Mixed-Reality Robot Behavior Replay: A System Implementation

Zhao Han,[1*] **Tom Williams,**[1] **Holly A. Yanco**[2]

[1]MIRRORLab, Department of Computer Science, Colorado School of Mines, 1500 Illinois St., Golden, CO, USA 80401
[2]HRI Lab, Department of Computer Science, University of Massachusetts Lowell, 1 University Ave., Lowell, MA, USA 01854
zhaohan@mines.edu, twilliams@mines.edu, holly@cs.uml.edu,

Abstract

As robots become increasingly complex, they must explain their behaviors to gain trust and acceptance. However, it may be difficult through verbal explanation alone to fully convey information about past behavior, especially regarding objects no longer present due to robots' or humans' actions. Humans often try to physically mimic past movements to accompany verbal explanations. Inspired by this human-human interaction, we describe the technical implementation of a system for past behavior replay for robots in this tool paper. Specifically, we used Behavior Trees to encode and separate robot behaviors, and schemaless MongoDB to structurally store and query the underlying sensor data and joint control messages for future replay. Our approach generalizes to different types of replays, including both manipulation and navigation replay, and visual (i.e., augmented reality (AR)) and auditory replay. Additionally, we briefly summarize a user study to further provide empirical evidence of its effectiveness and efficiency. Sample code and instructions are available on GitHub at https://github.com/umhan35/robot-behavior-replay.

1 Introduction

Robots used in domains like collaborative manufacturing, warehousing, and assistive living stand to have benefits such as improving productivity, reducing work-related injuries, and increasing the standard of living. Yet the increasingly complexity of the manipulation and navigation tasks needed in these domains can be difficult for users to understand, especially when users need to ascertain the reasons behind robot failures. As such, there is a surge of interest in improving robot understandability by enabling them to explain themselves, e.g., through function annotation (Hayes and Shah 2017), encoder-decoder deep learning framework (Amir, Doshi-Velez, and Sarne 2018), interpretable task representation (Han et al. 2021), and software architecture (Stange et al. 2022). Different dimensions of robot explanations have also been explored, such as proactive explanations (Zhu and Williams 2020), preferred explanations (Han, Phillips, and Yanco 2021), and undesired behaviors (Stange and Kopp 2020). However, these works focused on explaining a robot's current behaviors.

[*]Most of this work was completed while Zhao Han was affiliated with the University of Massachusetts Lowell.

Copyright © 2022, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

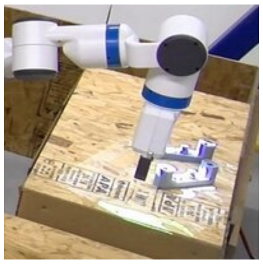

Figure 1: Manipulation replay using the replay technique described in this paper. The robot's arm movement and the green projection (bottom) to indicate the object to be grasped were being replayed to clarify a perception failure: A torn-up wood chip was unknowingly misrecognized as one of the gearbox bottoms. Key frames from the same replay and two other types of replays are illustrated in Figure 2–4.

One challenge within this space is enabling robots to explain their past behavior after their environment has changed. This is an interesting yet challenging problem because objects present in the past might have already been replaced or removed from the scene, making the task of referring to those objects during explanation particularly challenging (see also Han, Rygina, and Williams 2022). Moreover, a robot may not be capable of reasoning and explaining its past behaviors due to unawareness of failures (see Figure 2 and 4), and limited semantic reasoning about objects like ground obstacles or tabletop objects (see also Figure 3).

To help explain a robot's past behaviors, we describe in this tool paper the implementation of a mixed-reality robot behavior replay system that builds on previous work on *Visualization Robots* Virtual Design Elements (VDEs) (Walker et al. 2022). While previous VDEs in this category have primarily sought to visualize future robot behaviors (Rosen et al. 2019), we instead use this technique to visualize previously executed behaviors. The robot behaviors that our technique is capable of replaying generalize to replay of both manipulation and navigation behaviors. (See Figure 2–4). Our replay technique can also handle replay of non-physical cues: verbalization, e.g., sound and speech and visualization, such as projector-based augmented reality (Han et al. 2020b, 2022). Empirical evidence of the effectiveness and efficiency of our approach in explaining past behavior has

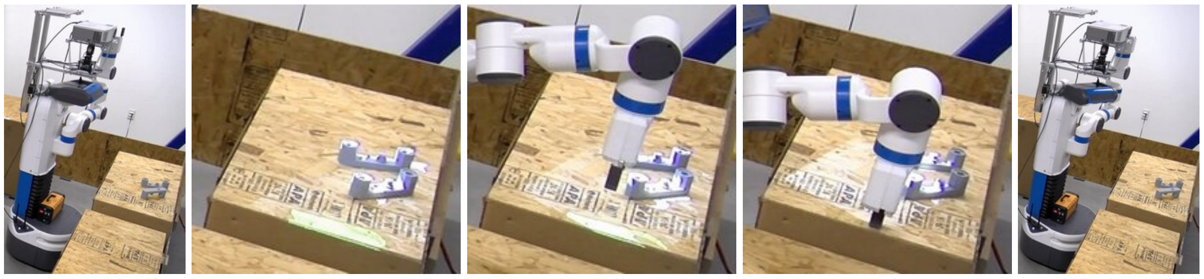
Figure 2: **Manipulation replay of picking a misrecognized object:** Start, perceive, reach above, pick, reset. Both arm movement and AR visualizations are replayed. The rectangular green area (bottom) shows the grasped object. White area, projected onto the two gearbox bottoms, shows correctly recognized objects. (**Video**: https://youtu.be/pj7-LqEsb94)

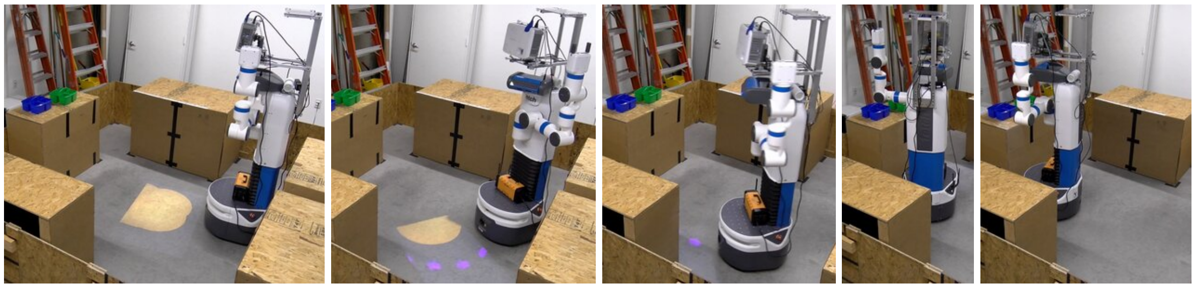
Figure 3: **Navigation replay of a detour path**: Start, rotate, detour, reach position, reach orientation. Both wheel movement and AR visualizations were replayed. Yellow area (spheres of laser scan points; bottom middle) were projected to show ground obstacle, and purple arrows (path poses; bottom) are projected to show past detour path. (**Video**: https://youtu.be/hV6jsA42YYY)

been presented in our previous work (Han and Yanco Under review). While beyond the scope of this tool paper, we will briefly mention the experimental results in Section 4.

We demonstrate our technique on a mobile manipulator Fetch robot (Wise et al. 2016) using the widely-used Robot Operating System (ROS) (Quigley et al. 2009), with the robot behavior encoded in hierarchical behavior trees (Colledanchise and Ögren 2018). Our use of ROS means that our implementation is more-or-less platform agnostic, as most current robots used in research and development have ROS support (OpenRobotics 2022) or bridges (Scheutz et al. 2019).

This work is beneficial to both manipulation and navigation researchers. In addition, our replay technique is helpful for visual debugging for robot developers (Ikeda and Szafir 2022), and for explaining past behaviors to non-expert users.

2 Related Work: Choosing Underlying Technologies

2.1 Robot Data Storage

To replay robot behavior, the first step is to store robot data. One popular tool is rosbag[1], which uses filesystems (bag files) to store and play ROS messages. Despite being persistent on disks, relying on filesystems, compared to databases that we will discuss soon, made it challenging to query specific behaviors for replaying purposes, because related data in different bag files are unstructured and unlinked, requiring writing custom code and logic.

Thus, roboticists have been exploring database technologies. The schemaless MongoDB database is a popular and justified choice among many researches, e.g., Beetz, Mösenlechner, and Tenorth (2010); Niemueller, Lakemeyer, and Srinivasa (2012); Beetz, Tenorth, and Winkler (2015), to store data from sensors or communication messages. Being schemaless allows storing different data types without creating different data structures for different data messages, such as tables in relational Structured Query Language (SQL) databases, e.g., MySQL. In addition to the large number of different robotics data messages, they are often hierarchical/nested and commonly seen in ROS messages, such as the *PoseStamped* message in the geometry_msgs package[2]. The hierarchical *PoseStamped* message contains a *Header* message to include a reference coordinate frame and a timestamp, and a *Pose* message to include a hierarchical *Point* message for position information and a *Quaternion* message for orientation information. It is imaginably tedious to create all these tables for nested data messages one by one. The advantage of schemaless database is also known as minimal configuration, allowing evolving data structures to support innovation and development (Niemueller, Lakemeyer, and Srinivasa 2012). In this work, we used the *mongodb_log* library, open-sourced by Niemueller, Lakemeyer, and Srinivasa (2012), with slight modifications to synchronize timing of different timestamped ROS messages for replay.

[1]https://wiki.ros.org/rosbag

[2]https://wiki.ros.org/geometry_msgs

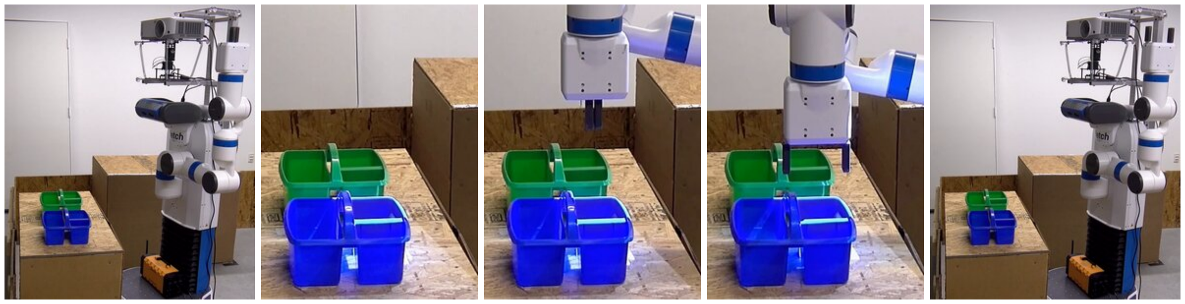

Figure 4: **Manipulation replay of placing (a gearbox bottom) into a wrong caddy section** (the left large section is the correct one): Start, perceive, reach above, place, release, reset. Both arm movement and an AR visualization were replayed. A white cube was projected into the blue caddy's top-right section to show the misrecognized. (**Video**: https://youtu.be/kIJjU2FR4XU)

2.2 Robot Behavior Representation

With a justified choice to use the schemaless MongoDB for robotic data, we then decide how to represent robot behaviors and store related data tied to specific behaviors. A number of methods have been used in prior work to represent sequences of robot actions (Nakawala et al. 2018), including ontologies, state machines, Petri Nets, and behavior trees.

Ontology belongs to the knowledge representation family. Ontologies can be used to infer task specifications from high-level, abstract, underspecified input in a predefined set of actions. Popular implementation include KnowRob (Tenorth and Beetz 2009; Beetz et al. 2018) and CRAM (Beetz, Mösenlechner, and Tenorth 2010). Yet, this approach typically focuses on specification of high-level tasks rather than low-level motion primitives.

Finite state machines (FSM) are a well-established method for modeling computation (Schneider 1990) and have been used for robot task specification and execution, such as the SMACH (State MACHine) library (Bohren and Cousins 2010) and hierarchical RAFCON (Brunner et al. 2016). Although FSM is very flexible at describing task workflow and well-validated, a workflow in FSM can have a significant number of states with intertwined dependencies through transitions, making it hard to maintain, scale and reuse (Colledanchise and Ögren 2018). Particularly for robot behavior replay, it is challenging to clearly separate different robot behaviors.

Petri Nets were created to model concurrency and distributed execution and coordination, which can be seen in multi-robot systems (Ziparo et al. 2008) and soccer robot (Palamara et al. 2008). Although useful, we are more interested in sequential actions and thus leave parallel behavior replay to future work.

Finally, behavior trees (BTs) use tree structures to encapsulate behaviors in different kind of parent control nodes with child execution nodes (Colledanchise and Ögren 2018). BTs are commonly used to model AI agents in games (Lim, Baumgarten, and Colton 2010; Sagredo-Olivenza et al. 2017) and recently have been gaining momentum in robotics, e.g., end-user programming (Paxton et al. 2017), industrial robots (RethinkRobotics 2022), learning from demonstration (French et al. 2019), and navigation (Macenski et al. 2020). Compared to the aforementioned three robot action sequence methods, we choose BTs because they are particularly well suited to represent atomic and separable behaviors as subtrees with control nodes for replay. For a gentle introduction to behavior trees and in a mobile manipulation task, please see our prior work (Han et al. 2021).

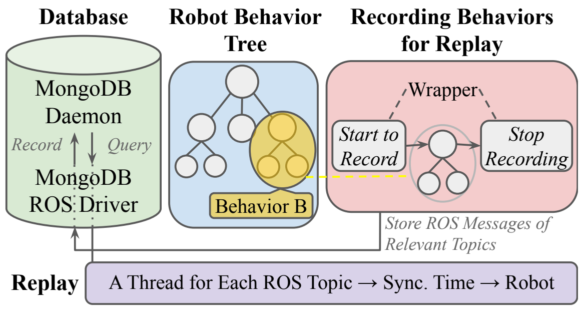

Figure 5: A high-level diagram for our robot behavior replay implementation. Robot behaviors are encoded in behavior trees. After Behavior B is identified for future replay, a wrapper is used to record relevant ROS topic to MongoDB through a MongoDB ROS driver. After querying a specific behavior, multiple threads are created for all relevant ROS topics, replaying them after replay time is synchronized.

3 Behavior Replay Implementation in ROS

3.1 High-Level Workflow

As this is a tool paper, now we go through each step in the replay implementation in ROS. Figure 5 illustrates them.

First, MongoDB and its ROS driver (See http://wiki.ros.org/mongodb_store) need to be installed and running.

Second, the robot behaviors should be encoded in behavior trees. Specifically, we have used the `BehaviorTree.CPP` framework[3]. Our sample code shows how behavior nodes are registered and then specified in an XML behavior tree file. In theory, any task specification should work because only the underlying ROS topics are stored in MongoDB database. However, we recommend specifying robot tasks in a behavior tree to have atomic and separable behaviors. With the behavior tree representation,

[3]https://www.behaviortree.dev/

it removes the burden to separate specific behaviors for replay and it is easy to wrap the control node, representing a behavior, to record concerned ROS topics for replay.

Third, the specific behavior represented by the control parent node should be identified for replay.

Fourth, before and after a control parent node is executed, code for starting and stopping recording, i.e., storing related topics, should be added. This is also called wrapper code and is done through the `MongodbLogger` class: After initializing an `MongodbLogger` instance, specific topics for future replay can be set. A number of them will be discussed in the next subsection.

Finally, to query the recorded topics from MongoDB database, one could simply replay the whole collection (analogous to a table in SQL databases) or query by time or topics. Under the hood, a thread is first created for the replay of every ROS topic after they are retrieved, and after a common clock is established, different ROS messages are replayed at the right timestamp to replicate the movements or visualizations that happened at record time. The code for querying and replaying is in `mongodb_play.py`.

3.2 Concrete Examples

We used the implementation to replay a complex mobile manipulation (kitting) task (Han et al. 2020a), including picking, navigation, and placement. As shown in Figure 2, a Fetch robot successfully replayed its past arm movement and manipulation of a misrecognized object. An AR visualization, the green projection, was also replayed to indicate the grasped object in the past. In Figure 3, the robot replayed its past navigation behavior around an obstacle, with yellow projection for the obstacle and arrows for the detour path. As another manipulation example (Figure 4), Fetch replayed its object placement arm movement with AR projection onto the section of caddy that was misrecognized.

Here, we list the topics we have identified and recorded, with the Fetch robot in mind. Although different robots use different topics, it is particularly easy to find corresponding topics. For body (arm and wheel) movement, the following topics were replayed:

- Gripper: "/gripper_controller/gripper_action/goal"
- Arm and torso: "/arm_with_torso_controller/follow_joint _trajectory/goal"
- Torso: "/torso_controller/follow_joint_trajectory/goal"
- Head: "/head_controller/point_head/goal"
- Head: "/head_controller/follow_joint_trajectory/goal"
- Wheel movement: "/cmd_vel"

Indeed, any topics can be replayed. For example, the green projection in Figure 2 and the yellow projection (Figure 3) to indicate ground obstacle is a PointCloud2 message in the sensor_msgs package[4]. The arrow in Figure 3 is a Path message from the nav_msgs package[5]. The white projection for the caddy section is a Marker message provided by the rviz package[6]. In addition to visuals, non-visual messages can also be replayed. For example, we have been able to replay speech by replaying messages from "/robotsound", used by the sound_play package[7].

4 Evaluation

We have also validated the effectiveness and efficiency of robot behavior replay for past behavior explanation using our implementation in the scenarios described in Figure 2–4. In an experiment (N=665) we reported in detail in another paper (Han and Yanco Under review), we assessed a combination of different replays: Physical replay, AR projection replay, and speech replay. The combination of these three modalities has achieved the best overall effectiveness, helping participants infer where the robot grasped the misrecognized object (Manipulation inference; Figure 2), why the robot made a detour (Navigation inference; Figure 3), and which section of the caddy it wrongly placed an object into (Placement inference; Figure 4). For efficiency, 60% of the participants were able to infer the detour reason with AR projection replay only. Speech replay alone helped participants to quickly make inference on the caddy section. For manipulation inference, although more participants can quickly make the inference with projection or speech reply, more participants also reported they were not able to get the answer from such replays. While physical replay is time-consuming, it ensured inference accuracy. Trust and workload were also rated by participants, but there were no conclusive findings. More details about this experiment can be found in (Han and Yanco Under review).

5 Limitations

Although we were able to implement both manipulation and navigation replays, the navigation replay has approximately a few centimeter error when reaching the destination position in the case of Figure 3. Although this may seem short, it could be problematic where a following dependent behavior, e.g., the placement replay shown in Figure 4, will not have the desired state, e.g., the gripper may not be above the correct caddy section, as seen in the fourth photo in Figure 4. In such cases, we may solve this issue by recording intermediate goals as how body movement was recorded, shown in Section 3.2.

6 Conclusion

In this work, we presented an implementation of robot behavior replay. We justified the choices of database for robotic data storage and robot task representation for specifying and separating robot behaviors. Figure 5 shows the required components and steps to replay robot behaviors. We also discussed the wide applicability of this technique, i.e., capability to replay any kind of ROS messages.

Acknowledgments

This work has been supported in part by the Office of Naval Research (N00014-18-1-2503) and the National Science Foundation (IIS-1909864). We also thank Vittoria Santoro and Jenna Parrillo for code help.

[4]https://wiki.ros.org/sensor_msgs

[5]https://wiki.ros.org/nav_msgs

[6]https://wiki.ros.org/rviz/DisplayTypes/Marker

[7]https://wiki.ros.org/sound_play

References

Amir, O.; Doshi-Velez, F.; and Sarne, D. 2018. Agent strategy summarization. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, 1203–1207.

Beetz, M.; Beßler, D.; Haidu, A.; Pomarlan, M.; Bozcuoğlu, A. K.; and Bartels, G. 2018. Know rob 2.0—a 2nd generation knowledge processing framework for cognition-enabled robotic agents. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 512–519. IEEE.

Beetz, M.; Mösenlechner, L.; and Tenorth, M. 2010. CRAM—A Cognitive Robot Abstract Machine for everyday manipulation in human environments. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1012–1017. IEEE.

Beetz, M.; Tenorth, M.; and Winkler, J. 2015. Open-EASE–a knowledge processing service for robots and robotics/AI researchers. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 1983–1990. IEEE.

Bohren, J.; and Cousins, S. 2010. The smach high-level executive. *IEEE Robotics & Automation Magazine*, 17(4): 18–20.

Brunner, S. G.; Steinmetz, F.; Belder, R.; and Dömel, A. 2016. RAFCON: A graphical tool for engineering complex, robotic tasks. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3283–3290. IEEE.

Colledanchise, M.; and Ögren, P. 2018. *Behavior trees in robotics and AI: An introduction*. CRC Press.

French, K.; Wu, S.; Pan, T.; Zhou, Z.; and Jenkins, O. C. 2019. Learning behavior trees from demonstration. In *2019 International Conference on Robotics and Automation (ICRA)*, 7791–7797. IEEE.

Han, Z.; Allspaw, J.; LeMasurier, G.; Parrillo, J.; Giger, D.; Ahmadzadeh, S. R.; and Yanco, H. A. 2020a. Towards mobile multi-task manipulation in a confined and integrated environment with irregular objects. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 11025–11031. IEEE.

Han, Z.; Giger, D.; Allspaw, J.; Lee, M. S.; Admoni, H.; and Yanco, H. A. 2021. Building the foundation of robot explanation generation using behavior trees. *ACM Transactions on Human-Robot Interaction (THRI)*, 10(3): 1–31.

Han, Z.; Parrillo, J.; Wilkinson, A.; Yanco, H. A.; and Williams, T. 2022. Projecting Robot Navigation Paths: Hardware and Software for Projected AR. In *Proceedings of the 2022 ACM/IEEE International Conference on Human-Robot Interaction*, 623–628.

Han, Z.; Phillips, E.; and Yanco, H. A. 2021. The need for verbal robot explanations and how people would like a robot to explain itself. *ACM Transactions on Human-Robot Interaction (THRI)*, 10(4): 1–42.

Han, Z.; Rygina, P.; and Williams, T. 2022. Evaluating Referring Form Selection Models in Partially-Known Environments. In *Proceedings of the 15th International Natural Language Generation Conference*.

Han, Z.; Wilkinson, A.; Parrillo, J.; Allspaw, J.; and Yanco, H. A. 2020b. Projection mapping implementation: Enabling direct externalization of perception results and action intent to improve robot explainability. In *Proceedings of the AI-HRI Symposium at AAAI-FSS 2020*.

Han, Z.; and Yanco, H. A. Under review. Communicating Missing Causal Information to Explain a Robot's Past Behavior. Under review.

Hayes, B.; and Shah, J. A. 2017. Improving robot controller transparency through autonomous policy explanation. In *2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI*, 303–312. IEEE.

Ikeda, B.; and Szafir, D. 2022. Advancing the Design of Visual Debugging Tools for Roboticists. In *Proceedings of the 2022 ACM/IEEE International Conference on Human-Robot Interaction*, 195–204.

Lim, C.-U.; Baumgarten, R.; and Colton, S. 2010. Evolving behaviour trees for the commercial game DEFCON. In *European conference on the applications of evolutionary computation*, 100–110. Springer.

Macenski, S.; Martín, F.; White, R.; and Clavero, J. G. 2020. The marathon 2: A navigation system. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2718–2725. IEEE.

Nakawala, H.; Goncalves, P. J.; Fiorini, P.; Ferringo, G.; and De Momi, E. 2018. Approaches for action sequence representation in robotics: A review. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 5666–5671. IEEE.

Niemueller, T.; Lakemeyer, G.; and Srinivasa, S. S. 2012. A generic robot database and its application in fault analysis and performance evaluation. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 364–369. IEEE.

OpenRobotics. 2022. Official ROS robot showcase. https://robots.ros.org. Accessed: 2022-07-13.

Palamara, P.; Ziparo, V.; Iocchi, L.; Nardi, D.; Lima, P.; and Costelha, H. 2008. A robotic soccer passing task using petri net plans (demo paper). In *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, 1711–1712.

Paxton, C.; Hundt, A.; Jonathan, F.; Guerin, K.; and Hager, G. D. 2017. CoSTAR: Instructing collaborative robots with behavior trees and vision. In *2017 IEEE international conference on robotics and automation (ICRA)*, 564–571. IEEE.

Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; Ng, A. Y.; et al. 2009. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, 5.

RethinkRobotics. 2022. Intera Software Platform for Industrial Automation. https://www.rethinkrobotics.com/intera. Accessed: 2022-07-26.

Rosen, E.; Whitney, D.; Phillips, E.; Chien, G.; Tompkin, J.; Konidaris, G.; and Tellex, S. 2019. Communicating and controlling robot arm motion intent through mixed-reality head-mounted displays. *The International Journal of Robotics Research*, 38(12-13): 1513–1526.

Sagredo-Olivenza, I.; Gómez-Martín, P. P.; Gómez-Martín, M. A.; and González-Calero, P. A. 2017. Trained behavior trees: Programming by demonstration to support ai game designers. *IEEE Transactions on Games*, 11(1): 5–14.

Scheutz, M.; Williams, T.; Krause, E.; Oosterveld, B.; Sarathy, V.; and Frasca, T. 2019. An overview of the distributed integrated cognition affect and reflection diarc architecture. *Cognitive architectures*, 165–193.

Schneider, F. B. 1990. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4): 299–319.

Stange, S.; Hassan, T.; Schröder, F.; Konkol, J.; and Kopp, S. 2022. Self-Explaining Social Robots: An Explainable Behavior Generation Architecture for Human-Robot Interaction. *Frontiers in Artificial Intelligence*, 87.

Stange, S.; and Kopp, S. 2020. Effects of a social robot's self-explanations on how humans understand and evaluate its behavior. In *2020 15th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 619–627. IEEE.

Tenorth, M.; and Beetz, M. 2009. KnowRob—knowledge processing for autonomous personal robots. In *2009 IEEE/RSJ international conference on intelligent robots and systems*, 4261–4266. IEEE.

Walker, M.; Phung, T.; Chakraborti, T.; Williams, T.; and Szafir, D. 2022. Virtual, augmented, and mixed reality for human-robot interaction: A survey and virtual design element taxonomy. *arXiv preprint arXiv:2202.11249*.

Wise, M.; Ferguson, M.; King, D.; Diehr, E.; and Dymesich, D. 2016. Fetch and freight: Standard platforms for service robot applications. In *Workshop on autonomous mobile service robots*.

Zhu, L.; and Williams, T. 2020. Effects of proactive explanations by robots on human-robot trust. In *International Conference on Social Robotics*, 85–95. Springer.

Ziparo, V. A.; Iocchi, L.; Nardi, D.; Palamara, P. F.; and Costelha, H. 2008. Petri net plans: a formal model for representation and execution of multi-robot plans. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, 79–86.