ROBOTURK: A Crowdsourcing Platform for Robotic Skill Learning through Imitation

Ajay Mandlekar[‡], Yuke Zhu, Animesh Garg, Jonathan Booher, Max Spero, Albert Tung,

Julian Gao, John Emmons, Anchit Gupta, Emre Orbay, Silvio Savarese, Li Fei-Fei

Department of Computer Science, Stanford University
[‡]amandlek@stanford.edu

Abstract: Imitation Learning has empowered recent advances in learning robotic manipulation tasks by addressing shortcomings of Reinforcement Learning such as exploration and reward specification. However, research in this area has been limited to modest-sized datasets due to the difficulty of collecting large quantities of task demonstrations through existing mechanisms. This work introduces ROBOTURK to address this challenge. ROBOTURK is a crowdsourcing platform for high quality 6-DoF trajectory based teleoperation through the use of widely available mobile devices (e.g. iPhone). We evaluate ROBOTURK on three manipulation tasks of varying timescales (15-120s) and observe that our user interface is statistically similar to special purpose hardware such as virtual reality controllers in terms of task completion times. Furthermore, we observe that poor network conditions, such as low bandwidth and high delay links, do not substantially affect the remote users' ability to perform task demonstrations successfully on ROBOTURK. Lastly, we demonstrate the efficacy of ROBOTURK through the collection of a pilot dataset; using ROBOTURK, we collected 137.5 hours of manipulation data from remote workers, amounting to over 2200 successful task demonstrations in 22 hours of total system usage. We show that the data obtained through ROBOTURK enables policy learning on multi-step manipulation tasks with sparse rewards and that using larger quantities of demonstrations during policy learning provides benefits in terms of both learning consistency and final performance. For additional results, videos, and to download our pilot dataset, visit roboturk.stanford.edu

Keywords: Crowdsourcing, Imitation Learning, Skill Learning, Manipulation

1 Introduction

Large-scale datasets, consisting of millions of data points, have accelerated performance in computer vision and natural language tasks and enabled many practical applications. Richly annotated data has provided a fertile ground for developing and evaluating a broad set of learning algorithms for problems such as image classification and machine translation [1, 2, 3]. Supervised data is similarly useful for sequential decision making problems in robotics.

Recent research has successfully employed Reinforcement Learning (RL), Self-Supervised Learning (SSL), and Imitation Learning (IL) for short-horizon skill learning, such as pushing and grasping objects [4, 5, 6]. However, RL methods require reward specification, face the burden of efficient exploration in large state spaces, and need large amounts of interaction with the environment to collect sufficient data for policy learning. SSL has been used successfully to collect and learn from large quantities of data in both simulated [7, 8, 9] and physical settings [5, 10, 11] for tasks such as grasping. However, the data has very low signal-to-noise ratio due to random exploration. In contrast, IL uses expert demonstrations to reduce environment interaction during skill learning. Imitation Learning is commonly posed either as inverse reinforcement learning [12, 13] or behavioral cloning [14, 15], both of which require demonstrations. Prior works show that data-driven IL methods improve both the sample efficiency of policy learning [16, 17, 18] and the performance of the trained policy [19, 20, 21].

2nd Conference on Robot Learning (CoRL 2018), Zürich, Switzerland.

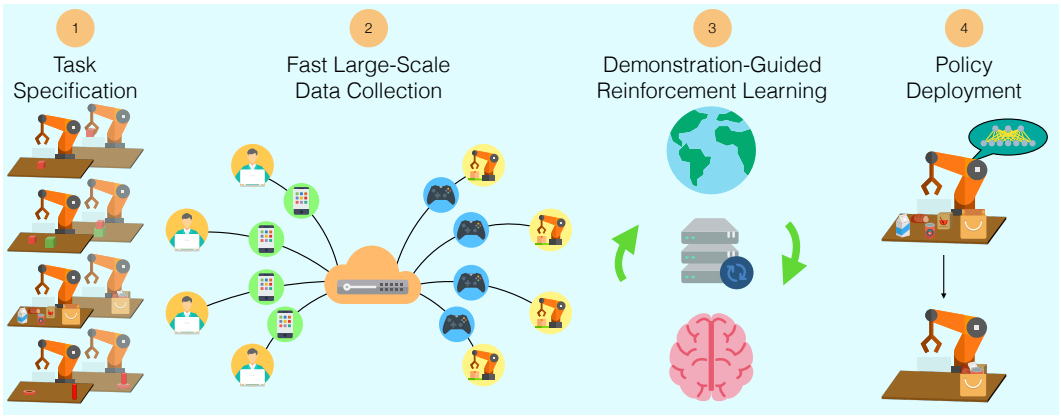


Figure 1: **System overview of RoboTurk.** RoboTurk enables quick imitation guided skill learning. Our system consists of the following major steps: 1) specifying a task, 2) collecting a large set of task demonstrations using RoboTurk, 3) using demonstration-augmented reinforcement learning to learn a policy, and 4) deploying the learned skill in the domain of interest.

There are many conventional mechanisms for collecting task demonstrations for robot learning. One popular option is to kinesthetically guide a robot through a desired trajectory [14]. Although intuitive, this mechanism for supervision can be tedious, and is often limited to collecting tens of demonstrations, while policy learning requires hundreds or even thousands of demonstrations [22]. Alternatively, teleoperation-based techniques for collecting task demonstrations have been employed for over two decades [23, 24, 25]. Teleoperated supervision can be provided through contemporary game interfaces such as a keyboard and mouse [26, 27], a video game controller [28], a 3D-mouse [29, 30], special purpose master-slave interfaces[31, 32], or through free-space positioning interfaces such as virtual reality (VR) [33, 34, 35].

Game interfaces frequently introduce artifacts in trajectories that reduce the utility of this data for imitation [36]. For instance, due to concurrent control of only a subspace of actions, the trajectories can be longer, exhibit axis-aligned movement, and lack natural variations in motion. By contrast, free-space positioning interfaces such as VR enable the user to directly move the robot's end effector by moving a hand controller in 3D space, enabling fine-grained dexterous control. Interestingly, Zhang et al. [34] showed that simpler algorithms such as variants of behavior cloning can be made to learn short-horizon manipulation tasks with data on the order of a few hundred VR demonstrations. However, the requirement of special purpose VR hardware and client-side compute resources has limited the deployment of these interfaces on crowdsourcing platforms such as Amazon Mechanical Turk where a typical worker is more likely to have a smartphone than VR hardware [37, 38].

In other domains, large-scale supervision for datasets is often collected with the assistance of crowdsourcing [1, 3]. This enables a scalable mechanism for diverse human supervision on an extensive set of problem instances. However, collecting large amounts of data has been a challenge for continuous control tasks, as they demand real-time interaction and feedback from annotators, placing difficult constraints on remote teleoperation platforms. Data collection mechanisms for skill learning in robotics need demonstrations from remote users that are both *natural* (which game interfaces tend to lack) and *plentiful* (which free-space positioning interfaces tend to lack).

This paper proposes RoboTurk, a platform to address the challenge of collecting large, crowdsourced sets of task demonstrations for robot manipulation tasks. RoboTurk enables: (a) real-time control of simulated systems, (b) accessibility to a large population of users, and (c) simultaneous platform use by many concurrent users. RoboTurk is one of the first platforms to facilitate crowdsourcing of trajectory-level supervision.

Contributions. The main contributions of this paper are:

1. We introduce RoboTurk, a data collection platform that allows remote workers to log on to a website and collect task demonstrations using a smartphone as a motion controller. RoboTurk is supported by a cloud-based simulation backend that streams video to a client's web browser using low-latency communication protocols. This ensures homogeneous quality of service regardless of a client's compute resources, resulting in a platform that is intuitive to use and has a low barrier to

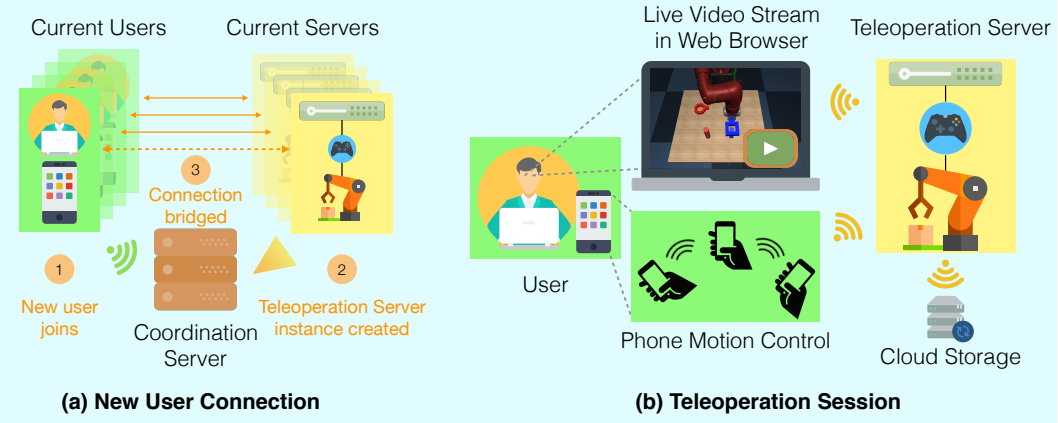


Figure 2: **System diagram of RoboTurk.** A new user connects to a website to join the system, and a coordination server launches a dedicated teleoperation server for the user, as shown in **(a)**. The coordination server then establishes direct communication channels between the user's web browser and iPhone and the teleoperation server to start the teleoperation session. The user controls the simulated robot by moving their phone, and receives a video stream as feedback in their web browser, as shown in **(b)**. After every successful demonstration, the teleoperation server pushes the collected data to a cloud storage system.

entry – the core requirements of a crowdsourced task. RoboTurk supports multiple robots, tasks, and simulators, and can easily be extended to support others.

2. We conduct a user study comparing the performance of three other interfaces, including a VR interface, on simulated lifting and bin picking tasks. Our platform is similar to VR in performance and better than 3D-mouse and keyboard interfaces. We also evaluate the performance of RoboTurk under poor network conditions using a network simulator.
3. We present an initial dataset consisting of over 2200 task demonstrations, amounting to 137 hours of data collected in 20 hours of system usage with contracted workers.
4. We show that the data collected through RoboTurk enables policy learning on challenging manipulation tasks with sparse rewards and that using larger quantities of demonstrations during policy learning provides benefits in terms of both learning consistency and final performance.

2 RoboTurk: Design and Development of the Proposed Platform

We present RoboTurk, a cloud-based large-scale data collection platform for robot learning tasks that enables the collection of thousands of task demonstrations within days.

2.1 Platform Features

1. **Real-time control of simulated robots.** RoboTurk enables users to control robots in simulated domains and provides users with real-time video and haptic feedback.

2. **Accessible to a large population of users.** In order to enable large-scale data collection, RoboTurk is easily accessible to the typical workers on crowdsourcing platforms such as Amazon MTurk. RoboTurk limits the hardware requirements necessary to use it, minimizes the complexity of any client-side application users interact with, and makes its internal communication protocols agnostic to the geographic proximity of users.

3. **Capable of providing real-time feedback and low-latency robot control for many simultaneous users.** In order to collect thousands of robot demonstrations quickly, RoboTurk is capable of interacting with many users at the same time. This requires careful system design with respect to user session management, communication protocols used to connect users to simulator instances, and the quality of the cloud machines used to host the platform.

4. **Modular design to enable platform extensibility.** RoboTurk adopts a modular design paradigm to decouple task and controller design from the rest of the infrastructure — this makes it easy to extend RoboTurk to new tasks, physical simulators, and robots.

Lifting Picking Assembly

Figure 3: **Tasks**: We evaluated our platform on three simulated manipulation tasks which contain a 7-DoF Sawyer robot arm and various objects. In the *lifting* task (left) the objective is to control the robot arm to grab the cube and lift it. In the *picking* task (middle) the objective is to place each object into its corresponding bin. In the *assembly* task (right), the objective is to place a round nut and a square nut onto their corresponding pegs.

2.2 Platform Design and Implementation

RoboTurk implements the features listed in Sec. 2.1 by deploying its core components on cloud infrastructure. RoboTurk offloads robot simulation to powerful cloud machines instead of local execution. The robot simulation machines communicate over low-latency channels with users to receive commands controlling the position of the simulated robot arms; simulator camera views of the robots are rendered in real-time and transmitted to the users over separate low-latency links. RoboTurk leverages Web Real-Time Communication (WebRTC), an open-source framework for low-latency transmission of both robot control commands and the rendered camera views from the robot simulator. Fig. 2 outlines how system components are involved in (1) letting a new user seamlessly connect to the platform and (2) allowing a user to collect demonstrations inside a dedicated teleoperation session.

By taking this design approach, RoboTurk eliminates the need for users to have a powerful local workstation and to install or configure any software. Instead, users only need to have (1) an iPhone that is compatible with Apple's ARKit framework (iPhone 6S and above) and (2) access to a separate device with a web browser, both of which are ubiquitous [38].

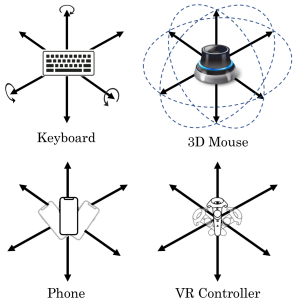
User Endpoint. The user receives a real-time video stream of the robot arm in their browser window and moves their iPhone to control the robot arm. The phone's pose is tracked via Apple's ARKit platform, which uses a combination of camera frames and motion sensor data. The phone's pose is packaged along with some status information and transmitted through our platform to a teleoperation server instance, which is a dedicated process responsible for controlling the robot. The teleoperation server sends back task-specific information, allowing the user to receive haptic feedback through phone vibrations when the robot arm makes contact with objects in the simulation environment.

Coordination Server. The coordination server is responsible for creating and maintaining user teleoperation sessions. As Fig. 2a shows, when a new user enters the system, the coordination server spawns a dedicated teleoperation server instance, so that every user can control their own simulated robot arm. The coordination server then establishes two low latency WebRTC communication channels — one between the user's browser and the teleoperation server and another between the user's phone and the teleoperation server.

Teleoperation Server. Each teleoperation server receives phone commands from exactly one user and processes them to control the robot arm in the simulator. Phone poses are mapped to a set of joint velocities to control the simulated robot at a regulated rate. Additional details in Appendix A.6.

3 System Analysis

In this section, we benchmark RoboTurk and evaluate some of our design choices. In particular, we evaluate our iPhone interface against other common interfaces motivated by prior work [27, 30, 34] and the performance of our platform under low bandwidth and high delay network conditions.



	Keyboard	3D Mouse	VR Controller	Phone
Keyboard	-	0.575 (0.000)	0.900 (0.000)	0.725 (0.000)
3D Mouse	0.575 (0.000)	-	0.375 (0.005)	0.325 (0.022)
VR Controller	0.900 (0.000)	0.375 (0.005)	-	**0.225 (0.231)**
Phone (ours)	0.725 (0.000)	0.325 (0.022)	**0.225 (0.231)**	-

Figure 4: **UI Comparison: (a)** Illustration of Interfaces and the movement they allow — axis-aligned (Keyboard), 6-DoF (3D Mouse), and Free-Space (VR and Phone). **(b)** Table: Kolmogorov-Smirnov statistics between distributions for completion times in the *picking* task across different interfaces, followed by associated p-values. Pairwise K-S statistic is used as a measure of difference between underlying distributions. Based on statistical significance level of 5%, we observe that the completion times follow the order: Phone $\approx$ VR Controller $\succ$ 3D Mouse $\succ$ Keyboard

3.1 Tasks

Our experiments use three simulated tasks: *Block Lifting* (*lifting*), *Bin Picking* (*picking*), and *Nut-and-peg Assembly* (*assembly*), as illustrated in Fig. 3. *Lifting*, a simple task where a Sawyer robot arm must lift a cube, serves as a diagnostic example, while the *picking* and *assembly* tasks, which consist of sorting objects into bins and fitting nuts onto pegs respectively, are more challenging settings. These are a part of the SURREAL Robotics Suite [39], a set of manipulation tasks developed using the MuJoCo physics engine [40] inspired by the tasks in World Robot Summit [41]. Please refer to Appendix A.1 for task details.

To evaluate our platform, we conducted a user study with 8 university students aged 18-30. Each user provided 20 demonstrations on the *lifting* task and 5 on the *picking* task for each of 8 different test conditions. In the first 4 test conditions, we varied the control interfaces. In the latter 4 test conditions, we varied the network conditions that users experienced while controlling the robot arm.

3.2 User Interface Evaluation

As Fig. 4a shows, we compare four user interfaces for robot control: Keyboard, 3D Mouse, Virtual Reality (VR) Controller, and Phone (ours). The HTC Vive is special-purpose VR hardware that uses external-tracked pose for hand-held controllers, which we map one-to-one to the robot end effector pose. The tracking is high fidelity, but every remote teleoperator would require access to specialized VR hardware. The 3D mouse interface is a joystick that can translate simultaneously along x, y, and z and rotate about these axes as well. The local movements of the 3D mouse are mapped to the robot end effector. While it may not offer the same accuracy as the HTC Vive, it is only a fraction of the cost, and can be seen as a compromise between cost and performance. The keyboard is the most ubiquitous of the four interfaces considered, but it lacks natural 3D control and could easily lead to degenerate, axis-aligned demonstrations that do not resemble free-form, 6-DoF movement. In our implementation, we used different keys to translate or rotate by a fixed amount in each direction, and a separate key to open and close the gripper. An ideal control interface would be as commonplace as a keyboard while as intuitive as a VR controller.

We designed the user interface on the phone to try and combine the ease of control and accuracy of the VR controller with the ubiquity of the keyboard. Pose-tracking on phones has been massively improved in the last few years, especially with the advent of ARKit and ARCore platforms. We utilize ARKit to track the phone's pose using the camera and internal gyroscope. While the phone's tracking mechanism is often noisy and does not match the fidelity of the VR controllers, we observed that our subjects often adapted to account for tracking noise after collecting a few demonstrations and getting familiar with the interface.

We compare all four user interfaces on both the *lifting* task and the *picking* task. Results on both tasks show that the keyboard interface was significantly slower than the other three. Fig. 5A shows that the 3D mouse requires a mean completion time of 112.57 seconds on the *picking* task, being slower than both the phone and VR controller. Please refer to Appendices A.2 and A.3 for detailed data on

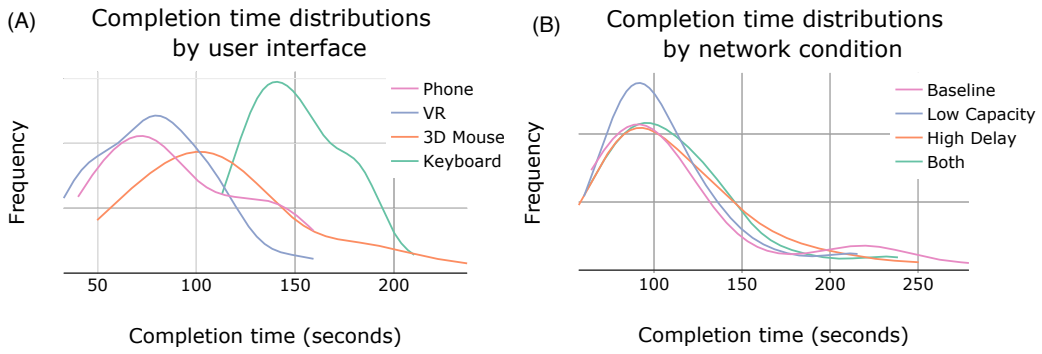


Figure 5: **System Analysis**: **(A)** Comparison of completion time distributions of different user interfaces on the *picking* task. **(B)** Comparison of completion time distributions of different network conditions using the phone interface for the *picking* task.

completion times. Fig. 4b summarizes the results of Kolmogorov-Smirnov tests over distribution of completion times for each interface on the *picking* task. We observe that the keyboard (mean 151.45 seconds) and 3D mouse (mean 112.57 seconds) interfaces are statistically slower than the Phone (mean 89.97 seconds) and VR (mean 79.36 seconds) at a 5% significance level. However, we were unable to conclude that the distribution of completion times on the Phone interface was different than that of the VR controller. These results support our hypothesis that the RoboTurk phone interface is similar in performance to the VR interface, but widely available owing to the ubiquity of the iPhone.

3.3 Robustness to Network Capacity and Delays

We evaluate the performance of RoboTurk under various network conditions. We used the Cellsim network simulation tool [42] to regulate the network capacity and delay between the platform and the users. Cellsim can faithfully replay network conditions recorded from real networks or operate according to synthetically generated network traces. Using Cellsim, we characterized the platform under four different network conditions. The first test was a baseline where the network had a constant 2.4Mbps of up/downlink bandwidth and 20ms of one-way delay. The second test emulated a low-capacity network link (constant 500Kbps; 20ms delay), the third emulated a high-delay link (constant 2.4Mbps; 120ms delay), and the last emulated a low-capacity and high-delay link (constant 500Kbps; 120ms delay). We used 500Kbps as a conservative estimate of a slow internet connection (such as a 3G network) and 2.4Mbps as an estimate of a slow home consumer connection. We used 120ms of one-way delay to approximate a Trans-Pacific connection (e.g. from the US to China) which was confirmed as a reasonable estimate through real world testing (see Sec. 3.4). Please refer to Table 6 in the Appendix for further details on Kolmogorov-Smirnov tests over these distributions.

Evaluations performed on the three troublesome network traces produced roughly the same distributions of completions times as the baseline (see Fig. 5). This robustness evaluation of RoboTurk under a wide range of network conditions is a first step to affirm its capacity to serve clients globally regardless of their locations.

3.4 System Stress Test: Remote Teleoperation from California to China

We also evaluate RoboTurk for demonstration collection at large distances. We compare the platform's performance over real-world networks across different geographic locations. In the first experiment, users in California performed tasks using a deployment of RoboTurk running in a data center in China (approximately 6500 miles away); in the second experiment, the same users performed the same tasks but with RoboTurk running in a data center in Oregon (approximately 500 miles away). Qualitatively, the users found that using the data center in China introduced a nearly constant delay compared to the Oregon servers. Users were able to complete the tasks despite the high delays in network communication to and from China, but completion times were slower than when using the Oregon data center (Fig. 6). The mean completion times different by 24 and 28 seconds for the *assembly* and *picking* tasks respectively. Overall, this stress test demonstrates that RoboTurk is robust to poor real-world network conditions, allowing for tasks to be successfully completed (albeit at a slower pace) and has potential to connect servers to remote workers worldwide.

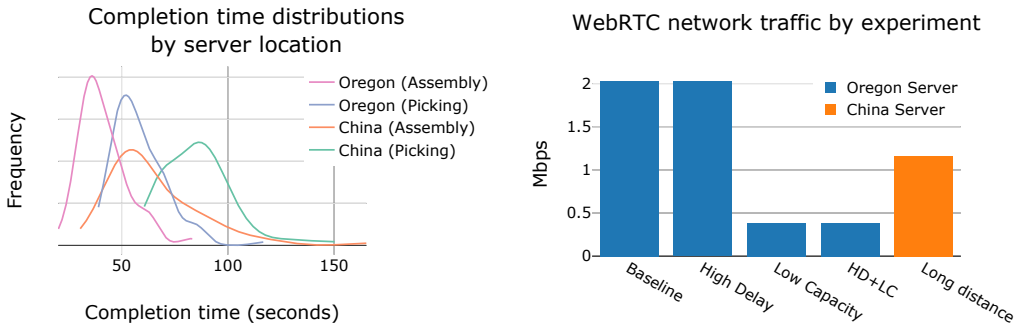


Figure 6: **System Stress Test**: Left: Comparison of completion time distributions between servers located in Oregon vs. in China, when tested from California. Right: Comparison of average throughput of video stream by network experiment, as well as for cross-pacific experiments.

4 Policy Learning from Teleoperated Demonstrations

In this section, we evaluate the utility of the demonstration data collected by RoboTurk. To this end, we collected a pilot dataset on two challenging manipulation tasks, and we answer the following questions: (1) Can the collected demonstrations help enable, accelerate, and improve policy learning? (2) Is a larger number of task demonstrations valuable to learning? The second question is vital, as the core goal of RoboTurk is to scale demonstration collection.

4.1 Data Collection and Task Setup

As an initial pilot for our platform, we collected a dataset of over 1000 demonstrations on both the *picking* and *assembly* tasks with 20 hours of total platform usage by using contractors that were located remotely. Indeed, this validates the notion that RoboTurk is a dynamic data collection platform that can be used to quickly crowdsource data on-demand. The Appendix contains some additional information and statistics on the dataset.

We evaluate the utility of the collected demonstrations for learning policies with reinforcement learning from sparse rewards. We consider the *Bin Picking (Can)* and *Nut-and-peg Assembly (Round)* tasks from the Surreal Robotics Suite [39], abbreviated as *can picking* and *round assembly* respectively. These are simplified versions of the full *picking* and *assembly* tasks, where the goal is to place a single can into its corresponding bin and to fit a single round nut onto its corresponding peg respectively. We do not use reward shaping; a non-zero reward is provided to the agent only if the agent successfully completes the task. This makes exploration challenging – the agent has little chance of finding a goal state via random exploration due to the compositional structure of these tasks.

4.2 Demonstration-Guided Reinforcement Learning

To leverage demonstrations during reinforcement learning, we initialize training episodes from states sampled randomly from the demonstration trajectories and do policy learning using a distributed implementation of proximal policy optimization (PPO) [43, 44]. This simple method for doing reinforcement learning with demonstrations has been motivated theoretically by [45] and used in practice in prior and concurrent work [46, 47, 39]. We encourage using states visited in demos, as opposed to new experience by setting a myopic horizon of 100 timesteps for training episodes.

To investigate the benefits of leveraging more demonstrations to guide reinforcement learning, we compared utilizing None (pure RL), 1, 10, 100, and 1000 demonstrations on the two tasks. We trained policies from 10 different random seeds for each task and demonstration count and report the mean and standard deviation of the final policy return after 24 hours on the *can picking* task and 48 hours on the *round assembly* task. The results are presented in Table 1 and additional details are described in Appendix A.8.

4.3 Results

Table 1 demonstrates the value of leveraging large numbers of demonstrations to guide reinforcement learning. Using 1000 demonstrations (nearly the entire RoboTurk pilot dataset) resulted in the best

	Number of Demonstrations				
Task	None	1	10	100	1000
Bin Picking (Can)	0 ± 0	278 ± 351	273 ± 417	385 ± 466	$\mathbf{641 \pm 421}$
Nut-and-peg Assembly (Round)	0 ± 0	381 ± 467	663 ± 435	575 ± 470	$\mathbf{775 \pm 388}$

Table 1: **Quantitative results of learning with demonstrations.** This table shows how the average performance of trained policies varies with the number of demonstrations utilized during policy learning. The maximum possible return for each task is 1000 — this would correspond to a policy that instantaneously solves the task. We observe that utilizing more demonstrations generally leads to better average task performance, and our best experimental results were obtained by using 1000 demonstrations (nearly the entire ROBOTURK dataset) on each task.

mean task performance on both tasks. This implies that leveraging larger numbers of demonstrations can help policies learn to solve the task more consistently. The results reported in Table 1 exhibit high variance since every setup had some fraction of the 10 different policies that were unable to solve the task successfully in the allotted time, and consequently had a return of 0. There are also policies that could not solve the task consistently from every start state, or policies that took longer to solve the task. Both behaviors result in lower average returns for the policy. Thus, the high average performance for the 1000 demonstration experiments suggest that the increased number of demonstrations enable consistently training near-optimal policies that solve the task.

We observed a much higher gap in performance between the 1000 demonstration results and the other results for the *can picking* task than for the *round assembly* task. We suspect that this is because policies on the *round assembly* task were given more time to train. We allowed a period of 48 hours for the *round assembly* task because we found that 24 hours was an insufficient amount of time for policies to solve the task. The larger quantity of agent experience collected during the 48 hours could explain how some agents trained with only 10 demonstrations were able to achieve comparable performance to agents trained with 100 demonstrations. Over longer time periods, the benefits of experience diversity present in the 1000 demonstration dataset could plausibly be mirrored by using 10 demonstrations and giving our distributed agents enough time to explore from demonstration states. However, we emphasize that tasks that exhibit greater diversity between episodes or are compositionally complex (such as the full version of our simulated tasks) will extend this gap, needing more supervisory data. Similarly, policy learning methods that leverage the demonstrations in more sophisticated ways might also overcome these limitations. Nevertheless, our findings indicate that a large and diverse dataset of demonstrations can encourage agents to explore and learn more effectively.

5 Conclusion

We presented ROBOTURK, a platform that supports large-scale on-demand data collection for robot learning tasks. We conducted a user study to evaluate the efficacy of our iPhone user interface and WebRTC-based communication backend, and also demonstrated that our platform is able to support real-time trans-Pacific communication and control without a significant difference in performance and task completion times. These results suggest that ROBOTURK can be used by a significant portion of the worldwide population, regardless of geographic location.

As an initial pilot for ROBOTURK, we collected a large pilot dataset consisting of over 2200 total successful demonstrations in only 22 hours of total platform usage on the *Bin Picking* and *Nut-and-peg Assembly* tasks. We used demonstrations collected from our platform to train policies on the *Bin Picking (Can)* and *Nut-and-peg Assembly (Round)* tasks, which are simplified variants of the original tasks, showed that the data allows the policies to solve the tasks with reinforcement learning from sparse rewards, and found that performance generally improves with the quantity of demonstrations used. Our final model is a very simple way to utilize the demonstrations by controlling the start state distribution that the agent sees during training — we envision that more sophisticated algorithms can better leverage the demonstrations we collected. Future work will involve utilizing the platform to collect more data on a larger collection of diverse tasks, extending the platform to support remote teleoperation of real robot arms, and developing more sophisticated algorithms that can leverage large quantities of data collected for policy learning. For additional results, videos, and to download our pilot dataset, visit `roboturk.stanford.edu`.

Acknowledgments

We thank DataTang for supporting our data collection efforts and making it possible for us to collect our pilot dataset. We acknowledge the support of ONR - MURI (1175361-6-TDFEK), Toyota Research Institute (S-2015-09-Garg), Weichai America Corp.(1208338-1-GWMZY), CloudMinds Technology Inc. (1203462-1-GWMXN), Nvidia (1200225-1-GWMVU), Toyota (1186781-31-UBLFA), Panasonic, and Google Cloud. We would also like to thank members of the Stanford People, AI & Robots (PAIR) group (pair.stanford.edu) and the anonymous reviewers for their constructive feedback. We thank Pengda Liu for help with infrastructure.

References

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognitio*, pages 248–255, 2009.

[2] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, 2014.

[3] P. Rajpurkar, R. Jia, and P. Liang. Know What You Don't Know: Unanswerable Questions for SQuAD. *arXiv preprint arXiv:1806.03822*, 2018.

[4] K.-T. Yu, M. Bauza, N. Fazeli, and A. Rodriguez. More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing. In *Int'l Conference on Intelligent Robots and Systems*, 2016.

[5] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen. Learning hand-eye coordination for robotic grasping with large-scale data collection. In *ISER*, pages 173–184, 2016.

[6] K. Fang, Y. Zhu, A. Garg, A. Kurenkov, V. Mehta, L. Fei-Fei, and S. Savarese. Learning task-oriented grasping for tool manipulation from simulated self-supervision. In *Robotics: Systems and Science*, 2018.

[7] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *arXiv preprint arXiv:1703.09312*, 2017.

[8] A. Kasper, Z. Xue, and R. Dillmann. The kit object models database: An object model database for object recognition, localization and manipulation in service robotics. *The International Journal of Robotics Research*, 31(8):927–934, 2012.

[9] C. Goldfeder, M. Ciocarlie, H. Dang, and P. K. Allen. The columbia grasp database. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 1710–1716. IEEE, 2009.

[10] L. Pinto and A. Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016.

[11] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

[12] P. Abbeel and A. Y. Ng. Inverse reinforcement learning. In *Encyclopedia of machine learning*, pages 554–558. Springer, 2011.

[13] A. Y. Ng, S. J. Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, pages 663–670, 2000.

[14] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[15] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

[16] P. Abbeel, A. Coates, and A. Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639, 2010.

[17] M. Večerík, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

[18] S. Krishnan, A. Garg, R. Liaw, B. Thananjeyan, L. Miller, F. T. Pokorny, and K. Goldberg. Swirl: A sequential windowed inverse reinforcement learning algorithm for robot tasks with delayed rewards. In *Workshop on Algorithmic Foundations of Robotics (WAFR)*, 2016.

[19] A. Gupta, C. Eppner, S. Levine, and P. Abbeel. Learning dexterous manipulation for a soft robotic hand from human demonstrations. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 3786–3793. IEEE, 2016.

[20] A. Boularias, J. Kober, and J. Peters. Relative entropy inverse reinforcement learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 182–189, 2011.

[21] J. Ho and S. Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573, 2016.

[22] S. Osentoski, C. Crick, G. Jay, and O. C. Jenkins. Crowdsourcing for closed loop control. In *Proc. of the NIPS Workshop on Computational Social Science and the Wisdom of Crowds, NIPS*, 2010.

[23] K. Goldberg. Beyond the web: Excavating the real world via mosaic. In *Second International WWW Conference*, 1994.

[24] J. Sung, S. H. Jin, and A. Saxena. Robobarista: Object part based transfer of manipulation trajectories from crowd-sourcing in 3d pointclouds. In *Robotics Research*, pages 701–720. Springer, 2018.

[25] P. F. Hokayem and M. W. Spong. Bilateral teleoperation: An historical survey. *Automatica*, 42(12): 2035–2057, 2006.

[26] D. Kent, C. Saldanha, and S. Chernova. A comparison of remote robot teleoperation interfaces for general object manipulation. In *HRI*, pages 371–379. ACM, 2017.

[27] A. Leeper, K. Hsiao, M. Ciocarlie, L. Takayama, and D. Gossow. Strategies for human-in-the-loop robotic grasping. In *International Conference on Human-Robot Interaction*, pages 1–8. IEEE, 2012.

[28] M. Laskey, C. Chuck, J. Lee, J. Mahler, S. Krishnan, K. Jamieson, A. Dragan, and K. Goldberg. Comparing human-centric and robot-centric sampling for robot deep learning from demonstrations. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 358–365. IEEE, 2017.

[29] A. Dragan and S. Srinivasa. Online customization of teleoperation interfaces. In *21st IEEE International Symposium on Robot and Human Interactive Communication (Ro-Man)*, 2012.

[30] Y. Zhu, Z. Wang, J. Merel, A. Rusu, T. Erez, S. Cabi, S. Tunyasuvunakool, J. Kramár, R. Hadsell, N. de Freitas, et al. Reinforcement and imitation learning for diverse visuomotor skills. In *RSS*, 2018.

[31] B. Akgun, K. Subramanian, and A. L. Thomaz. Novel interaction strategies for learning from teleoperation. In *AAAI Fall Symposium: Robots Learning Interactively from Human Teachers*, 2012.

[32] J. Liang, J. Mahler, M. Laskey, P. Li, and K. Goldberg. Using dvrk teleoperation to facilitate deep learning of automation tasks for an industrial robot. In *CASE*, Aug 2017.

[33] D. Whitney, E. Rosen, E. Phillips, G. Konidaris, and S. Tellex. Comparing robot grasping teleoperation across desktop and virtual reality with ros reality. *Int'l Symp. of Robotics Research*, 2017.

[34] T. Zhang, Z. McCarthy, O. Jow, D. Lee, K. Goldberg, and P. Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. *arXiv preprint arXiv:1710.04615*, 2017.

[35] J. I. Lipton, A. J. Fay, and D. Rus. Baxter's homunculus: Virtual reality spaces for teleoperation in manufacturing. *IEEE Robotics and Automation Letters*, 3(1):179–186, 2018.

[36] B. Akgun, M. Cakmak, K. Jiang, and A. L. Thomaz. Keyframe-based learning from demonstration. *International Journal of Social Robotics*, 2012.

[37] Superdata research. 2017. Virtual Reality Consumers. Technical Report., 2017.

[38] Apple inc. annual report to US SEBI, 2017.

[39] L. Fan, Y. Zhu, J. Zhu, Z. Liu, O. Zeng, A. Gupta, J. Creus-Costa, S. Savarese, and L. Fei-Fei. Surreal: Open-source reinforcement learning framework and robot manipulation benchmark. In *Conference on Robot Learning*, 2018.

[40] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

[41] Industrial robotics category. `http://worldrobotsummit.org/en/wrc2018/industrial/`.

[42] K. Winstein, A. Sivaraman, and H. Balakrishnan. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI '13)*, pages 459–471, 2013.

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

[44] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, A. Eslami, M. Riedmiller, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[45] S. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In *ICML*, 2002.

[46] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel. Overcoming exploration in reinforcement learning with demonstrations. 2018.

[47] C. Resnick, R. Raileanu, S. Kapoor, A. Peysakhovich, K. Cho, and J. Bruna. Backplay:" man muss immer umkehren". *arXiv preprint arXiv:1807.06919*, 2018.

A Appendix

A.1 Simulation Environment Details

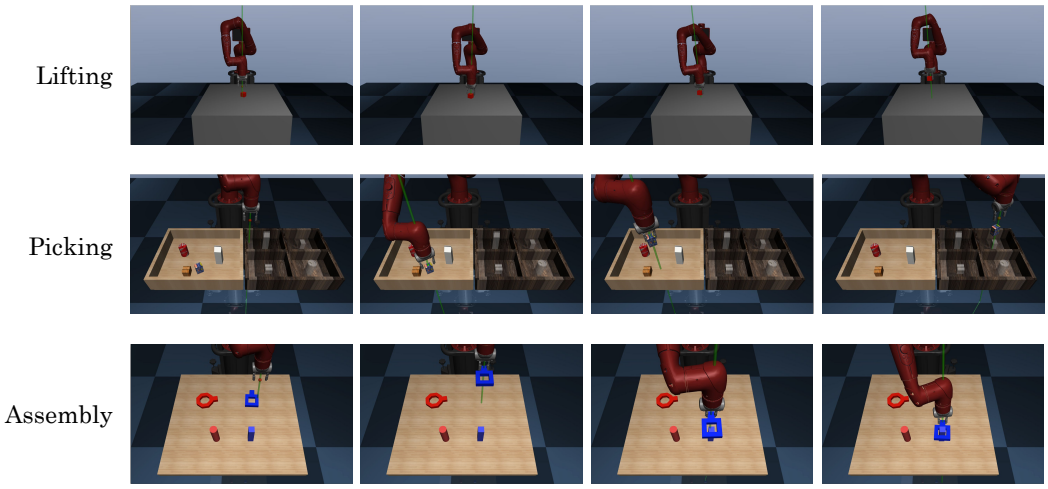


Figure 7: **Task Details**: A detailed depiction of the tasks used to evaluate our system. Each simulated manipulation environment contains a 7-DoF Sawyer robot arm and various objects. In the *lifting* task (top row) the objective is to control the robot arm to grab the cube and lift it. In the *picking* task (middle row) the objective is to place each object into its corresponding bin. In the *assembly* task (bottom row), the objective is to place a round nut and square nut onto their corresponding pegs. These screenshots were taken along successful demonstrations of each task.

Environment design. In this work, we aim at demonstrating the effectiveness of our system in challenging multi-staged tasks which require both low-level dexterity and high-level reasoning. We evaluate our system in three simulated manipulation tasks with the Sawyer robot. The manipulation tasks were developed in the MuJoCo physics engine [40], which provides a high-speed simulation of contact dynamics with rigid 3D objects. The robot interacts with a set of objects in the workspace. Each operator observes the robot workspace from a camera facing the robot.

We describe the three tasks in Fig. 7. First, we use a *lifting* task as a diagnostic example to evaluate our choice of control interface and our system's characteristics. We also evaluate our system with two complex tasks: the *picking* task and *assembly* task. We perform large-scale data collection and model training on the *picking* task and the *assembly* task. These tasks involve a variety of dexterous manipulation skills, such as grasping, pick-and-place, nut assembly, and high-level perception tasks of understanding object categories and geometries.

The above tasks resemble subtasks necessary for robot automation of part assembly. These tasks were inspired by the assembly challenge in the industrial robotics category of the World Robot Summit [41]. The *picking* task is similar to the kitting task, where parts need to be picked and laid out in preparation for assembly, and the *assembly* task is similar to the task board task, where nuts must be placed onto designated pegs on a board. Fig. 7 shows screenshots taken along a successful demonstration of each of the three tasks.

A.2 UI Experiments: Lifting Task

In Sec. 3 we described the UI and network experiments we performed. We ran experiments testing four different user interfaces on both the *lifting* and *picking* tasks. Here, we describe the results of these experiments on the *lifting* task. As Fig. 8 shows, users took significantly longer to complete the lifting task using the keyboard interface as compared to the other three interfaces, which all perform roughly equally. The lifting task is relatively simple and every episode of interaction is around 10 seconds, which was too short to reveal any significant differences between the phone, VR, and 3D mouse interfaces.

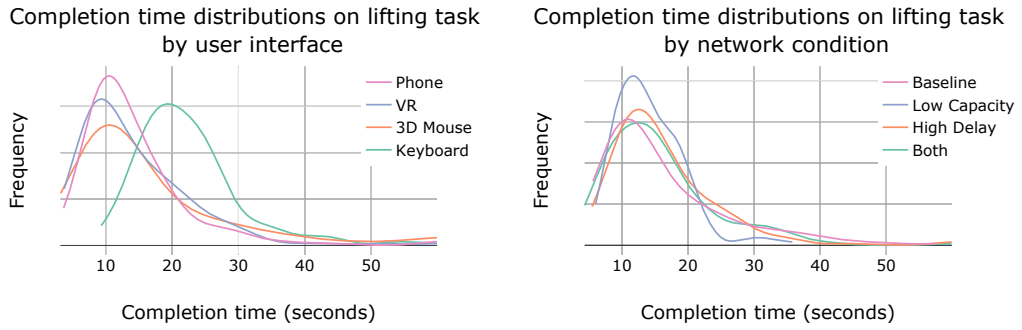


Figure 8: Comparison of completion time distributions of different user interfaces on the *lifting* task.

A.3 UI Experiments: Details

In Sec. 3 and Sec. A.2 we described the general trends that we found during our UI experiments. In Table 2 and Table 3, we detail the concrete results of those same experiments.

A.4 Network Experiments

We ran tests on four different network conditions, with bandwidth at 2.4Mbps or 500kbps and delay at 20ms or 120ms. The *baseline* condition had 2.4Mbps bandwidth and 20ms delay. The *low capacity* condition had bandwidth reduced to 500kbps and the *high delay* condition had delay increased to 120ms. The *both* condition had the worst bandwidth and delay. Each user completed the *lifting* and *picking* task at varying network condition levels. Fig. 8 shows how similar the completion time distributions were for the lifting task, and Table 4 shows the same results with the mean and standard deviation for each distribution.

Table 6 shows the KolmogorovSmirnov statistic between distributions of completion times for each of our network experiments. Based on a statistical significance level of 5%, we found no statistically significant difference in performance between network conditions, despite worsened network conditions. This is largely due to WebRTC's adaptive video compression keeping quality high and delays at a minimum. Table 5 shows the mean and standard deviation for each distribution compared in Table 6.

A.5 Analysis of object difficulty on the Bin Picking task

For the *picking* task, we compared the time to grasp a target object and the time to place an object in a bin across the different user control interfaces. Fig. 9 shows a qualitative comparison of how users struggled with the various objects and interfaces. Fig. 10 implies that milk tended to be the easiest object to grasp for all interfaces, while the rest of the objects were comparably difficult. Furthermore, we found that the phone experiments tended to have the most consistent results while the keyboard experiments showed a large increase in completion times.

A.6 Teleoperation Controller Details

The teleoperation server maps received user phone poses and controls the simulated robot arm appropriately. Upon receiving a new phone pose with respect to the AR world coordinates of the phone, the controller matches this to a desired robot end effector pose. Next, an Inverse Kinematics procedure is invoked to compute a desired set point for the robot joint positions $\mathbf{q}^*$. These target joint positions are fed to a controller which computes joint velocities $\dot{\mathbf{q}} = -k_v(\mathbf{q} - \mathbf{q}^*)$ and uses them to control the arm.

A.7 Pilot Dataset Distribution

Using RoboTurk, we were able to collect a total of 3224 demonstrations over the course of 22 hours of total system usage. A total of 2218 demonstrations were successful and included in the dataest with 1171 demonstrations on the *picking* task and 1147 on the *assembly* task. These successful demonstrations correspond to 137.5 hours of trajectories.

Table 2: Statistics on completion times of the *lifting* task across different user interfaces.

	Mean	Standard Deviation	Num. Demonstrations
Keyboard	22.34	7.92	187
3D Mouse	16.58	12.27	193
VR Controller	14.29	8.38	160
Phone	14.26	8.27	167

Table 3: Statistics on completion times of the *picking* task across different user interfaces.

	Mean	Standard Deviation	Num. Demonstrations
Keyboard	151.45	23.69	40
3D Mouse	112.57	43.65	40
VR Controller	79.36	29.41	40
Phone	89.97	34.94	40

Table 4: Statistics on completion times of the *lifting* task across different network conditions.

	Mean	Standard Deviation	Num. Demonstrations
Baseline	15.66	8.22	160
Low Capacity	16.13	9.41	168
High Delay	14.19	5.07	164
Both	16.31	8.29	160

Table 5: Statistics on completion times of the *picking* task across different network conditions.

	Mean	Standard Deviation	Num. Demonstrations
Baseline	111.54	39.84	40
Low Capacity	113.93	51.62	40
High Delay	106.4236	34.72	40
Both	113.85	41.81	41

Table 6: Kolmogorov-Smirnov statistic between distributions and p-value for completion times in the *picking* task across the different network configurations.

	Baseline	Low Capacity	High Delay	Both
Baseline	-	0.125 (0.893)	0.165 (0.598)	0.150 (0.724)
Low Capacity	0.125 (0.893)	-	0.165 (0.603)	0.150 (0.724)
High Delay	0.165 (0.598)	0.165 (0.603)	-	0.077 (0.999)
Both	0.150 (0.724)	0.150 (0.724)	0.077 (0.999)	-

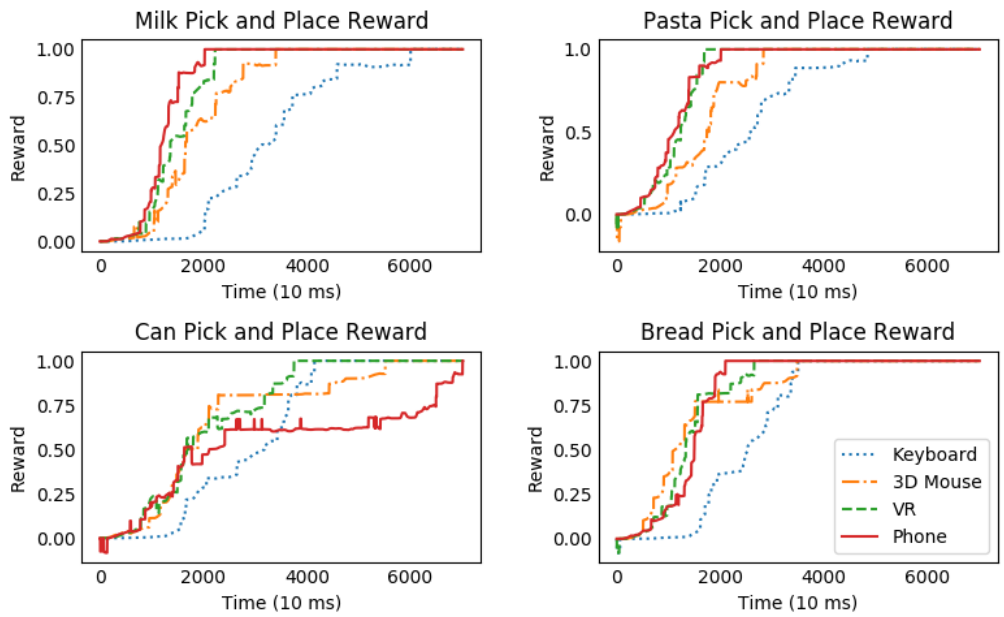


Figure 9: Comparison of environment rewards, which measure the completion of the pick and place operation.

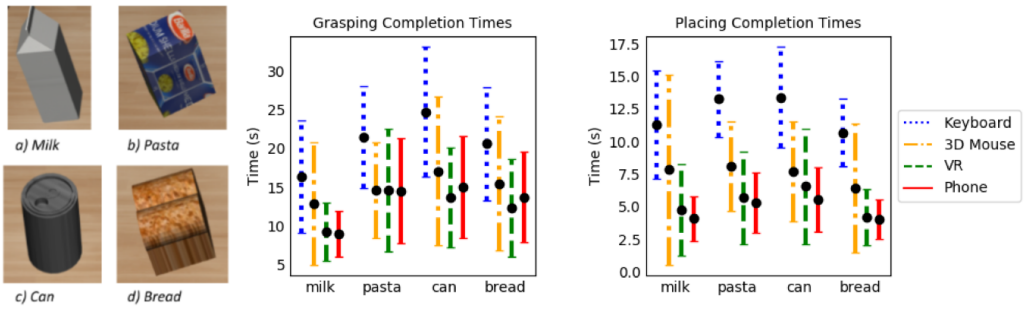


Figure 10: Time taken to grasp and place the four different types of objects in the pick and place task averaged across eight users for different user interfaces.

The mean completion time for the *assembly* task is 112.12 seconds, with a standard deviation of 59.95. The mean completion time for the *picking* task is 146.71 seconds, with a standard deviation of 52.67. The distribution of completion times in this dataset is shown in Fig.11.

A.8 Policy Learning From Demonstrations

We ran our policy learning experiments on the *can picking* and *round assembly* tasks. For both tasks, the agent is given 100 time steps to solve the task, and is provided a sparse reward that is 1 on task completion.

We used the distributed PPO implementation from [39] with 32 actors to train policies. We used neural networks with an LSTM layer of size 100 followed by two fully-connected layers of size 300 and 200 for both the actor network (mean) and the value function network. The policy is a Gaussian distribution parametrized by the mean network and a separate log standard deviation parameter. See [39] for additional hyperparameter details.

On every environment reset, with 90% probability, a demonstration was sampled uniformly at random from the demonstration set, and the simulator was reset to a state sampled uniformly from the selected demonstration, and with 10% probability, the environment was reset normally.

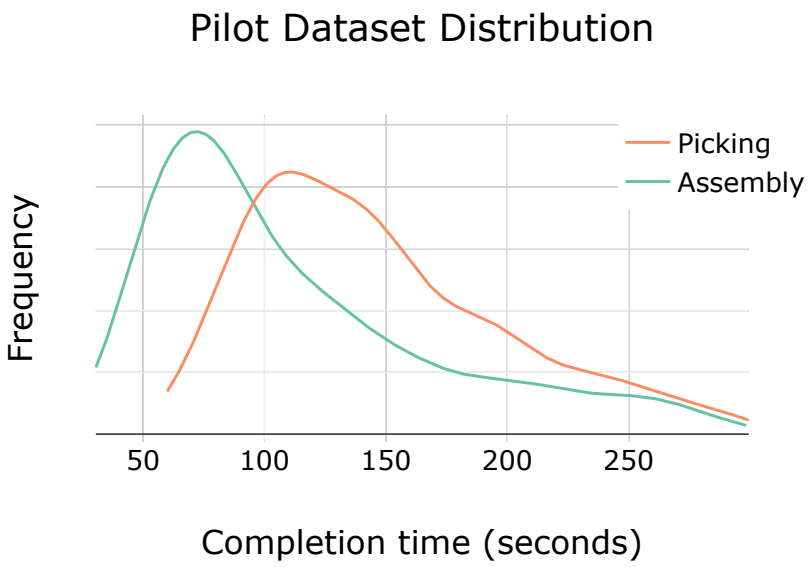


Figure 11: **RoboTurk Pilot Dataset Distribution**: Distribution of task completion times across all users in the large-scale dataset.

Experiments on the *can picking* task were each given 24 hours of training time, while those on the *round assembly* task were given 48 hours of training time. These cutoff times were experimentally determined to be sufficient for training policies to complete the tasks.