Human-Robot Collaboration and Dialogue for Fault Recovery on Hierarchical Tasks

Janelle Blankenburg[1], Mariya Zagainova[1], S. Michael Simmons[2], Gabrielle Talavera[1], Monica Nicolescu[1], and David Feil-Seifer[1]

[1] Department of Computer Science & Engineering University of Nevada, Reno, Reno, NV, 89557 jjblankenburg@nevada.unr.edu

[2] Department of Computer Science, Brigham Young University, Provo, Utah, 84602

Abstract. Robotic systems typically follow a rigid approach to task execution, in which they perform the necessary steps in a specific order, but fail when having to cope with issues that arise during execution. We propose an approach that handles such cases through dialogue and human-robot collaboration. The proposed approach contributes a hierarchical control architecture that *1) autonomously detects and is cognizant of task execution failures*, *2) initiates a dialogue with a human helper to obtain assistance*, and *3) enables collaborative human-robot task execution through extended dialogue* in order to *4) ensure robust execution of hierarchical tasks with complex constraints, such as sequential, non-ordering, and multiple paths of execution*. The architecture ensures that the constraints are adhered to throughout the entire task execution, including during failures. The recovery of the architecture from issues during execution is validated by a human-robot team on a building task.

Keywords: human-robot collaboration · human-robot dialogue · dialogue-based fault recovery · hierarchical planning.

1 Introduction

Fault recovery in autonomous robot systems is an essential component for ensuring that any unexpected circumstances can be handled without the complete failure of a task. The goal of this work is to develop a control architecture for hierarchical tasks which is able to recover from faults during execution through dialogue and human-robot collaboration. The proposed architecture is cognizant of failures and can initiate a dialogue to resolve an issue. Extended dialogue between the robot and human, rather than a single request for help, allows for multiple ways of resolving a fault. Failures are autonomously detected through a combination of views from multiple cameras. The architecture ensures that the task constraints are held throughout the entire task execution, even during failures. This allows for a robust execution of a hierarchical task with multiple types of constraints such as sequential, non-ordering, and multiple paths of execution. We provide an extension to our previously developed control architecture [8] to allow for fault recovery, to allow for a smooth interaction between a human and robot collaborating on a complex, hierarchical task.

2 Related Work

Joint assembly tasks employ several elements in order for the system to acquire/learn a model of the task, to monitor its progress, and to repair the system when things fail. Task construction is an essential part of this process, demonstrated through human-robot collaboration [5] or dialogue-based systems [17]. Human demonstrations can be used to learn a hierarchical plan [10]. Task description can also be manually specified by using a graphical user interface [15]. However, failure resolution was not done when collisions occurred in these works, the robots would generally defer to what their partner wanted to do.

Task verification is critical in joint assembly tasks for robotic systems. It has been implemented in single [14] and multiple [2] robot systems that use computer vision for task verification. In the 2016 Amazon Picking Challenge, a vacuum sensor was used to receive boolean feedback on the grasp of an object [12]. The proposed approach uses multiple on-board sensors to ensure proper completion of each task step and that constraints are upheld during resolution.

When a robot fails a task it is important that the team can resolve the problem autonomously [18]. Fong et. al. found that dialogue makes human users more aware of the problems robots face [7]. When plans fail, it can help when a system can explain why it made certain decisions [11]. Unlike these works, the proposed work seeks to resolve conflicts in hierarchical tasks with complex constraints by utilizing an extended dialogue between a human and robot.

Several studies have had robots initiate communication with humans when a problem arose [7, 6]. Fong et. al. had a robot explore a room via teleoperation and ask a remote human about how to proceed when confronted with uncertainty [7]. Extensions of this work had a team of robots conduct a surveillance task, illuminating that dialogue improved the human's ability to deal with context switching [6]. Both [7, 6] focused on collaborative teleoperation based tasks. These studies primarily focused on how humans interact with robots asking questions. However, humans primarily offered additional information to the robot but were not capable of helping the robot complete the task, which our work allows.

Robots have also asked humans for assistance in building a piece of IKEA furniture [14]. The robots identified the causes of their problems and initiated a dialogue to solve it. This work is most similar to ours but has several key differences: their architecture was bound by a rigid instruction set; the system required an external Vicon system to perform the fault-detection whereas we utilize on-board sensors; and it focuses on the complexity of the robots' requests for help whereas we focus on an extended dialogue between human and robot.

3 Control Architecture with Fault Recovery

3.1 Distributed Control Architecture

This work extends the control architecture developed in [8] to incorporate a dialogue-based management system of task faults capable of autonomously detecting and resolving issues through human-robot collaboration and dialogue.

The architecture uses a behavior-based paradigm [1], which allows communication and connectivity between sub-tasks. It encodes tasks into a hierarchical structure capable of incorporating multiple types of constraints, namely sequential, non-ordering, and multiple paths of execution (as shown in Figure 1). The structures are composed of two types of nodes. **Goal Nodes** provide the base goal control behaviors of the hierarchical task structure used to encode task constraints: sequential (**THEN**), non-ordering (**AND**), and alternative paths of execution (**OR**). **Behavior Nodes** are leaf nodes that encode the robot's physical behaviors.

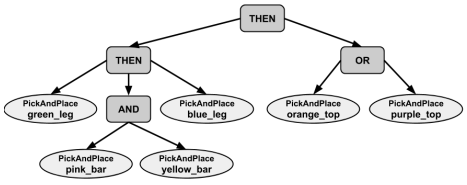

Fig. 1. The task tree for the EKET building task. Internal nodes are goal nodes and leaves are behavior nodes.

Each node maintains a state consisting of several components (*activation level, activation potential, active,* and *done*), which is used for communication and connectivity between nodes. Each node's state is continuously updated to perform top-down and bottom-up activation spreading, which guarantees proper execution of the task with respect to its constraints. To execute a task, *activation spreading messages* are sent from the root node towards its children, thereby performing a top-down spreading of the *activation level* throughout the tree. Simultaneously, each nodes sends *status messages* (encoding a node's current state) to its parent node, thereby performing a bottom-up spreading of the *activation potential*. Each node's state is maintained via an update loop that performs a series of checks at each cycle. This loop uses the *activation potential* information to activate the node that has the highest potential [8].

3.2 Interfacing with the Control Architecture

To allow the architecture to handle interruptions that come from the fault detection system, the update loop of the nodes from [8] was modified by adding a checking mechanism that allows the loop to continue as normal unless a failure is detected. In case of a detected fault, a Robot Operating System (ROS) [16] message is published to the corresponding node's *issue* topic. Once the node receives such a message, the node's *issue* callback function is triggered (Figure 2). In this function, a ROS message is published on the *dialogue* topic to initialize the dialogue that corresponds to the specific failure that was detected. This initiates the dialogue between the robot and human and allows the human to provide assistance (Section 3.3). After the dialogue is initialized, a while loop stops the current behavior in the architecture, as well as the physical motion of the robot, from finishing until a resolution has been reached through the dialogue between the robot and human. Since the node that the robot is working on is active at the time of the detected failure, no other nodes can be activated until that node is done or reset, allowing the entire architecture, and therefore task progress, to be paused from within a single node. This pause ensures that no task constraints are broken during the handling of the fault.

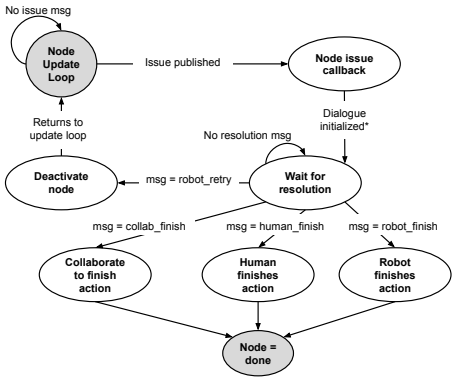

Fig. 2. State machine diagram of architecture flow upon issue detection.

Once a resolution message is received from the dialogue system, changes are made to the node's state based on the type of resolution. If the resolution involves either the human, robot, or both, to complete the task then the node's state is set to *done* and its *activation level* is set to zero. In the case that the resolution is *human_finish*, the human will perform the required work to complete the task. If the resolution is *robot_finish*, then the robot will continue on with the remaining work required to finish the task, after being briefly assisted by the human (i.e. the human hands the robot an object that is out of its reach). Once the human completes the action, the robot is able to finish the task without further help. This assistance varies based on the task at hand and the issue found. If the resolution is *collab_finish*, then the human must work together with the robot simultaneously to complete the task (i.e., the human must hold and align an object as the robot connects another object). This resolution requires both agents to work together at the same time to fully complete the task.

Lastly, if the robot is required to retry the execution (*robot_retry*), the node gets deactivated. This deactivation sets the node's state back to what it was before the node was activated, thereby ensuring that the task constraints encoded by the task tree are still upheld after the conflict is resolved. The node's state is set to *not done* and its *activation level* is reset to its original level upon activation. If a node is deactivated, it can be chosen for activation at a later time and the robot can attempt the execution of that behavior again.

Pause and deactivate ensures that our architecture is able to maintain the task constraints during the entire task execution. The various resolution messages allow the architecture to utilize multiple ways to resolve a conflict, thus being able to handle different levels of conflict. Some resolutions only require a temporary pause of the architecture until the work is completed (minor fault), while others require the robot to retry (major fault, as it requires both pausing and deactivating the node, which in turn resets part of the task tree).

3.3 Dialogue Module

When a ROS message is published to the *dialogue* topic, the dialogue is initialized, as shown in Figure 2. This initiates a communication between the robot and human. The high-level flow-chart for the initiated dialogue (Figure 3) illustrates the major interactions that occur between the human and robot that encompass the extended dialogue. There are two main components to this interaction that are specific to the failure that was detected: **Detected issue** (the name of the issue detected) and **Action** (the action that needs to be performed).

Additionally, there are two internal checks which affect the outcome of the dialogue: *1) Human collaboration required?* and *2) Should robot complete task now?* The first checks if collaboration is required to complete the task, i.e. the human and robot must work together simultaneously to finish the task. The second checks whether the robot should complete the task at the current time.

If the issue does not require human collaboration the robot will ask if it should complete the task. If the human replies with *yes*, the robot will provide the human with instructions on how to reset the objects so the robot can complete the task on its own. Then, depending on the second check, the robot will either finish the task at the current time or inform the human that it will retry the task again later and the corresponding resolution message is published. If the human replies with a *no*, then the robot will ask if the human will complete the task. If the human again responds with *no*, the resolution message is published to enable the robot to retry and the human is notified. If the human responds with *yes*, the robot thanks the human and the resolution message for the human completing the task is published.

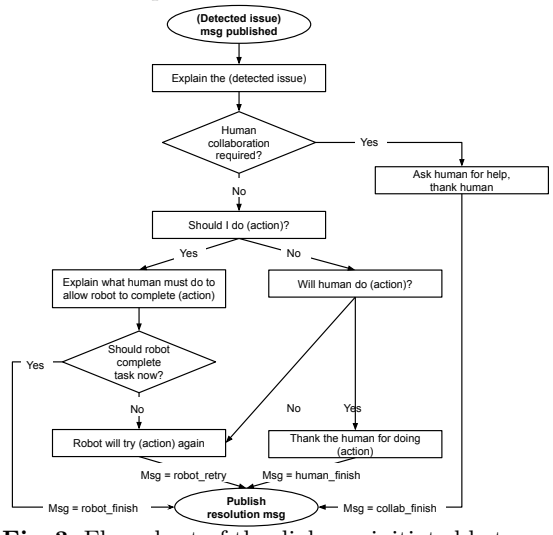

Fig. 3. Flow-chart of the dialogue initiated between robot and human when an issue is detected.

The robot utilizes an on-board speaker and the sound_play [9] software to communicate with the human. PocketSphinx [13] is used to recognize the human's *yes* or *no* responses. Once a response is recognized, the dialogue flow between robot and human continues accordingly.

This approach provides a simple way for new issues to be added into the system. Although specific details (such as the exact dialogue exchanges) will vary based on the issue, this flow outlines all of the necessary interactions that would occur between the human and the robot for any simple issue that could be added, emphasizing the generality of the proposed dialogue-based management system for fault recovery. The faults currently detected by our system for the assembly scenario (Section 4) are summarized below:

Positioning: The *positioning* issue message is raised if a robot needs assistance with precisely positioning an object as it is placed. The robot asks the human for help placing the object and thanks the human. The motion of the robot is then slowed down and the human can assist with the positioning of the object.

Missed: The *missed* issue message is raised when the robot misses an object during pick-up. The robot explains it missed the object and asks to try again. If the human agrees, the robot will ask the human to move the object to its

original position on the table, and says it will try picking it again later. If the human disagrees, the robot asks if the human will place the object. If the human says *yes*, they will place it to the final location. Otherwise, the robot says it will try again later.

Dropped: The *dropped* issue message is raised when the robot drops an object after picking it up. The dialogue flow is the same as in the *missed* case, indicating drop instead of missed.

Unreachable: The *unreachable* issue message is raised if a robot is unable to reach an object. The robot will ask if the human can hand the object to the robot. If the human complies, the robot will grab the object and finish completing the task. If the human refuses, the robot will ask if the human will place the object. If the human says *yes*, the human will place the object to its final location. Otherwise, the robot says it will try again later.

3.4 Fault Detection System

In order for the architecture to detect issues, a fault-monitoring system has been added to each node in the task tree for the base control architecture [8]. Once a node gets activated, the system begins monitoring for faults during the execution of the node's work. In our work, this occurs during the execution of the *PickAndPlace*, which performs the following steps, in order: *1) move above the pick position, 2) move to the pick position, 3) close the gripper, 4) move back above the pick position, 5) move above the place position, 6) move to the place position, 7) open the gripper, and 8) move back above the place position.* To ensure the arm is not colliding with objects as it moves between pick and place locations, the arm is moved above (positive z-offset) the pick and place position after opening/closing the gripper.

During this sequence of steps, the monitoring system checks for various fault cases using a combination of the robot's on-board sensors. In the *dropped* and *missed* cases, the robot's motion along this path is interrupted and the arm is moved to a neutral location to wait until a resolution is reached.

In order to extend this monitoring system to new issues, we only need to define the start/stop step along the *PickAndPlace* sequence, which will begin/end the monitoring. However, the sensors used to check if the issue has occurred will vary based on the specifics of the issue. Additionally, these sensors might require specific settings, such as locations of objects in a particular camera.

For the *unreachable* fault, the starting and stopping steps are the first step in the sequence (i.e. move to *above pick location*). Before this motion occurs, the system checks if the object is within the robot's graspable range using a simple distance check from the robot to the object's initial location as detected from the Kinect on the PR2 robot's head.

A simple color blob detector implemented with OpenCV [4] is used to find objects in an image. HSV-segmentation of pre-trained color histograms, combined with morphological open/close operations, isolates large regions of color in the image to represent each object. The monitoring system uses these trained

colors to identify whether or not the object is in the gripper by using the RGB image from the PR2's right forearm camera. The fault detection system searches the image for the color blob of the object's respective color. If the center of the blob is not within a predefined range of values in the image, it registers either the *missed* or *dropped* fault, depending on which part of the motion was being executed when the fault was detected.

The fault-monitoring system checks for a missed fault between the two *above pick location* steps (steps 1-4). If the color blob detection does not detect the object in the correct location in the forearm camera, it registers a *missed* fault. The system then checks for a *dropped* issue between the second *above pick location* step and the first *above place location* step (steps 4-5). At any point between these steps in the execution, if the color-blob detection does not detect the object in the correct location in the forearm camera, it registers a *dropped* issue.

The *positioning* issue is only checked at the first *above place location* (step 5). The system checks whether the carried object is in the list of predefined objects which require assistance. If the object requires help, the system registers a *positioning* issue: the robot slows down the movement from the *above place location* to the *place location* (steps 5-6), which allows the human ample time to align the necessary object as the robot connects the new object. The robot then moves to the *above place location* (step 8) and returns to regular speed.

The *dropped*, *missed*, and *unreachable* failures represent major issues as they require a complete interrupt of the robot motion and the architecture. Without human assistance, the robot would be unable to complete the task. On the other hand, the *positioning* issue is a lesser fault. The robot requires assistance to align the objects perfectly, but neither the robot motion nor the architecture need to be interrupted in this case. Thus, our dialogue-based management system for fault recovery allows the architecture to recover from faults of varying degrees.

4 Experimental Validation

The proposed architecture has been validated with a robot-human team in a scenario specifically designed to illustrate the key proposed contribution: *a control architecture that 1) autonomously detects and is cognizant of task execution failures, 2) initiates a dialogue with a human helper to obtain assistance, and 3) enables collaborative human-robot task execution through extended dialogue in order to 4) ensure robust execution of hierarchical tasks with complex constraints, such as sequential, non-ordering, and multiple paths of execution.* Most of the proposed additions to the architecture are outlined by general methods, so a concrete scenario with example cases for each addition is utilized to validate the combined functionality of the architecture.

The task used to validate the architecture involves building a modified EKET base from IKEA, with all parts painted in different colors for disambiguation. The task structure is shown in Figure 1, representing inherent constraints for attaching the parts. The task is performed as follows: *1) place the green leg in front of the robot, 2) attach the pink and yellow bars in either order, 3) attach*

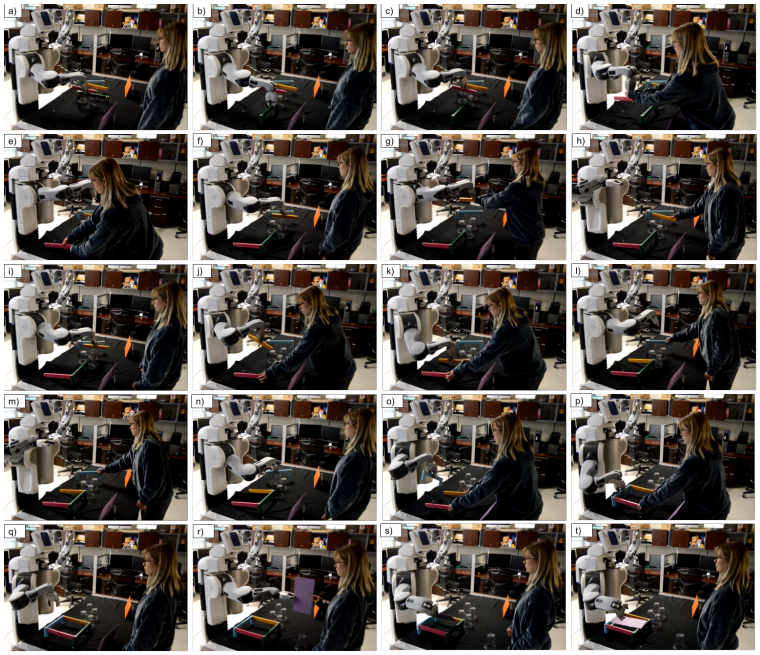

Fig. 4. Execution of the task with issues and assistance provided by the human.

the blue leg, and 4) place either the purple or orange top on top. Step 2 reflects the non-ordering constraints of the task since the order of placing the pink and yellow bars does not matter. Step 4 reflects the alternate paths of execution in the task since either one of the tops can be placed.

The parts are placed on a table in front of the PR2 robot. The *PickAndPlace* behavior nodes get their respective pick locations from an initial detection of objects using the Kinect by applying the same type of color blob detection utilized on the forearm cameras as discussed in Section 3.4. The respective place locations of the *PickAndPlace* behavior nodes are set as pre-specified locations in order to allow the objects to be attached together. End-effector trajectories to the pick/place locations are generated using the MoveIt library [19].

During the experiment, the robot determines the order of actions to take based on the activation spreading mechanism in our previously developed architecture [8]. This mechanism creates a dynamic ordering in which to complete the sub-tasks based on the environmental conditions and guarantees this ordering adheres to the constraints of the task. To validate the fault recovery of the proposed architecture, a human simulates each of the possible fault cases by interrupting robot during this task.

4.1 Task Execution

The execution of the experiment is shown in Figure 4. To illustrate the recovery and collaboration capabilities of the proposed architecture, the human interfered by stealing several objects during the execution of the task, causing the robot to detect the various types of faults handled by the architecture.

The task begins with the robot performing the *PickAndPlace(green_leg)* behavior in pictures (a) and (b) in Figure 4. This sub-task completed correctly without any faults, illustrating that the control architecture is able to perform as usual under normal conditions. Next, the control architecture specifies that the robot must place the pink and yellow bars in any order. The pink bar is closer to the robot's gripper so the architecture tells the robot to grab it first using the activation mechanism in [8]. Thus, in (c) the robot begins the *PickAndPlace(pink_bar)* behavior. However, the human steals the pink bar right before the robot places the object (d). After the human steals the object the fault detection system detects a fault as described in Section 3.4. At this point, the vision system running on the forearm camera no longer detects the object in the robot's gripper. The fault detection system then uses the point during execution at which the object was lost to determine which fault occurred. Since the robot was en route to the place location, the fault system triggers a *dropped* issue to be published, which then causes the robot to begin a dialogue with the human as described in Figure 3. The human follows the branch of the dialogue flow that leads to the human placing the pink bar as shown in (e).

After the fault is resolved and the object is placed, the robot continues the task by beginning the *PickAndPlace(yellow_bar)* behavior in (f). After the robot picks up the yellow bar, the human immediately steals it as shown in (g). Again, the fault detection system loses track of the object after the robot had successfully grasped the object so it triggers the *dropped* issue to be published, which causes the robot to begin a dialogue with the human. This illustrates that the dropped issue can get triggered in multiple parts of a sub-task's execution. This time however, the human follows the dialogue path that leads to the robot having to try again. In (g) the human places the yellow bar back on the table, as prompted by the dialogue, and the architecture resets the corresponding part of the task. Due to the task constraints, both the pink bar and yellow bar must be placed before moving on to the next part of the task, so the robot attempts the *PickAndPlace(yellow_bar)* behavior again in (i). During the placement of the yellow bar (j-k), the fault detection system determines that the object is one which was specified to require human assistance for placement. It then raises the *positioning* issue and asks the human for help in placing the object as described in the rightmost branch in Figure 3. Once the object has been placed with the human's assistance, the robot continues to the next part of the task.

In (l) the human steals the blue leg right before the robot picks it up during the *PickAndPlace(blue_leg)* behavior. At this point during the execution, the fault detection system expects the object to be detected in the gripper. However, the object is not detected which means that the robot did not successfully grasp the object. Thus, the fault detection system triggers the *missed* issue to be published, which causes the robot to begin a dialogue with the human. The human follows the dialogue flow which leads to the robot trying again. In (m) the human places the blue leg back on the table, as prompted by the dialogue, and the architecture resets the corresponding part of the task. Due to the sequential task constraints, the blue leg must be placed before the next part of the task

can happen, so the robot attempts the *PickAndPlace(blue_leg)* behavior again as shown in (n). In (o-p) the fault detection system once again triggers the *positioning* issue and asks the human for help since this object was also determined to be one which was specified to require assistance.

Based on the task constraints, the robot can choose to place either the orange or the purple top. Since the purple top is closer to the robot's gripper, the purple top is placed. In (q) the robot begins the *PickAndPlace(purple_top)* behavior. The fault detection system discovers that the purple top is out of the robot's reachable space since the distance check described in Section 3.4 fails. It then raises an *unreachable* issue. This triggers the dialogue and the human follows the flow that results in a hand-off between the human and robot (Figure 3, left-most path) in (r). The robot finishes placing the purple top in (s). Finally, (t) shows the completed task with the fully assembled EKET base.

4.2 Discussion of Experiment

The execution of the experiment (Section 4.1) validates that our proposed architecture effectively utilizes the dialogue-based management system for fault recovery of hierarchical tasks. The detection of faults is done entirely with sensors on-board the robot through a combination of views from multiple cameras. The robot was able to complete the *PickAndPlace(green_leg)* behavior without fault. This shows the architecture can complete tasks without assistance when no faults occur (Section 3.1). The execution of the *PickAndPlace(pink_bar)* and *PickAndPlace(yellow_bar)* behaviors both illustrate examples of a *dropped* failure. These objects were detected as dropped along the behaviors' execution which demonstrates the system is able to detect and resolve faults at various points, as long as they occur between the start and stop steps during which the issue is monitored. Furthermore, they illustrate that failures can be resolved through either human assistance or another attempt from the robot. The execution of the *PickAndPlace(blue_leg)* behavior demonstrated that the system is able to both detect and handle a *missed* failure. The steps for placing the blue leg and the yellow bar demonstrated that the dialogue-based management system is able to handle various major faults which require it to reset parts of the control architecture. This shows that the task constraints are upheld after the architecture is reset, since the architecture completed the placements of these objects before moving on (as defined by the task tree constraints). Lastly, the *PickAndPlace(purple_top)* behavior illustrates that the system is able to manage the *unreachable* fault and negotiate a hand-off between the human and robot.

The various behaviors demonstrate that the dialogue-based system is able to assist with the task execution in multiple resolution cases by utilizing the extended dialogue between the human and the robot. By showing each of these failure cases in a single scenario, it shows that the proposed system is able to robustly handle faults that occur during the execution of complex, hierarchical tasks. The robot is able to autonomously detect faults occurring from execution failures, begin a dialogue with the human to resolve these faults, and resume the normal task execution upon fault recovery without breaking any constraints.

5 Conclusion and Future Work

This paper presents an extension to our previous control architecture [8], which incorporates a dialogue-based management system for fault recovery of hierarchical tasks through the use of human-robot collaboration. The contribution of this approach is a control architecture that *1) autonomously detects and is cognizant of task execution failures*, *2) initiates a dialogue with a human helper to obtain assistance*, and *3) enables collaborative human-robot task execution through extended dialogue* in order to *4) ensure robust execution of hierarchical tasks with complex constraints*. This method is able to autonomously detect faults occurring from execution failures, begin a dialogue with the human to resolve these faults, and resume normal task execution upon recovery. Furthermore, the architecture is able to uphold all task constraints while faults are being handled. Extended dialogue with the human allows for multiple avenues to resolve a detected fault, instead of a single request for help. Faults are detected autonomously with onboard sensors, through the robot's multiple cameras. The proposed approach is validated on a building task with a human-robot team. The system can robustly detect and recover from faults that occur during the execution of a complex, hierarchical task through the use of human-robot collaboration and dialogue.

An immediate extension of this work is to incorporate the proposed fault handling system into our multi-robot control architecture [3]. This extension will allow for humans to collaborate with multi-robot teams working on a joint task. For this extension, each robot's control architecture would be modified to incorporate our dialogue-based fault detection system as described in Section 3.2 for a single robot. This would allow each of the robots to initiate a dialogue with the human when a fault is detected. Additionally, the dialogue-based system could be modified to give the robots the option to ask each other for assistance to recover from faults as well. Furthermore, other fault types which may arise from translating to multi-robot teams can be easily implemented in this system, as they would follow the same general framework outlined in Section 3.

Acknowledgment

This work was supported by the National Science Foundation (IIS-1757929) and by the Office of Naval Research (ONR) award #N00014-16-1-2312.

References

1. Arkin, R.C.: An Behavior-based Robotics. MIT Press, Cambridge, MA, USA, 1st edn. (1998)
2. Beetz, M., Klank, U., Kresse, I., Maldonado, A., Mösenlechner, L., Pangercic, D., Rühr, T., Tenorth, M.: Robotic roommates making pancakes. In: 2011 11th IEEE-RAS International Conference on Humanoid Robots. pp. 529–536 (Oct 2011)
3. Blankenburg, J., Banisetty, S.B., Alinodehi, S.P.H., Fraser, L., Feil-Seifer, D., Nicolescu, M., Nicolescu, M.: A distributed control architecture for collaborative multi-robot task allocation. In: 2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids). pp. 585–592 (Nov 2017)

4. Bradski, G., Kaehler, A.: Learning OpenCV: Computer vision with the OpenCV library. " O'Reilly Media, Inc." (2008)
5. Breazeal, C., Hoffman, G., Lockerd, A.: Teaching and working with robots as a collaboration. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 3. pp. 1030–1037. IEEE Computer Society (2004)
6. Fong, T., Thorpe, C., Baur, C.: Multi-robot remote driving with collaborative control. IEEE Transactions on Industrial Electronics **50**(4), 699–704 (Aug 2003)
7. Fong, T., Thorpe, C., Baur, C.: Robot, asker of questions. Robotics and Autonomous Systems **42**(3), 235 – 243 (2003). https://doi.org/10.1016/S0921-8890(02)00378-0, socially Interactive Robots
8. Fraser, L., Rekabdar, B., Nicolescu, M., Nicolescu, M., Feil-Seifer, D., Bebis, G.: A compact task representation for hierarchical robot control. In: 2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids). pp. 697–704 (Nov 2016). https://doi.org/10.1109/HUMANOIDS.2016.7803350
9. Gassend, B.: sound_play ros package, http://wiki.ros.org/sound_play
10. Hayes, B., Scassellati, B.: Autonomously constructing hierarchical task networks for planning and human-robot collaboration. In: 2016 IEEE International Conference on Robotics and Automation (ICRA). pp. 5469–5476. IEEE (2016)
11. Hayes, B., Shah, J.A.: Improving robot controller transparency through autonomous policy explanation. In: 2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI. pp. 303–312. IEEE (2017)
12. Hernandez, C., Bharatheesha, M., Ko, W., Gaiser, H., Tan, J., van Deurzen, K., de Vries, M., Van Mil, B., van Egmond, J., Burger, R., Morariu, M., Ju, J., Gerrmann, X., Ensing, R., Van Frankenhuyzen, J., Wisse, M.: Team delft's robot winner of the amazon picking challenge 2016. In: Behnke, S., Sheh, R., Sarıel, S., Lee, D.D. (eds.) RoboCup 2016: Robot World Cup XX. pp. 613–624. Springer International Publishing, Cham (2017)
13. Huggins-Daines, D., Kumar, M., Chan, A., Black, A.W., Ravishankar, M., Rudnicky, A.I.: Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices. In: 2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings. vol. 1, pp. I–I. IEEE (2006)
14. Knepper, R.A., Tellex, S., Li, A., Roy, N., Rus, D.: Recovering from failure by asking for help. Autonomous Robots **39**(3), 347–362 (Oct 2015)
15. Mohseni-Kabir, A., Rich, C., Chernova, S., Sidner, C.L., Miller, D.: Interactive hierarchical task learning from a single demonstration. In: Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction. pp. 205–212. ACM (2015)
16. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: ROS: an open-source Robot Operating System. In: ICRA workshop on open source software. vol. 3 (2009)
17. Rybski, P.E., Yoon, K., Stolarz, J., Veloso, M.M.: Interactive robot task training through dialog and demonstration. In: Proceedings of the ACM/IEEE international conference on Human-robot interaction. pp. 49–56. ACM (2007)
18. Schillinger, P., Kohlbrecher, S., von Stryk, O.: Human-robot collaborative high-level control with application to rescue robotics. In: 2016 IEEE International Conference on Robotics and Automation (ICRA). pp. 2796–2802. IEEE (2016)
19. Sucan, I.A., Chitta, S.: Moveit! Online at http://moveit. ros. org (2013)