECA benchmark: Total Raw Scores, Age Equivalent, and Growth Scale Value(GSV). The total raw score (TRS) is simply a sum of each task score in our VECA benchmark, as follows.

$$TRS = \sum_{i=1}^{N} (\text{score})_{T_i} \tag{1}$$

where $\mathcal{D} = \{T_i\}_{i=1,\cdots,N}$ is the VECA benchmark task set, and $(\text{score})_{T_i}$ is the score an agent attained on task T_i. This score is converted to Age Equivalent and GSV score using the score conversion table, which is listed on Figure **??**. Score conversion table is verified with a standardization research on Bayley-4 (Aylward 2017).

Experimental Setup. We adapted our VECA environment to the Torchbeast's (Küttler et al. 2019) implementation of IMPALA (Espeholt et al. 2018) and (Burda et al. 2019)'s implementation of dynamics-based curiosity-driven learning(CUR) (Pathak et al. 2017). To receive multimodal sensation, IMPALA adds two fully-connected layer-based encoders for audio and tactile sensory input. Encoder layer dimension of IMPALA is 16384×256 for audio, and 3296×256 for tactile. CUR also uses two single fully-connected layer-based encoders to the dynamics model and the policy model, for audio and tactile sensation. Encoder layer dimension of CUR is 16384×512 for audio, and 3296×512 for tactile. We used 4 actors for IMPALA and 8 actors for CUR. A 1×10^6 iteration of both CUR and IMPALA took around 7 days. We list detailed hyperparameter setup of IMPALA and CUR on Table 3. Hyperparameters are the default value of their source, except for the number of actors; we chose them in the range of $[1, 16]$ considering the training time. Tasks are randomly sampled on each episode, and previous task is not consecutively sampled. We used a random seed 999 for the VECA unity3D application.

Experiment 2: Validity Analysis

We used a seed 100 for the random policy agent. The solution probability $p_{sol}^{i=1,2}$ is calculated as,

$$p_{sol}^{i} = \frac{x_{success}^{i}}{x_{success}^{i} + x_{fail}^{i}} \tag{2}$$

Parameter	PPO	SAC
Learning rate	Adaptive to KL Div.	2.5e-4
Number of workers	8	8
λ (GAE)	0.95	N/A
Clipping ratio	0.2	N/A
Entropy coefficient	0.03	0.01
Gradient norm clipping	5	5
γ Discount factor	0.99	0.99
Optimizer	Adam	Adam
Training epoch/batch per update	4/4	N/A
Value function Coefficient	0.5	N/A
Batch size	N/A	64

Table 2: PPO and SAC algorithm hyperparameters for experiments 3 and 4. These values are chosen to increase the final performance.

Method	Hyperparameters
IMPALA	Batch size 8, Unroll length 80, Uses lstm, Entropy cost 0.0006, Discounting factor 0.99, Reward clipping, Learning rate 0.00048, Alpha 0.99, Momentum 0, Epsilon 0.01, Global gradient norm clip 40.
CUR	Batch size 8, Lambda 0.95, Gamma 0.99, Norm-adv 1, Norm-rew 1, Seed 0, Entropy coeff. 0.001, No feature learning External coeff. 0, Internal coeff. 1, Learning rate 1×10^{-4}

Table 3: IMPALA and Curiosity-drivel learning (CUR) hyperparameters for experiment 1. These are default value of the original repository of them.

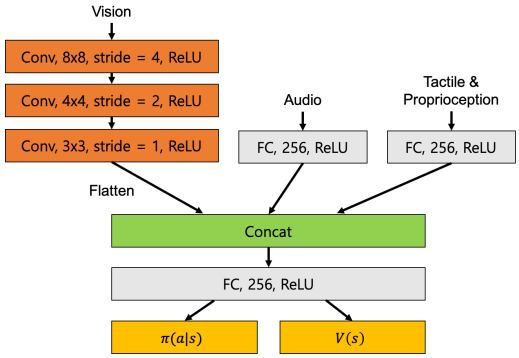


Figure 7: Policy Network Architecture for experiment 3 and 4. The model outputs the Q value instead of V value for SAC.

where $i = \{1, 2\}$ indicates the score 1 or 2, $x^i_{success}$ the number of success cases of scoring, and x^i_{fail} the number of failure cases of scoring. Likewise, the success percentage of VECA benchmark $p^{i=1,2}_{succ}$ is, for $t \in \mathbb{N}$,

$$p^i_{succ}(t) = \max\left(p^i_{succ}(t-1), \frac{n^i_s(t)}{n^i_s(t) + n^i_f(t)}\right) \quad (3)$$

where $i = \{1, 2\}$ also indicates the score 1 or 2, t the number of random attempts, $p^i_{succ}(t)$ the success percentage of VECA benchmark on t attempts, $n^i_s(t)$ the number of succeeded tasks on t-th attempt, and $x^i_f(t)$ the number of failed tasks on t-th attempt. $p^i_{succ}(0) = 0$ for both $i = 1, 2$.

For the first experiment, we tried 10^3 random attempts per task. We tried 10^6 random attempts for the second experiment, where tasks are randomly sampled on each episode.

Experiment 3: Effect of Human-like Perception

Experiment Setup. We compared the performance of the agents trained with PPO in various setting of auditory and tactile perception. Hyperparameters for PPO is in Table 2. For auditory perception, we trained an agent to perform the *Kick Squeaky Ball* task in *VECASingleRoomEnv* with varying audio quality: stereo + HRTF, stereo (without HRTF), and mono (without spatialization). *Kick Squeaky Ball* task is identical to *KickTheBall* task on the Table 4. For tactile perception, we trained an agent to perform the *Grab Block* task trained with and without tactile perception. *Grab Block* task is identical to *Grab Object* task, but the only difference is that it object is a large block. For the *Grab Block* task, the agent always looks toward the block (i.e., the block is always located at the center of each vision).

We used the policy network architecture in Figure 7. On the *Kick Squeaky Ball* task, we give a helper reward calculated as $0.01 \cos\theta$, where θ is the angle between the velocity vector and the displacement vector to the ball. To encourage the agent to move its hands to the object, *Grab Block* task gives a helper reward calculated as $d_{t-1} - d_t$, where the sum-of-distance $d_t = |R_t - O_t| + |L_t - O_t|$, and L_t the position vector of left hand, R_t the position vector of right hand, and O_t the position vector of obejct. On top of that, we also gave a negative quadratic penalty of $-0.004 * |a_t|^2$, where a_t is the action vector, to prevent agent from performing outrageous actions.

Experiment 4: Usefulness of VECA Toolkit.

We illustrate the result plot of this experiment in the **Figure 8** of this technical appendix.

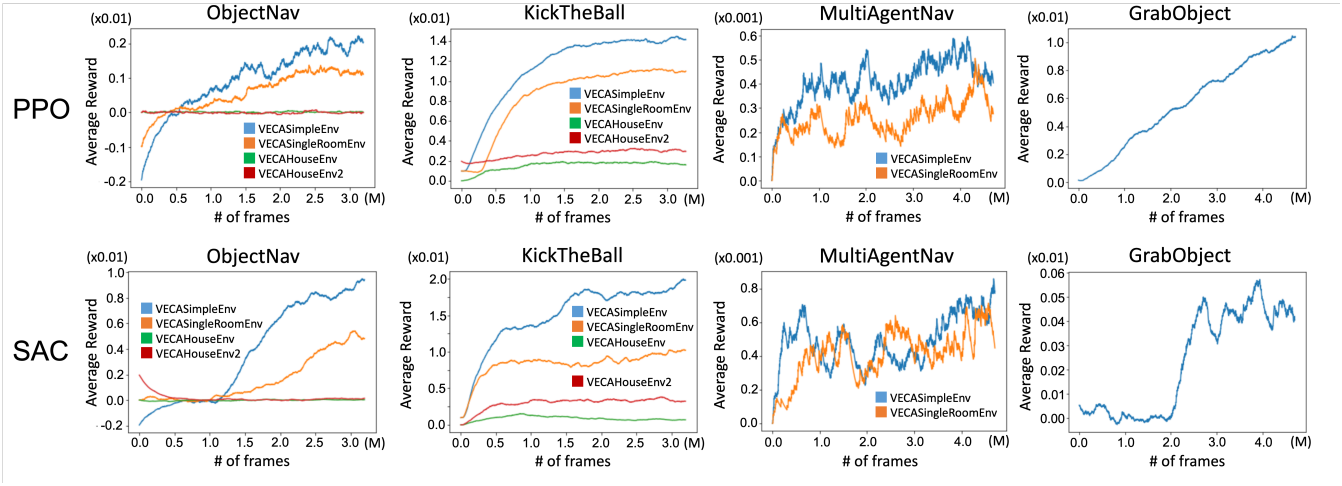


Figure 8: Learning curve over mean reward(y-axis) of agents trained by PPO and SAC. It shows that our VECA toolkit can easily develop diverse domains of cognitive tasks with controllable difficulties. We use the subset of toolkit-generated tasks which are not in our VECA benchmark: *ObjectNav*, *KickTheBall*, *MultiAgentNav*, *GrabObject*. Difficulty is controlled with a playground (organization of the environment).

Experiment Setup. We apply PPO (Schulman et al. 2017) and SAC (Haarnoja et al. 2018) policy gradient algorithm to train VECA agents. Hyperparameters of these methods are set as Table 2. We pick four task representing the aforementioned task categories: *GrabObject*, *ObjectNav*, *KickTheBall*, *MultiAgentNav*; refer to Table 4 for detailed task description. We experiment with all compatible playgrounds to vary the task difficulty.

We add helper rewards to accelerate the training of the tasks. As did in experiment 3, *KickTheBall* task gives a helper reward calculated as $0.01cos\theta$, where θ is the angle between the velocity vector and the displacement vector to the ball. *GrabObject* task also gives a helper reward and a negative quadratic penalty. *ObjectNav and MultiAgentNav* task gives a helper reward calculated as $I_{vis} \cdot (0.05 \cdot a_f + 0.03 \cdot a_l \cdot L)$, where a_f and a_l is the magnitude of velocity vector projected on forward and left direction. I_{vis} is an indicator function which is 1 when the object is visible to the agent, and 0 if not. L is 1 when the object is on the left side of the agent, and -1 if on the right.

References

Akkaya, I.; Andrychowicz, M.; Chociej, M.; Litwin, M.; McGrew, B.; Petron, A.; Paino, A.; Plappert, M.; Powell, G.; Ribas, R.; et al. 2019. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*.

Aylward, G. 2017. Bayley Scales of Infant and Toddler Development.

Beattie, C.; Leibo, J. Z.; Teplyashin, D.; Ward, T.; Wainwright, M.; Küttler, H.; Lefrancq, A.; Green, S.; Valdés, V.; Sadik, A.; et al. 2016. Deepmind lab. *arXiv preprint arXiv:1612.03801*.

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279.

Brodeur, S.; Perez, E.; Anand, A.; Golemo, F.; Celotti, L.; Strub, F.; Rouat, J.; Larochelle, H.; and Courville, A. 2017. Home: A household multimodal environment. *arXiv preprint arXiv:1711.11017*.

Burda, Y.; Edwards, H.; Pathak, D.; Storkey, A.; Darrell, T.; and Efros, A. A. 2019. Large-Scale Study of Curiosity-Driven Learning. *ArXiv*, abs/1808.04355.

Chen, C.; Jain, U.; Schissler, C.; Gari, S. V. A.; Al-Halah, Z.; Ithapu, V. K.; Robinson, P.; and Grauman, K. 2020. Soundspaces: Audio-visual navigation in 3d environments. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VI 16*, 17–36. Springer.

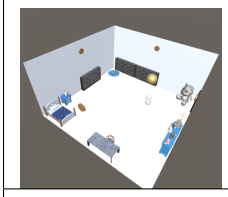
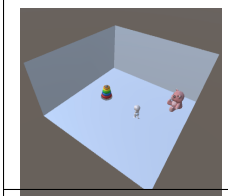
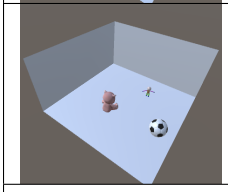
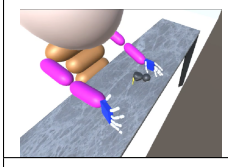
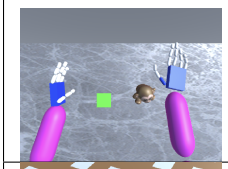
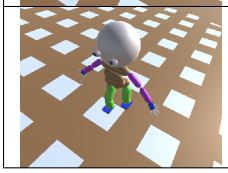
Cohen, L. B.; and Cashon, C. H. 2007. Infant cognition. *Handbook of child psychology*, 2.

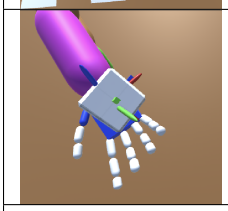
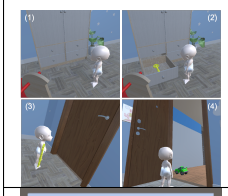
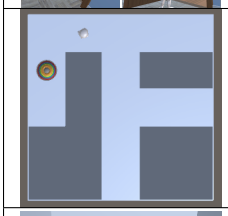
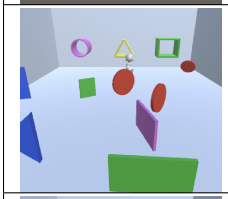
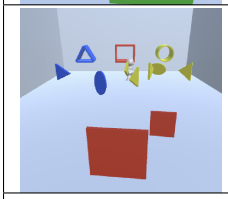
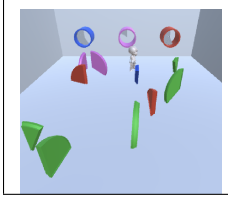
De Vries, H.; Strub, F.; Chandar, S.; Pietquin, O.; Larochelle, H.; and Courville, A. 2017. Guesswhat?! visual object discovery through multi-modal dialogue. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 5503–5512.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255. Ieee.

Duan, Y.; Schulman, J.; Chen, X.; Bartlett, P. L.; Sutskever, I.; and Abbeel, P. 2016. RL2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*.

Eckerman, C. O.; and Whatley, J. L. 1977. Toys and social interaction between infant peers. *Child Development*, 1645–1656.

Figure	Task	Observation	Action space	Description
	KickTheBall	Vision, Audio	Continuous / Discrete	A ball with a buzzing sound continuously comes out from one of the sidewalls. The agent needs to go closer to the ball to kick it. The agent has to use auditory perception to predict the ball's position since its field of view is limited.
	ObjectNav	Vision	Continuous / Discrete	In the environment, there are multiple objects, including the target object. When the agent has navigated to the target object, then the agent receives +1 reward. If the agent has navigated to the wrong object, then the agent receives -1 reward.
	ObjectPhysNav	Vision, Tactile, Proprioception	Continuous	Same as *ObjectNav*, but have to learn the task with joint-level actions.
	MultiAgentNav	Vision, Tactile, Proprioception	Continuous / Discrete	Agents are distributed in each room, and the object is in one of those rooms. All agents receive +1 reward when each agent navigates to the object. Without cooperation, the agent must traverse all rooms and find the desired object. The agent who found the object must notify other agents by making a sound, enabling other agents to navigate the object using auditory perception.
	GrabObject	Vision, Tactile, Proprioception	Continuous	Learn how to recognize and grab an object with joint-level actions. The agent is rewarded by the vertical position difference when the object is close enough to each hand (preventing the agent from "throwing" the object).
	MoveToTarget	Vision, Tactile, Proprioception	Continuous	Learn how to recognize and move the object to the target position with joint-level actions. The agent is rewarded by the difference of distance between the target position and the object's position.
	TurnBaby	Tactile, Proprioception	Continuous	Learn how to turn its body with joint-level actions. The agent is rewarded by the orientation of its torso and hip.
	RunBaby	Tactile, Proprioception	Continuous	Learn how to run with joint-level actions. The agent is rewarded by the velocity to the direction of its torso.

	CrawlBaby	Tactile, Proprioception	Continuous	Learn how to crawl with joint-level actions. The agent is rewarded by the velocity and the orientation of its torso and hip.
	SitBaby	Tactile, Proprioception	Continuous	Learn how to sit with joint-level actions. The agent is rewarded by the position of its torso and head.
	RotateCube	Tactile, Proprioception	Continuous	Learn how to rotate the cube to target rotation using its hand with joint-level actions. The agent is rewarded by the difference in distance between the target orientation and the cube's orientation.
	Babyzuma's revenge	Vision, Audio	Continuous & Discrete	Similar to *Montezuma's revenge*, the agent has to learn complex sequence of interactions to complete the level in sparse reward settings, with vision and audio senses. For instance, the agent has to: 1) navigate to the drawer, 2) open the drawer, 3) grab the key (in the drawer), 4) open the door to clear the first room.
	MazeNav	Vision	Continuous / Discrete	Adopted the navigation experiment from RL^2(Duan et al. 2016). The agent is rewarded when the agent navigates to the object. The agent can repeat five trials for each maze.
	ShapeSort	Vision	Continuous & Discrete	The agent has to sort the objects according to their shape. The agent is rewarded when the agent grabs the objects and puts the object to the correct shape basket.
	ColorSort	Vision	Continuous & Discrete	The agent has to sort the objects according to their color. The agent is rewarded when the agent grabs the objects and puts the object to the correct color basket.
	FillFraction	Vision	Continuous & Discrete	The agent has to fill three circles using the fraction circle objects. The agent can not put the object when the sum of the fraction exceeds the circle. The agent is rewarded when the agent puts the object to the circle and is additionally rewarded when it is full.

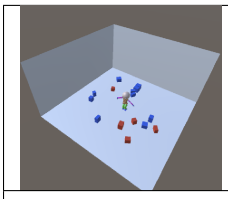
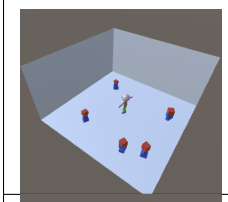
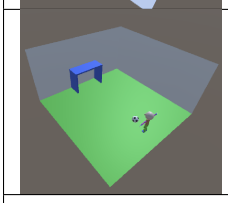
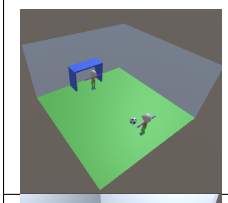
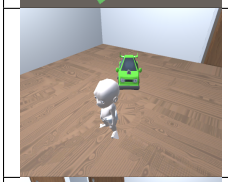
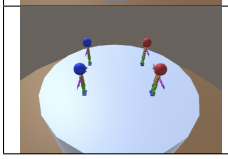
	PileUpBlock	Vision, Tactile, Proprioception	Continuous	The agent has to pile up the (red) blocks with joint-level actions. The agent is rewarded according to the sum of the y-axis position of red blocks.
	PutDownBlock	Vision, Tactile, Proprioception	Continuous	The agent has to put down (only the red) blocks with joint-level actions. The agent is rewarded according to the negative sum of the y-axis position of red blocks. Additionally, the agent gets a negative reward when the blue blocks fall.
	SoccerOnePlayer	Vision, Tactile, Proprioception	Continuous	Put the ball into the goal with joint-level actions. Note that there is no rule: e.g., the agent may use hands.
	SoccerTwoPlayer	Vision, Tactile, Proprioception	Continuous	The attacker must put the ball into the goal, and the defender must block the ball from the goal with joint-level actions. Note that there is no rule: e.g., the agent may use hands and block or tackle the opponent player.
	DodgeCar	Vision	Continuous / Discrete	The toy car moves around in the room fast. The agent must dodge the car.
	ChaseCar	Vision, Tactile, Proprioception	Continuous & Discrete	The toy car moves around in the room fast. The agent must chase and interact with the car, but must not collide with the car.
	PushOut1v1	Vision, Tactile, Proprioception	Continuous	The agent must push other agents out of the ring to win. The agent is equipped with joint-level actions.
	PushOut2v2	Vision, Tactile, Proprioception	Continuous	Agents must push agents of the opponent team out of the ring to win. The agent is equipped with joint-level actions.

Table 4: Detailed description of various VECA toolkit-generated tasks, which are not VECA benchmark tasks.

ObservationUtils	float[] getImage (*camera*, *height*, *width*, *grayscale*)	Return the image of the camera with the size of (*height*, *width*). Returns grayscale image if *grayscale* is true, otherwise returns RGB image.
	float[] getSpatialAudio (*head*, *earL*, *earR*, *env*)	Return the raw spatilized audio data with audio sources included in *env* based on the position of *head* and both ears (*earL*, *earR*).
	float[] getSpatialAudio (*head*, *earL*, *earR*, *env*, Vector3 roomSize, float beta)	Return the raw spatilized audio data, also including the room impulse generated by the room size of *roomSize* and reflection coffecient *beta*.
	float[] getTactile (*mesh*)	Return the tactile data with the mesh of the agent.
	float[] getTactile (*taxels*, *taxelratio*, *debug*)	Return the tactile data with the *taxels* of the agent. If *debug* is true, each taxel displays a blue line which represents the direction and impulse of tactile perception. To prevent massive display of lines during debugging, user could disable some of the taxels using *taxelratio*.
GeneralAgent	void AddObservations (*key*, *observation*)	Add the *observation* vector in its data buffer with its *key*.

Table 5: Some of the VECA toolkit API interfaces regarding the perception and sensory input of VECA embodied agent.

Entwistle, N. 2007. Conceptions of learning and the experience of understanding: Thresholds, contextual influences, and knowledge objects.

Espeholt, L.; Soyer, H.; Munos, R.; Simonyan, K.; Mnih, V.; Ward, T.; Doron, Y.; Firoiu, V.; Harley, T.; Dunning, I.; Legg, S.; and Kavukcuoglu, K. 2018. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. *ArXiv*, abs/1802.01561.

Foerster, J.; Farquhar, G.; Afouras, T.; Nardelli, N.; and Whiteson, S. 2017. Counterfactual multi-agent policy gradients. *arXiv preprint arXiv:1705.08926*.

Gemmeke, J. F.; Ellis, D. P.; Freedman, D.; Jansen, A.; Lawrence, W.; Moore, R. C.; Plakal, M.; and Ritter, M. 2017. Audio set: An ontology and human-labeled dataset for audio events. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 776–780. IEEE.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*.

Ho, J.; and Ermon, S. 2016. Generative adversarial imitation learning. In *Advances in neural information processing systems*, 4565–4573.

Johnson, M.; Hofmann, K.; Hutton, T.; and Bignell, D. 2016. The Malmo Platform for Artificial Intelligence Experimentation. In *IJCAI*, 4246–4247.

Juliani, A.; Berges, V.-P.; Vckay, E.; Gao, Y.; Henry, H.; Mattar, M.; and Lange, D. 2018. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*.

Kempka, M.; Wydmuch, M.; Runc, G.; Toczek, J.; and Jaśkowski, W. 2016. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, 1–8. IEEE.

Kolve, E.; Mottaghi, R.; Han, W.; VanderBilt, E.; Weihs, L.; Herrasti, A.; Gordon, D.; Zhu, Y.; Gupta, A.; and Farhadi, A. 2017. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*.

Küttler, H.; Nardelli, N.; Lavril, T.; Selvatici, M.; Sivakumar, V.; Rocktäschel, T.; and Grefenstette, E. 2019. TorchBeast: A PyTorch Platform for Distributed RL. *arXiv preprint arXiv:1910.03552*.

Landau, B.; Smith, L.; and Jones, S. 1998. Object perception and object naming in early development. *Trends in cognitive sciences*, 2(1): 19–24.

Lowe, R.; Wu, Y. I.; Tamar, A.; Harb, J.; Abbeel, O. P.; and Mordatch, I. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in neural information processing systems*, 6379–6390.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 3111–3119.

Ngiam, J.; Khosla, A.; Kim, M.; Nam, J.; Lee, H.; and Ng, A. Y. 2011. Multimodal deep learning. In *ICML*.

OpenAI. 2016. Openai universe. https://universe.openai.com.

Pathak, D.; Agrawal, P.; Efros, A. A.; and Darrell, T. 2017. Curiosity-Driven Exploration by Self-Supervised Prediction. *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 488–489.

Savva, M.; Chang, A. X.; Dosovitskiy, A.; Funkhouser, T.; and Koltun, V. 2017. MINOS: Multimodal Indoor Simulator for Navigation in Complex Environments. *arXiv:1712.03931*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Class Name	Function	Description
VECAHumanoid ExampleInteract	void walk (*walkSpeed*, *turnSpeed*)	Make the agent walk. Speed and trajectory of the agent is determined by *walkSpeed* and *turnSpeed*.
	void kick (*obj*)	Make the agent kick the object (*obj*).
	void grab (*obj*)	Make the agent grab the object (*obj*).
	void release ()	Make the agent release the grabbed object.
	void adjustFocalLength(*newFocalLength*)	Adjust the focal distance of the cameras.
	void lookTowardPoint(*pos*)	Make the agent look at 3-dimensional point. Note that this function doesn't rotate the head : it only rotates the camera(eye). Also, the agent will look at the point until releaseTowardPoint() is called, even when the agent is moving.
	void rotateUpDownHead(*deg*)	Rotate the head in the Up-Down plane. Head goes down when deg>0. Please note that this function doesn't have any constraints: Excessive rotation might distort the mesh of the agent.
	void rotateLeftRightHead(*deg*)	Rotate the head in the Left-right plane. Head goes right when deg>0. Please note that this function doesn't have any constraints: Excessive rotation might distort the mesh of the agent.
	void makeSound(*audiodata*)	Make a voice according to the audiodata. Please note that the agent also percieves its voice.
	bool isVisible(*obj*, *cam*)	Check if the object *obj* is visible by the camera *cam*. Note that the object is considered visible even if it is occluded by transparent objects.
	bool isInteractable(*obj*)	Check if the object *obj* is interactable with the agent. The object is considered interactable when 1) the object is visible, 2) there is no transparent objects between the object and the agent 3) the distance is closer than 2.5m.
	List<VECAObjectInteract> getInteractableObjects()	Returns the list of objects which is interactable with the agent.
	VECAObjectInteract getInteractableObject()	Returns a single object which is interactable with the agent. If there are multiple interactable objects, it chooses the object which is closest to the center of agent's viewport.
VECAHumanoid PhysicalExample Interact	List<Vector3> GetVelocity()	Returns the velocity of the bones.
	List<Vector3> GetAngVelocity()	Returns the angular velocity of the bones.
	List<float> GetAngles()	Returns the current angle for all joints (length is same to degree of freedom).
	void ApplyTorque(*normalizedTorque*)	Apply the torques to the joint according to the *normalizedTorque*. Input must be the length same to the degree of freedom and within [-1, 1].
	void updateTaxels()	Update the taxels (used in tactile calculation) according to the orientation of the bones and the mesh of the skin. In default, 6 taxels are distributed in triangle formulation for each triangle of the mesh.

Table 6: Some of the VECA toolkit API interfaces regarding the action capability of VECA embodied agent.

Song, Y.; Wojcicki, A.; Lukasiewicz, T.; Wang, J.; Aryan, A.; Xu, Z.; Xu, M.; Ding, Z.; and Wu, L. 2020. Arena: A General Evaluation Platform and Building Toolkit for Multi-Agent Intelligence. In *AAAI*, 7253–7260.

Tacca, M. C. 2011. Commonalities between perception and cognition. *Frontiers in psychology*, 2: 358.

Trevarthen, C. 1979. Communication and cooperation in early infancy: A description of primary intersubjectivity. *Before speech: The beginning of interpersonal communication*, 1: 530–571.

Vercauteren, T.; Unberath, M.; Padoy, N.; and Navab, N. 2019. Cai4cai: the rise of contextual artificial intelligence in computer-assisted interventions. *Proceedings of the IEEE*, 108(1): 198–214.

Wu, Y.; Wu, Y.; Gkioxari, G.; and Tian, Y. 2018. Building generalizable agents with a realistic and rich 3D environment. *arXiv preprint arXiv:1801.02209*.

Xia, F.; R. Zamir, A.; He, Z.-Y.; Sax, A.; Malik, J.; and Savarese, S. 2018. Gibson env: real-world perception for embodied agents. In *Computer Vision and Pattern Recognition (CVPR), 2018 IEEE Conference on*. IEEE.