Towards Automated Execution and Evaluation of Simulated Prototype HRI Experiments

Florian Lier
Cognitive Interaction Technology Center of Excellence
Bielefeld University
Bielefeld, Germany
flier@techfak.uni-bielefeld.de

Ingo Lütkebohle
Machine Learning and Robotics Lab
University of Stuttgart
Stuttgart, Germany
ingo.luetkebohle@ipvs.uni-stuttgart.de

Sven Wachsmuth
Cognitive Interaction Technology Center of Excellence
Bielefeld University
Bielefeld, Germany
swachsmu@techfak.uni-bielefeld.de

ABSTRACT

[PRE PRINT VERSION] Autonomous robots are highly relevant targets for interaction studies, but can exhibit behavioral variability that confounds experimental validity. Currently, testing on real systems is the only means to prevent this, but remains very labour-intensive and often happens too late. To improve this situation, we are working towards early testing by means of *partial* simulation, with automated assessment, and based upon continuous software integration to prevent regressions. We will introduce the concept and describe a proof-of-concept that demonstrates fast feedback and coherent experiment results across repeated trials.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*Designs, Program Verification*

Keywords

Human-Robot Interaction, Simulation, Testing, Continuous Integration, System Evaluation

1. INTRODUCTION

Most robots are expected to eventually possess a degree of autonomy that improves their capability of achieving tasks and therefore, performing interaction on autonomous systems – with all their associated technical issues – is of high relevance. It is a well-established fact that valid experiments need a well-defined experimental protocol[3, 1]. However, today's state of the art autonomous systems may exhibit surprisingly high variability in the face of relatively small changes in the environment and user behavior, which could confound data interpretation. To control this, detailed testing of experiment designs and their corresponding software realizations is required but difficult to maintain, especially for experimenters who are not part of the engineering team. To enable experimenters as well as system developers to cooperatively design, test and integrate their experiments as easily and often as possible, and thus to improve the design and evaluation process, we propose *a)* to establish experiment prototyping. This uses simulation environments including a virtual human component *b)* to extend the concept of an experiment protocol to the orchestration of software components, and *c)* to execute and assess the results of a prototype experiment in an automated, easy-to-use fashion. In the remainder of this late breaking report we will introduce a software tool chain which implements our propositions and a purposefully limited scenario in order to present preliminary results.

2. SOFTWARE TOOL CHAIN

Our tool chain comprises three components: an interactive simulation environment (Section 3), a framework to automatically bootstrap a prototype system, verify correct execution of components, asses results gathered from experiments (Section 4) and a Continuous Integration [2] server to centralize experiment evaluation. This tool chain provides the following benefits: simulation environments offer easy and cheap access to data which is usually acquired in labour-intensive "real life" experiments. Moreover, experimenters and engineers can mutually design and discuss prototype experiments "on-the-fly". By providing an easy-to-use, automated and well-defined mechanism to execute and evaluate a prototype system, we minimize the complexity of this task, and make the experiment environment almost instantly available. Experimenters are also able integrate their evaluation methods into the tool chain directly by, e.g., providing MATLAB, R, or Python scripts, which are automatically executed during the experiment evaluation. Lastly, experimenters can easily script the "behavior" of the simulation, by changing a few lines of code without programming experience to explore diverse scenarios and conditions.

3. A SIMULATED HRI SCENARIO

In this purposefully limited scenario, a virtual PR2 robot continuously reports the spacial location of a human avatar in a domestic apartment environment. Both, the robot and the human, are moving about in the scene and eventually meet in front of a table (Figure 1). In order to realize this simulation, we utilized the Modular OpenRobots Simulation

Engine (MORSE) [4]. In this setup, we control the PR2 and the human avatar via ROS [6] middleware. Therefore, we are able to interactively set and publish waypoints in the scene for both agents at runtime. The PR2 is equipped with an abstract sensor (a semantic camera) which reports the human's active coordinates — if the avatar enters the robot's field of view. In this scenario, we have explicitly chosen to simplify the extraction of the human's location by using the semantic camera to obtain a ground truth in the first iteration. However, a virtual human component that is interactively controllable, is a general benefit which enables us to develop more complex scenarios in further iterations.

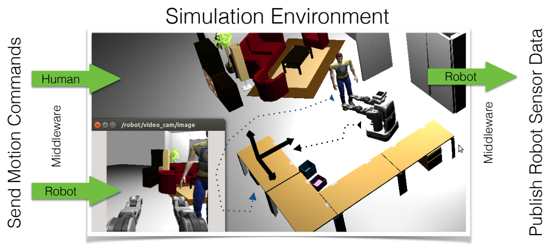

Figure 1: Prototype HRI Simulation

4. SOFTWARE PROTOCOL

Similar to strict adherence of an experiment protocol, the software used in an experiment to produce and assess data, must also follow "protocol" to obtain coherent results. Especially in distributed systems it is crucial to start all software components in the correct sequence, i.e., data producer before consumer, to avoid malfunctions and inconsistent data. Often, software execution/testing is carried out manually and is thus prone to errors. Additionally, experimenters who are "just" interested in the resulting data, must gain additional knowledge about the technical details of a system to conduct their experiments. To address these issues we have developed a finite-state-machine-based testing framework (FSMT) [5] that supports explicit specification of the software ecosystem used, for instance in an experiment. FSMT provides definition of environment variables, component parameter specifications, hierarchical state-based execution of software components, and status (health) checks. Moreover, FSMT specifications consist of three *mandatory* states: environment definition, run state and an assessment state. In the run state, the actual experiment is conducted meaning all components of a system are running and data is recorded. In the assessment state, the recorded data is evaluated by assessment components. Experimenters may provide scripts to assess the data gathered in each run, i.e., plot specified data points. Finally, by implementing this formalization of an experiment protocol with regard to software, we are able to automatically start a software system, verify correct execution and eventually produce consistent results. This approach makes it an ideal candidate for Continuous Integration, but also enables experimenters to run an experiment on their local machine by executing a FSMT specification.

5. RESULTS AND CONCLUSION

To obtain the results depicted in Figure 2, we automatically executed a FSMT specification consisting of ten components, including the simulation as described in (Section 3), on our CI server. In the run state, the simulation is bootstrapped and waypoints are sent to the robot and the human. Simultaneously, the location reported by the robot is recorded to a log file. After twenty seconds the simulation and the motion generation component are stopped. In the assessment state, two components are started: one to clean up and transform the logs in CSV files and a second one to generate plots of the coordinates of the virtual human. To demonstrate the ease of changing the simulation behavior and assessment we conducted three different runs. In the first run we assessed the "x" component of the location, in the second run the "y" component and in the third run we changed one waypoint of the human avatar. Each modification required the change of just 1 line of code. The execution duration of each test lasted 42 seconds. With this setup in place, we are now able to conduct a simulated experiment practically every minute, which will also reflect the impact of changes in the software stack on the desired results. Moreover, experimenters are able to easily add additional evaluation components to this setup. We are aware of the fact, that this setup is basic, but from our point of view, it is a first step towards automated, consistent and cooperative testing of simulated HRI experiments.

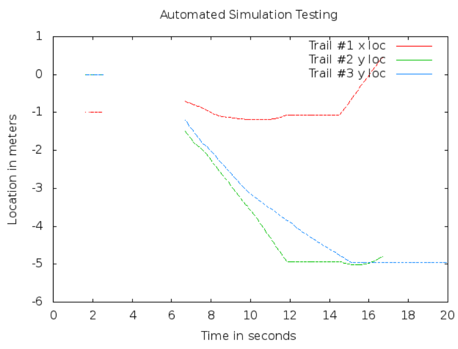

Figure 2: Position of the Human Avatar

6. REFERENCES

[1] K. Dautenhahn. Methodology and themes of human-robot interaction: a growing research field. *International Journal of Advanced Robotic Systems*, 2007.

[2] P. M. Duvall, S. Matyas, and A. Glover. *Continuous integration: improving software quality and reducing risk*. Pearson Education, 2007.

[3] C. D. Kidd and C. Breazeal. Human-robot interaction experiments: Lessons learned. In *Proceeding of AISB*, volume 5, pages 141–142, 2005.

[4] S. Lemaignan, G. Echeverria, M. Karg, J. Mainprice, A. Kirsch, and R. Alami. Human-robot interaction in the morse simulator. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pages 181–182. ACM, 2012.

[5] F. Lier, N. Köster, I. Lütkebohle, and S. Wachsmuth. State machine based simulation testing, 2013.

[6] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, 2009.