A JavaScript Framework for Crowdsourced Human-Robot Interaction Experiments: *RemoteHRI*

Finley Lau,[1] * **Deepak Gopinath,** [2,3] * **Brenna D. Argall**[1,2,3]
* Equal contribution
[1]Department of Computer Science, Northwestern University, Evanston, Illinois
[2]Department of Mechanical Engineering, Northwestern University, Evanston, Illinois
[3]Shirley Ryan AbilityLab, Chicago, Illinois
finleylau2021@u.northwestern.edu, deepakgopinath@u.northwestern.edu, brenna.argall@northwestern.edu

Abstract

In this paper, we present *RemoteHRI*, a JavaScript (JS) software framework for conducting Human-Robot Interaction (HRI) experiments in a web browser. Built with HRI researchers in mind, *RemoteHRI* includes a flexible set of software tools that allows for rapid prototyping and quick deployment of a wide range of laboratory-like experiments that can be run online. *RemoteHRI* uses the state-of-the-art ReactJS[1] framework to build standard HRI stimulus environments such as grid worlds, differential drive cars, and robotic arms. As a result, the researcher can solely focus on the experimental design thereby saving valuable time and effort. Code for *RemoteHRI* is available at https://github.com/argallab/RemoteHRI.

1 Introduction

Evaluating progress in science and engineering heavily depends on proper experimentation protocols. In the domain of Human-Robot Interaction (HRI), proper experiments are needed both to understand the human decision-making process while interacting with robots and to evaluate the success of robotics autonomy algorithms in interacting with humans and other entities.

In an ideal situation, researchers conduct HRI experiments with real robotic systems. This provides researchers with rich data that encodes the sensing and actuating complexities of robot operation in the real world. However, designing and conducting HRI experiments on real robotic systems come with a great deal of challenges, especially in the academic setting.

First, real robotic systems are expensive, and academic labs rarely own multiple robots of the same type. This drastically limits the number of studies that could be run in parallel, thereby making data collection extremely slow. Second, subject recruitment for academic studies can suffer from biases due to lack of diversity in the recruitment pool. Third, in-person subject studies typically require researchers to be in close contact with the subjects. However, in light of COVID-19, academic researchers are now required to maintain stricter social distancing protocols, rendering an in-person study practically impossible. Particularly, in the subdomain of assistive robotics, the end-user population (people with motor-impairments as a result of trauma or neurodegenerative disease) typically belongs to a high-risk group. Therefore, participation in an in-person study can pose significant health risks.

By contrast, simulation-based experiments are less expensive and can enhance the usefulness of in-person studies. By conducting experiments online, researchers can parallelize data collection and achieve greater diversity in the recruitment pool. Furthermore, online settings remove the need for in-person supervision, alleviating the health risks of being in close proximity to others. Exploratory HRI studies conducted in a simulated environment can also provide strong priors, in terms of baseline models for human behavior and initial evaluation of robotics algorithms. These studies additionally inform resourceful design of a real-world experiment. Performing human subject studies in a simulated environment is not novel. For example, *jsPsych* (De Leeuw 2015) is a popular JavaScript library used by researchers in the field of psychology to perform simple online behavioral experiments. However, the stimulus plugins available in *jsPsych* do not cover the full space of rich and complex stimuli needed for HRI. The field of HRI research currently lacks a comprehensive software framework for conducting simulation-based experiments and relies on one-off solutions developed for specific projects, leading to wasted time and repeated effort. Although primarily used for reinforcement learning research, OpenAI's Gym (Brockman et al. 2016) provides a suite of simulated environments that are widely used for HRI experiments (Schaff and Walter 2020; Broad et al. 2020); however, the JS interface needed for crowd-sourced online studies is still under development.

RemoteHRI provides HRI researchers a set of tools to design and conduct common HRI studies in a seamless manner. The framework is designed with the researcher in mind, specifically focusing on ease of use and rapid prototyping. *RemoteHRI* derives its inspiration from frameworks such as *jsPsych* and supports flexible experiment design, an easy-to-use researcher interface, and prepackaged standard HRI stimulus environments. *RemoteHRI* uses ReactJS for

[1]https://reactjs.org/

client-side rendering, which allows for increased flexibility, speed, and ease of development through its management of state and structural organization of components (groups of elements displayed on the screen) as compared to vanilla JavaScript.

The key features of *RemoteHRI* are as follows:

- **Researcher-Centric Design**: *RemoteHRI* was built with researchers from different backgrounds and programming experience in mind and provides a low entry point for easily prototyping and building HRI experiments, without having to develop the client-side application (which typically requires researchers to learn to use graphics APIs, and physics libraries) or the server-side application.
- **Modular/Extensible**: *RemoteHRI* is highly modular in that the different applications of the framework operate independently of each other and may be replaced with an equivalent module. For example, the *RemoteHRI* server could be replaced with a custom-built server without affecting the client-side application. Furthermore, implementation of different stimulus types is similar and therefore the framework can be easily extended to work with new types of stimuli.
- **Unified Researcher Interface**: Regardless of the specific experimental domain or design choice, the researcher specifies experimental flow through a single JSON file, which we refer to as *Experiment.json* (discussed more in detail in Section 4). *RemoteHRI* provides a GUI interface and a set of utility functions allowing researchers to quickly generate *Experiment.json*.
- **Plug and Play**: *RemoteHRI* experiments are completely controlled by the *Experiment.json* file. By specifying the stimuli and their respective properties used in the experiment in *Experiment.json*, researchers can create entire experiments without changing the client-side application.

In Section 2 we describe a typical HRI experiment design and how it informs some of the design choices for *RemoteHRI*. The main software modules that constitute *RemoteHRI* are described in Section 3. The researcher interface for experiment specification is presented in Section 4, followed by conclusions and future work in Section 5.

2 HRI Experiment Design

A quick analysis of different types of experiments conducted in the field of HRI reveals various commonalities in experimental design (Ghassemi et al. 2019; Breazeal et al. 2005; Javaremi, Young, and Argall 2019; Tsui et al. 2013; Gopinath, Jain, and Argall 2016). We identity four different phases in the majority of these experiments:

- **Consenting Phase**: This is common in any experiment done in an academic setting. During this phase, the researcher describes the experiment in detail to the subject (either in writing or verbally). After evaluating the risks and benefits of the experiment, the subject chooses whether to participate in the study. During this phase, researchers typically also collect non-identifiable demographic and subject-specific information, such as age, gender, and race.
- **Training Phase**: In the training phase, the participant gets familiarized with the experimental setup. This phase also helps researchers establish a baseline performance, which could be a useful measure in data analysis.
- **Testing Phase**: The testing phase typically consists of multiple blocks of experimental trials under different experimental conditions. This phase may serve to collect data (for example, human teleoperation data of robotic manipulators to build data-driven computational models of human decision making) or to evaluate an algorithm's performance (for example, evaluation of a shared control robot policy for assistive robotic manipulators).
- **Survey Phase**: The participant is presented with questions related to their experience of interacting with the robotic system.

Note that in a given experiment, possibly with the exception of the consenting phase, phases may occur multiple times in no particular order. *RemoteHRI* recognizes the need for such combinatorial flexibility in experimental design and offers researchers an easy approach to specify any experimental flow.

3 Framework Modules

RemoteHRI consists of two main software modules: 1) the client-side application and 2) the server-side application. Participants interact with the client-side application in their browsers during an experiment, while the server-side application ensures the proper delivery of experiment content. In the subsequent subsections, we will describe the details of the client-side and the server-side applications.

3.1 Client

The client-side application is responsible for presenting the stimulus for each trial during the course of the experiment. In *RemoteHRI* we consider two main classes of stimuli:

- **Passive:** A passive stimulus is an environment in which a participant 'passively' observes the presented stimuli (audio/video/text). For example, participants could be shown images of two different humanoid robots and asked which one has more human-like features. Participant responses to a passive stimulus depend on the stimulus type and can take many forms such as single answer/multiple choice, multiple answer/multiple choice, 5- or 7-point Likert scale, or free text response. *RemoteHRI* provides a suite of common passive stimuli such as videos, recordings, image and text displays, surveys/questionnaires, *et cetera*.
- **Active:** An active stimulus is an environment in which there is at least one agent that is actuated by the participant via some form of control input, by an autonomous controller, or both. We refer to a stimulus containing only a human-controlled agent as *H-Active* and a stimulus containing only an autonomy-controlled agent as *A-Active*. A stimulus environment containing agent(s) controlled by both a human and an autonomous agent is referred to as *HA-Active*. An active stimulus environment may have one or more active agents. *RemoteHRI* includes implementations of some of the standard active environments used in

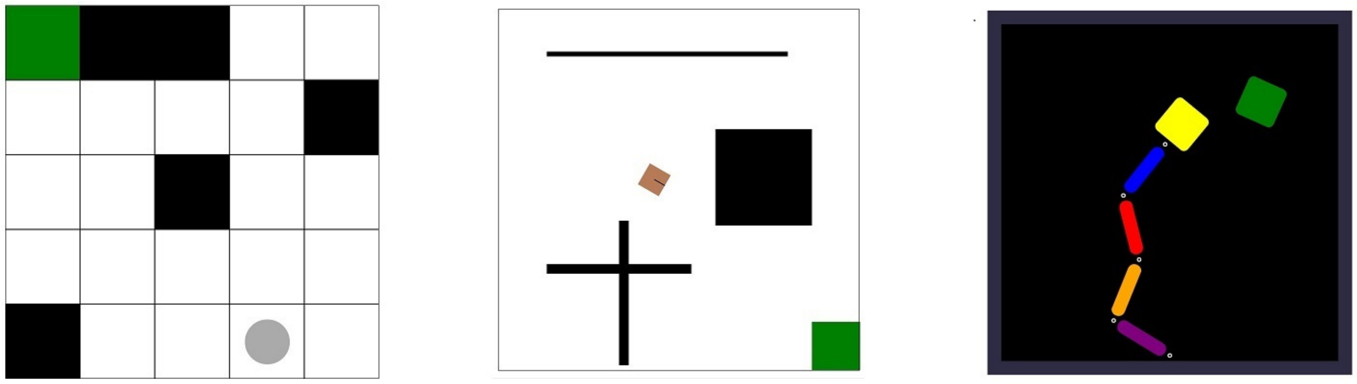


Figure 1: Examples of active stimuli in *RemoteHRI*. *Left*: *DiscreteGridWorld* stimulus with static obstacles. *Middle*: *DifferentialDriveRobot* stimulus with static obstacles for navigation tasks. *Right*: *RoboticArm* stimulus for reaching tasks. The goal states are indicated in green.

simulated HRI experiments, such as discrete grid-worlds, differential-drive robots, and planar robotic arms. The implementation of these standard environments is shown in Figure 1.

Environments can also be classified based on whether an active agent is operating in a static (*S-Env*) or dynamic (*D-Env*) environment. For example, a human-controlled point robot operating in a discrete grid-world with moving obstacles is an *H-Active/D-Env* stimulus whereas an autonomy-controlled planar robotic arm performing reaching motion towards a fixed goal location is an *A-Active/S-Env* stimulus.

RemoteHRI comes with standard implementations of planning and control algorithms to autonomously control agents in an active environment. Similarly the framework also implements simple algorithms such as random walks and periodic motion for the control of dynamic aspects of the environment. The researcher can readily activate any of these implemented algorithms in *Experiment.json*.

Implementation details: The ReactJS client-side application consists of a modular component structure allowing variable display of stimuli on the screen (Figure 2).

An experiment is rendered through a React component called *Content*, which represents the space on the browser screen belonging to the experiment's content. Depending on the specification in *Experiment.json*, this content can take the form of various stimulus components, such as *DiscreteGridWorld*, *DifferentialDriveRobot*, or *RoboticArm*. Each of these stimulus components contains the same child component structure (*Header/Instructions*, *StimulusView*, *Continuation*), providing for consistency and ease of implementation for new stimulus types. The header/instructions component specifies each trial's title and instructions for the participant. The stimulus view renders the specified stimulus, such as the grid world, differential-drive robot world, or robotic arm world. The continuation component specifies how the participant may proceed to the next trial after completion of a trial, such as by clicking a button or pressing any key.

Since the client-side application is only involved with taking a JSON specification for a trial and rendering the appropriate stimulus on the screen, all experimental flow logic can be abstracted to the server application. This allows for the modular plug-and-play feature of *RemoteHRI*, as researchers can easily use the provided client-side application without modification to render their experiments.

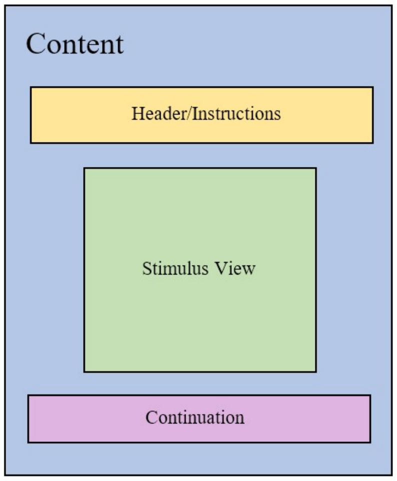


Figure 2: Diagram showing structural rendering layout of the client-side application.

3.2 Server

The experiment flow for each participant in an experiment is managed by the server application. It is built using the *Nodus Ponens* framework, a light, full-stack framework for running high-level reasoning and cognitive science experiments in Node.js.[2]

The key features of the *RemoteHRI* server are as follows:

[2]Code available at https://www.npmjs.com/package/nodus-ponens. Designed by Sangeet Khemlani. Distributed under the Creative Commons License.

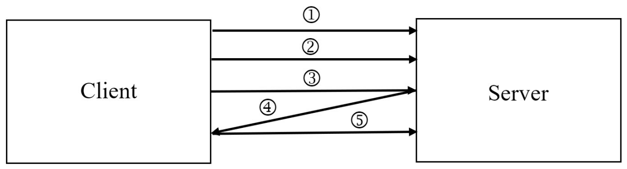


Figure 3: Diagram showing communication between the client-side application and the server. 1) Client sends initial request to server for experiment name. 2) Client sends request to start experiment. 3) Client sends request for first trial. 4) Server responds with JSON data for first trial. 5) Client sends collected trial data back to server, along with request for next trial.

- **Experimental Flow Management**: By using the *Experiment.json* specification, the server is able to construct a list of stimuli trials to present to each participant. It handles randomization and construction of the unique order in which trials are presented to a particular participant. In addition to assigning a unique ID to each participant, it also keeps track of where participants are in the experiment in order to correctly serve subsequent trials.
- **Data Collection**: The server collects both incomplete and complete data on a trial-per-trial basis. After a participant finishes a trial, the server stores the collected data from the client-side application into a JSON file containing all information collected from the current trial as well as any previous ones.
- **Session Management**: The server keeps track of a participant's experiment data through browser sessions. Notably, this allows for participants to reconnect to an experiment and continue from where they left off, if a researcher specifies that this is allowed.

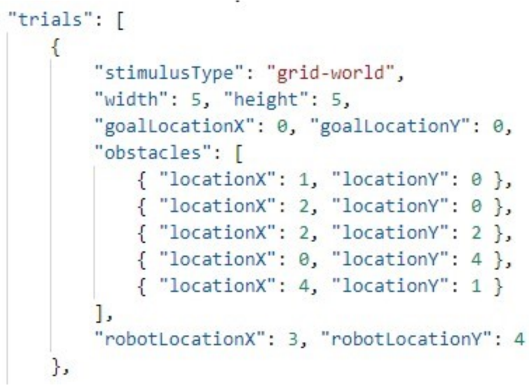
4 Experiment Specification

Experiment.json: This JSON file completely specifies an experiment. It contains information about experiment structure as well as trial specifications.

- **Experiment Structure**: *RemoteHRI* provides functionality to group trials through block-level organization. It also exposes boolean flags to specify whether blocks should be shuffled as well as whether trials within a block should be shuffled. It also allows for specification of trials (such as pre/post block survey questionnaires) preceding and following a trial block, regardless of whether the trials inside of the block are shuffled.
- **Trial Specifications**: *Experiment.json* contains information relevant to rendering the starting state of the stimulus presented in each trial. For example, in a grid world stimulus, *Experiment.json* would specify the width and height of the world and the starting positions of all objects in the world (Figure 4). Each stimulus type has its own JSON schema to completely specify its state.

Experiment.json can be generated by any means to create a JSON file. This can include manually writing a file, a GUI-based tool provided through *RemoteHRI*, or scripts in any programming language that can write to a JSON file. This allows for flexibility and scalability for the researcher to create an experiment, regardless of their programming experience or the scale of their experiment.

The *Experiment.json* interface also allows for minimal configuration when extending *RemoteHRI* with a new stimulus type. Due to the freedom in schema provided by JSON files, a researcher only needs to implement the client-side rendering of their new stimulus using ReactJS and decide what properties may be specified in *Experiment.json*.



```
"trials": [
    {
        "stimulusType": "grid-world",
        "width": 5, "height": 5,
        "goalLocationX": 0, "goalLocationY": 0,
        "obstacles": [
            { "locationX": 1, "locationY": 0 },
            { "locationX": 2, "locationY": 0 },
            { "locationX": 2, "locationY": 2 },
            { "locationX": 0, "locationY": 4 },
            { "locationX": 4, "locationY": 1 }
        ],
        "robotLocationX": 3, "robotLocationY": 4
    },
```

Figure 4: Snapshot of a possible *Experiment.json* specification for a grid-world experiment. The configuration specified corresponds to the *DiscreteGridWorld* in Figure 1.

5 Conclusions and Future Work

In this paper, we presented *RemoteHRI*, a JavaScript framework for designing and deploying crowdsourced HRI experiments online. We anticipate that *RemoteHRI* can also help in lowering the barrier to entry for HRI research and increase equity in terms of broadened participation from underrepresented HRI researchers. Future extensions of *RemoteHRI* will include functionality to use custom input devices such as joysticks and sip-and-puff (Farnet, McWilliams, and Stutz 2008) in addition to standard keyboard and mouse input. This specifically targets the subdomain of assistive robotics in which studies with motor-impaired subjects are of paramount importance for the successful adoption of assistive technologies. Future iterations of the framework will also include 3D simulated environments.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant CNS 1544741. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the aforementioned institutions.

References

Breazeal, C.; Kidd, C. D.; Thomaz, A. L.; Hoffman, G.; and Berlin, M. 2005. Effects of nonverbal communication on efficiency and robustness in human-robot teamwork. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 708–713. IEEE.

Broad, A.; Abraham, I.; Murphey, T.; and Argall, B. 2020. Data-driven koopman operators for model-based shared control of human–machine systems. *The International Journal of Robotics Research* 0278364920921935.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI gym. *arXiv preprint arXiv:1606.01540*.

De Leeuw, J. R. 2015. jsPsych: A JavaScript ibrary for creating behavioral experiments in a web browser. *Behavior Research Methods* 47(1):1–12.

Farnet, M. G.; McWilliams, H. R.; and Stutz, J. J. 2008. Sip and puff mouse. US Patent 7,412,891.

Ghassemi, M.; Triandafilou, K.; Barry, A.; Stoykov, M. E.; Roth, E.; Mussa-Ivaldi, F. A.; Kamper, D. G.; and Ranganathan, R. 2019. Development of an EMG-controlled serious game for rehabilitation. *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 27(2):283–292.

Gopinath, D.; Jain, S.; and Argall, B. D. 2016. Human-in-the-loop optimization of shared autonomy in assistive robotics. *IEEE Robotics and Automation Letters* 2(1):247–254.

Javaremi, M. N.; Young, M.; and Argall, B. D. 2019. Interface operation and implications for shared-control assistive robots. In *2019 IEEE 16th International Conference on Rehabilitation Robotics (ICORR)*, 232–239. IEEE.

Schaff, C., and Walter, M. R. 2020. Residual policy learning for shared autonomy. *arXiv preprint arXiv:2004.05097*.

Tsui, K. M.; Flynn, K.; McHugh, A.; Yanco, H. A.; and Kontak, D. 2013. Designing speech-based interfaces for telepresence robots for people with disabilities. In *2013 IEEE 13th International Conference on Rehabilitation Robotics (ICORR)*, 1–8. IEEE.