2024 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)
July 15-19, 2024. Boston, MA, USA

Enhancing Non-Expert User Interaction with Robots: An Advanced Interface for Error Handling

Johann Arthur Darboven[1], Ryota Takamido[2] and Jun Ota[2]

***Abstract*— A key aspect of robotic systems include the ability to recover from an error and re-planning motion accordingly. The widespread acceptance of robotics in various sectors of society is inhibited by the degree of expertise required to program even the simplest error recovery strategies. Previous studies have addressed the complexity of robotics programming through user-interfaces, though they were not intended for novices or did not address error handling. This study discusses a framework developed that acts as an Interactive Robot Monitor and Control System (IRMCS), that allows non-expert users to interface, understand and recover from errors in a robotic process. The framework takes the form of a system that is geared towards informing a user of a robotic process through the means of an activity diagram that allows for visual and intuitive human-robot interaction to support incremental learning and error handling of simple pick and place tasks. To evaluate the effectiveness of this approach, we conducted a control experiment with four novice users. The results revealed that by using the developed system, novices were able to recover from errors and were unsuccessful in the control condition. Furthermore, their subjective evaluation showed that these non-expert users are highly receptive to understanding and successfully implementing error recovery strategies in robotics.**

I. BACKGROUND

Acceptance of robotics in society relies on a multitude of factors such as error handling, usability, efficiency, technical advantage and many more. This acceptance is further enhanced when robots can seamlessly perform tasks, provide meaningful assistance, and interact in a manner that feels natural and comfortable to users. The objective of robot programming is not only to generate predetermined behavior, but also to anticipate and recover from errors that occur during such behavior. Ultimately, robot programming is therefore an activity that is limited to domain experts rather than novices. Therefore we have identified a need in defining a system that enables users to understand robotic processes and interact simply. Related works have elaborately tackled both error handling as well as improving human-robot interactions (HRI).

A. Error Handling

Error handling has a tendency of taking up a majority of robotics design, especially in an HRI context as user input or actions are often unpredictable. In [1] it is demonstrated that having a set solution pool available upon anomaly detection, a scheduler can allow for transitions between error recoveries in real-time (during execution). The effectiveness of such an approach alleviates a designer's workload significantly as different modalities can be chosen to deal with anomalies without the necessity of diagnostics at an initial stage. Research that clearly divides robotic actions into stratified robotic movements called "primitives" [2] has shown promising results in breaking down error handling such that error recovery strategies can be tailored to each primitive. Primitives include movements such as "approach", "lift", "grasp" or "home" (referring to returning to a calibrated position in the robotic workplace). These primitives were also visualized in a flowchart to improve readability and despite the goal of finding improved error recovery strategies, [2] implicitly paves the way to explainable robotics by breaking down complicated background processes and applying tailored error recovery solutions to high-level motions.

A holistic approach is presented by [3], as multiple error states are treated with the same error recovery strategy as these may overlap even when the source of error differs. The benefit of reducing error recovery strategies lies in simplicity of practical application such that new error modes are not necessarily causing the system to derail. In [4], technicians interact with high-level programming interfaces to attempt simple error recovery strategies though some expertise with the robots is required. A culmination of literature has both proven that handling of errors in a simplistic way improves usability and also that high-level abstraction of tasks is beneficial to both experts and novices.

B. Process Modelling

Process modelling is the visual or graphical representation of workflows that has attempted to make robotics more attractive and digestible to the general public. Process modelling can be a powerful tool with respect to error handling as it can be leveraged to pinpoint error states or even causes in order to facilitate recovery and handling of robotics. Common examples include unified modelling languages (UML) or unified modelling languages suitable for programming (UML/P) [4], [5], task-graphs [1], [6], Petri-nets [7] and flowcharts [3]. Generally these graphing tools are used for two purposes; informing/visualizing a process and allowing some interface to bypass hard coding. Whilst [4] and [5] have taken a modular interactive user interface approach to their modelling process, certain degrees of expertise are required to not only understand levels of abstraction but also sequencing actions. In [1], [6], sequencing of actions and expertise in terms of generating such

[1]Johann Darboven is a graduate student with the Faculty of Precision Engineering, The University of Tokyo, Tokyo, Japan darboven@g.ecc.u-tokyo.ac.jp

[2]Ryota Takamido and Jun Ota are with the Research into Artifacts Center for Engineering (RACE), School of Engineering, The University of Tokyo, Japan takamido@race.t.u-tokyo.ac.jp, ota@race.t.u-tokyo.ac.jp

is largely simplified for non-experts however, task-graphs display degrees of information such as decision states, nodes and parallel actions that may be better suited for well-versed robotics/software engineers only. Petri-nets similarly prove to be a useful tool for designing parallel processes in robotics and allow for easy implementation of modular sequences as in [7], though more complicated actions tend to require deeper understanding of Petri-net models.

C. *Learning from Demonstration*

Learning from Demonstration (LfD) is a practice by which a human operator demonstrates an ideal motion for a robot to follow, not too uncommonly in the case of errors as well. Although LfD can take many shapes or forms, most commonly, verbal and physical cues are implemented. The merit of LfD and even Programming by Demonstration (PbD) has widely been documented [1], [6], [8], [9], [10]. Studies like [9] and [10], sufficiently support their claims to the merit of LfD. Most explicitly these studies discuss the benefits of kinesthetic teaching, the ability to physically instruct a robot by either demonstrating a task or guiding a robot into positions by explicitly moving joints or hands. In the case of [10]'s case, human demonstration was even used as the ”golden standard” to teach a robot with a sampling-based approach to imitate human motion.

In error recovery scenarios however, sometimes it suffices for a singular way-point to be demonstrated, instead of an entire trajectory such that a motion planner invoked can solve the remaining path in avoiding collisions or obstacles.

D. *Usability*

Finally, usability is a key concept in any HRI. It encompasses the ease of use, intuitiveness, accessibility, and efficiency of interactions between humans and robots. According to [11] usability is positively correlated with HRI-trust and in turn increases an individuals' willingness to use a robot. In lieu of these findings, [12] concludes that although usability can tackle technological limitations, standardized benchmarks are pertinent to identifying bottlenecks of usability and acceptance. The study conducted in [13] outlines a robot mimicking actions through the use of wearable sensor gloves. In line with the previous studies, it is unclear whether this approach is preferable to novice users or is mostly aimed at simplifying a robotics engineers' responsibilities.

II. System Framework

The purpose of this system framework is to allow robotics to be more easily accessible to novice users, especially when it comes to error handling. A problem that previous literature touch on is the complexity of imitating human motion and the incorporated processes which poses the question whether HRI should be geared towards improved human imitation or adaptation. Adaptation focuses on tailoring a robot systems' capacities to a humans capabilities rather than the inverse. We propose an interactive robot monitoring and control system (IRMCS). The IRMCS error handling and improved usability to a robot system that is adapted to a novices' capabilities

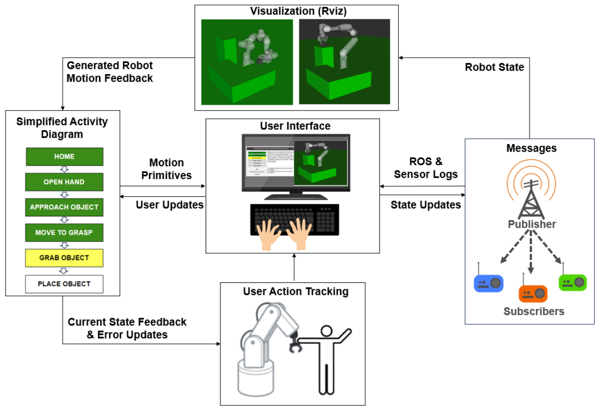

Fig. 1: IRMCS Framework with user-interface, data-flows between monitoring and control processes and visualization. An activity diagram, error messages and robot states are updated in real-time

rather than managing a users interaction with a robot in accordance to its limitations.

Figure 1 shows the overview of the proposed framework IRMCS. The IRMCS incorporates error handling and motion planning abstracted to a high-level overview that permits novices to interact with a robotic system in a novel way. The activity diagram shows the robot primitives in real-time and updates based on user input. Furthermore the IRMCS tracks ROS planning and sensor messages to update the user-interface and post visual clues in the simulation space (Rviz). Finally, all actions conducted by the user are tracked and posted to the user-interface such that all abstracted low-level actions are posted as simple actions such as a ”Reset” or a ”Collision object added to the scene”.

The system was designed with usability in mind. We consider usability based off of four categories:

- Ease of Use - the ability to interact with the system and robot effectively,
- Prior Knowledge - whether using the robot or system requires any domain expertise,
- Understanding - comprehension of displayed information and actions carried,
- Time - Total elapsed time required to complete a task.

The IRMCS was built using a Franka Emika Panda 7-DOF (Degrees of Freedom) arm in the Robot Operating System (ROS) and leverages the standard simulation and visualization tool Rviz. The MoveIt motion planning framework was used and the system was developed in pythonic code. The system is meant to be applicable in environments that do not have rigidly defined scenes and are more prone to errors.

A. *Simplified Activity Diagram*

Though classic activity diagrams can be highly informative to the designer, confusion may arise in real-time scenarios when a novice interfaces with a robotic system. To mitigate information loss and clarify when action is required of the user, the activity diagram in Fig. 1 has been designed for the purpose of an experiment. Here, the GUI has been abstracted to include only user-relevant information. Green, yellow and red colours indicate that a task is completed, is in progress or has failed, respectively. The success of the activity diagram is defined by a users' ability to understand a process and extrapolate a possible source of error as highlighted by motion primitives. The GUI, or IRMCS, streams information about error statuses and processes in real-time. For simplicity, the Activity Diagram includes a total of six, task abstractions. Users can follow along in real-time as a robot goes to a home location, opens its hand, approaches an object, moves to a grasp location, grabs the object and places an object. The tasks are abstracted by setting a set of constraints including a known home location, virtual gripper limits and attachment checks (object attached check in ROS), as well as displacement checks. A separate thread in the system continuously iterates to confirm whether a threshold of these constraints has been crossed to update the Activity Diagram.

B. Error Classification and Recovery

In ROS and typical robotics applications some common errors or error sources include:

- Motion Planning Errors,
- Collisions with Obstacles,
- Environmental changes

Although there are various methods to deal with such dynamic circumstances and errors, most commonly a designer will adapt by inspecting the system log (ROS log text), updating trajectories and goal states, updating the motion planner and planning scenes, defining joint configurations, adding intermediate goals for testing and many more similar approaches. In a defined processing thread, the IRMCS, leverages opencv and tesseract, open-source libraries that help process visual inspections of a query state that the simulation in Rviz provides. The joints display is captured, converted to binarized gray-scale images and converted to strings to read a joint degree value that is stored and passed to the robot manager to either create an obstacle or new trajectories.

Motion planning errors are commonly posted as nodes that can be inspected ("subscribed to") and are visible in the command log of the initialization of the ROS planning scene. Common error logs as published by ROS are posted in Fig 2. A separate thread in the system iterates through this log to identify these common error messages and suggest a number of solutions. To alert a user of collisions or failed grasps, sensor messages from the robots' description are passed as messages to the IRMCS, which a user can either respond to by avoiding obstacles, retrying motions or defining restricted movement zones.

Similarly to the ROS log, these messages can be subscribed to and passed to the IRMCS. The system uses a naive string matching algorithm that searches for keywords such as "goal tree", "insufficient" and matches these to a limited database of experimentally determined words indicating sources of error. The algorithm organizes the sought out strings by severity and passes error-level messages to the user and then based on the warn- and info-level message

```
[ERROR] [1693914500.522907380]
panda_arm/panda_arm: Unable to
sample any valid states for goal
tree [INFO] [1693914500.522998174]
panda_arm/panda_arm: Created
1 states (1 start + 0 goal)
[ERROR] [1707138963.881788661]
panda_arm/panda_arm: Insufficient
states in sampleable goal region
[WARN] [1707138963.88196226]
ParallelPlan::solve(): Unable to
find solution by any threads [INFO]
[1707138963.906282313] ABORTED:
TIMED_OUT
```

Fig. 2: ROS log example lines indicating either self-collisions, obstacles collisions or planning time outs.

makes an appropriate action suggestion which in turn most commonly suggests a reset and new configuration of goal joint trajectories for the sake of simplicity. In the case of no matches, a persisting error triggers the robot to return to a safe position.

Finally, environmental changes may cause errors, unseen error states, or require an updated definition of actions. A separate thread enables a user-defined collision object (obstacle) to be added to the planning scene through the IRMCS with the simulation space. The reasoning is that these error handling methods not only generalize the indefinite error sources that a designer usually is faced with but also abstract the expertise to improve understanding of a task. The strategies to deal with errors are not designed to imitate human motions rather than adapt to what a user and especially a novice to robotics is capable of intuitively. Instead of exactly locating an error source a novice is more likely to understand an error recovery strategy if the solution proposed is simplified to primitive actions.

III. Experimental Procedure

To verify the effectiveness of our proposed framework (IRMCS), four volunteers, two male and two female participants in the age range 24-32 years were selected. The subjects have no programming or robotics experience. This experiment consisted of comparing the outlined system to a control experiment in a standard pick and place task. Participants (P1-P4) were required to observe a task and fix one or more errors as described in Section II-B.

A. Pick and Place Task

For a simple standard pick and place task, four different scenarios were developed. An example of such is depicted in Fig. 3. In an Rviz simulation setup, two counters and a box are used to define the start and end goals of the object of manipulation respectively. A video of the ideal scenario is shown for explanation such that users understand what the objective of the task is. The four scenarios are as follows:

1) Simple pick and place task with defined start and end goals

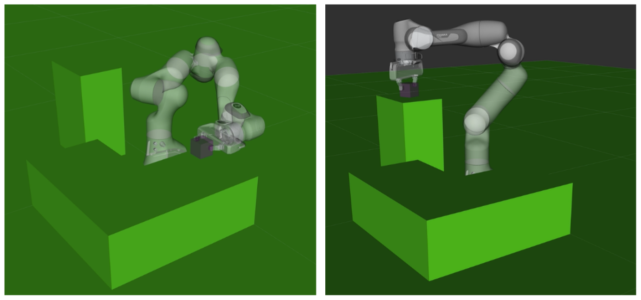

Fig. 3: Franka Panda arm in Rviz. Left: Start position of box placement, Right: Goal position of box placement on top of a user-created collision object.

2) Pick and place task with an added obstacle unknown to the motion planner
3) Pick and place task and a definition of an obstacle
4) Pick and place task with a definition of an obstacle and an undefined goal position

Given that industrial work environments already require a high degree of error handling to be implemented, it is hypothesized that a less defined work environment is worth studying as both non-expert users and variable conditions typically lead to more error states.

B. User Requirements

Assuming that the most effective and highest scoring usability will be accredited to systems that require the least amount of time of completion of a task and amount of supplementary knowledge, the users' role was defined with simplicity in mind. Imperative to completion of a task is reading and following instructions such as *"Push 'Start'"* or *"Drag the robot hand using the mouse to a desired location"*. Since the experiment was conducted in the standard ROS compatible Rviz simulation environment, the closest LfD method that resembles kinesthetic teaching is manual manipulation of the robot by moving either individual joints or moving an end-effector to a target location using a mouse. In each of the experiments a starting motion is demonstrated after which either a motion planning error, a collision or an environmental change error is detected. The IRMCS then suggests to either solve the error by dragging the robot arm to a safe position, defining a restricted area, resetting the robot and moving to a different goal position or by creating an obstacle by moving the robots' end-effector to three different positions in the simulation space to define its' limits. The activity diagram updates in real-time between yellow, green, red or transparent box colors to inform the user of an ongoing, completed, failed or not started process respectively (see Fig 1). The GUI includes two separate message boxes for instructions and process updates. Using the IRMCS, novices are empowered to add obstacles to the planning scene, define off-limit zones, edit goal positions and correct a multitude of errors that may arise for different reasons.

TABLE I: METRICS SUMMARY

Metric	Description
Attempts	Number of times the user attempted a motion before succeeding
Path Length(rad)	Effort in euclidean distance, a measure of rotation added up amongst all joints in the kinematic chain of the robotic arm
Execution Time(s)	Amount of time elapsed between the start of an experiment until the object is placed or a user has given up
Usability	A measure defined by a points system in which the participants evaluate the experiment based on ease of use (10 points), knowledge required (10 points, more points indicate less knowledge required), understanding of the robotic process (10 points) and total elapsed time (5 points)

C. Control Experiment

To compare the usability of the IRMCS to manual coding, a control experiment was conducted. The control experiment (i.e. the status quo), ROS and python coding is briefly explained to the participants as well as the relationships between ROS, the simulation (Rviz) and the code. Users were given full access to online resources, including google, chatgpt, stackoverflow and rosanswers. Requirements are to equally complete the four tasks and errors that show up on either simulation space, ROS log or command line that indicates the error modes outlined in Section II. The users were given as much time as they needed and the option to give up whenever they reached an impasse.

D. Metrics

In order to evaluate the success of the system in this experiment, metrics were derived and adapted from [14], which uses a bench-marking test with a block robotics pick and place task. Table I shows a summary of the metrics. Usability is scored out of 35 points and determined from the user survey. Time elapsed is weighed less, given the difficulties of comparison between a control experiment and the system.

In the *Results & Discussion* section, path lengths of each user are compared to the "ideal" outcome, in which the designer completed the task as intended by coding and fixing errors with the traditional approach.

IV. RESULTS AND DISCUSSION

The success rate of the control experiment was 0% amongst all four candidates whilst in our system it was 94% (15/16 successful conditions). From the feedback gathered in our survey, participants noted that usability was high and understandable. The control condition revealed that planning motion, re-programming and teaching a robot is too complicated for novice users as expected. Being able to search for answers is also an intrinsic skill that robotic programmers develop with experience and domain expertise. The IRMCS results on the other hand, demonstrate that novices can interact with a robotic system on a higher level and are able to perform basic motion planning and error handling tasks. The experimental results are derived from the simulation data and the user survey collected after completion of all tasks.

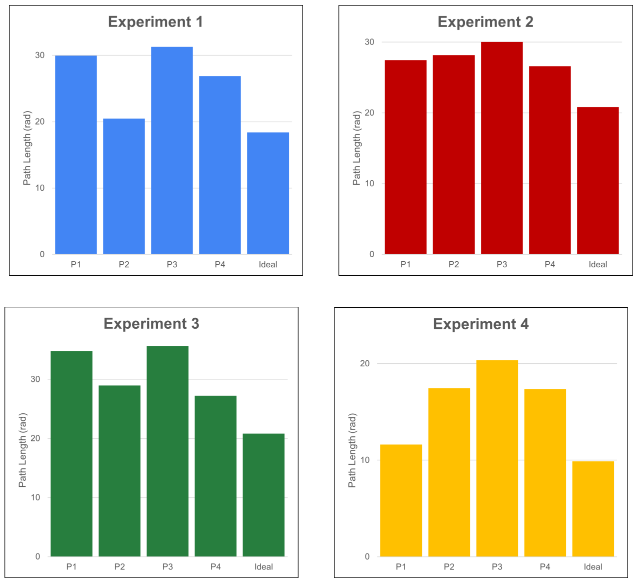

Fig. 4: Total Path length plotted that each participant created for the robotic tasks compared to an "Ideal" professionally planned program.

However, since participants were unable to successfully plan motion or recover from error states in the control condition, we excluded the data and compare the IRMCS experimental results with an ideally programmed result by a robotics expert.

A. Simulation Results

In Fig. 4, Experiment 1-4 refers to the scenarios outlined in Section III-A. The path length refers to the added path length the robot arm covered over each consecutive primitive/move. Users were able to finish Experiment 1 in 1 attempt with 5-6 moves, Experiment 2 in 3 attempts with 5-6 moves and Experiments 3 and 4 in 2 attempts with 5 moves. The ideally programmed path takes 4 moves for each experiment. In Experiments 1-4, each and every participant was successful in creating new robotic movements, interacting with the simulation environment in a meaningful way and understand the high-level processes. In Experiments 1-3 participants had to make sure the block would be placed on the opposite diagonal of the counter. Experiment 1 shows that all participants completed the task in 5 or 6 moves within 4 minutes, all participants were also able to complete this task within one try. In the second and third experiment the path was blocked and all users with the exception of one were able to avoid the obstacle by adding intermediary goals or add collision objects to the planning scene to automatically do so. In the last experiment, depicted by Fig. 3 three out of four participants were successful in defining an obstacle and placing the block on top of said obstacle. In the third and fourth experiment however, two participants required 3 and 4 trials until completion. The experimental evaluation shows that at the slight cost of increased path length/effort, a novice was able to understand a robotic process at a high-level as well as interact with the IRMCS to fix errors. Furthermore, the fact that participants 1-4 were able to add at least 1 way-point in each task on the way to the desired location suggests that users knew what to expect when moving a robot though clear trajectories were not defined.

TABLE II: TIME AND PATH LENGTH DIFFERENCE

Experiment	Path Length Diff.	Execution time(s)
1	48%	224
2	36%	189
3	52%	327
4	69%	446

TABLE III: USABILITY SURVEY RESULTS

Variable	IRMCS	Control
Ease of Use	7.75/10	1.25/10
Prior Knowledge	5.25/10	1.25/10
Understanding	8.25/10	2.50/10
Time	2.75/5	1.00/5
Total	**24/35**	**6/35**

Table II, shows the increase in path length between the average participants' created paths compared to an ideally programmed path. The execution time is also an average of the time from start to completion of a scenario including the user-end developed error handling methods. Given that the novices participants demonstrated an ability to re-plan motion and recover from an error, the moderate percentage increase in path length to an "ideal" scenario, which is based off of a motion planners ability to connect one point to another, is commendable. Most novices chose to plan their motions to move the robot somewhere between the start and end goal and even when outside of a given threshold tolerance this contributed to a trajectory that had the purpose of achieving the goal. Comparing to the status quo, where all participants gave up after 11 minutes of no successful motion plans, execution time is also a noteworthy marker that the IRMCS was able to perform well at.

B. User Survey

Users evaluated the usability of the system and the usability of the control experiment. In Table III, usability scores amongst the four categories defined in Section I-D were averaged for both the IRMCS experiment and the control experiment amongst all four participants. The IRMCS scored particularly high in the "ease of use" and "understanding" category which implies that the system was intuitive enough for interaction but prior knowledge such as explanations of ROS and simulation spaces were necessary to a degree. Further data of the control experiments was not usable given that all participants gave up after roughly 11 minutes and had achieved no significant results even with online resources.

V. CONCLUSION

In conclusion, our proposed framework, the IRMCS, can significantly increase error handling success rate up to 94% and likely positively influences adoption through improved usability. The results suggest that promising progress can be made by enhancing user understanding and empowering novices to undertake formerly inaccessible robotic tasks. The results and questionnaire indicate that the most significant contribution in guiding novices was illustrating a clear relationship between user input and system output, a relationship that is shrouded at best, when it comes to programming a robot. Though promising efforts were made, further testing with a larger number of participants and scenarios is required as well as a practical implementation with a physical robot. Simulation results were especially promising given that with little effort, otherwise highly involved error handling strategies were simplified such that with the IRMCS novices were able to fix an error. Not having to understand an error source but being able to pinpoint the primitive in which an error occurred greatly helped the users' understanding of the error recovery process. The experiment was somewhat limited to the amount of errors that the system can search through though more of these instances can be added to the search space.

Future work will focus on including more complicated error cases as well as testing out practical application in dynamic workplaces.

REFERENCES

[1] T. Eiband, C. Willbald, I. Tannert, B. Weber and D. Lee, "Collaborative programming of robotic task decisions and recovery behaviors," *Autonomous Robots*, vol. 47, no. 2, pp. 229–247, 2022.

[2] A. Nakamura, K. Nagata, K. Harada, N. Yamanobe, T. Tsuji, T. Foissotte and Y. Kawai, "Error Recovery Using Task Stratification and Error Classification for Manipulation Robots in Various Fields," *IEEE/RSJ International Conference on Intelligent Robots and Systems [Preprint]*, 2013.

[3] E. Kristiansen, E. Krabbe Nielsen, L. Hansen and D. Bourne, "A Novel Strategy for Automatic Error Classification and Error Recovery for Robotic Assembly in Flexible Production," *Journal of Intelligent & Robotic Systems*, vol. 100, no. 3-4, pp. 863–877, 2020.

[4] U. Thomas, G. Hirzinger, B. Rumpe, C. Schulze and A. Wortmann, "A New Skill Based Robot Programming Language Using UML/P Statecharts," *IEEE International on Robotics and Automation [Preprint]*, 2013.

[5] Pi. A. Akiki, Pa. A. Akiki, A. K. Bandara and Y. Yu, "EUD-MARS: End-user development of model-driven adaptive robotics software systems," *Science of Computer Programming*, vol. 200, 2020.

[6] R. Caccavale, M. Saveriano, A. Finzi and D. Lee, "Kinesthetic teaching and attentional supervision of structured tasks in human–robot interaction," *Autonomous Robots*, vol. 43, no. 6, pp. 1291–1307, 2018.

[7] P.B. Kumar, H. Rawat and D.R Parhi, "Path planning of humanoids based on artificial potential field method in unknown environments," *Expert Systems*, vol. 36, no. 2, 2018.

[8] D. Lee and C. Ott, "Incremental kinesthetic teaching of motion primitives using the motion refinement tube," *Autonomous Robots*, vol. 31, no. 2-3, pp. 115–131, 2011.

[9] B. Akgun and K. Subramanian, "Robot Learning from Demonstration: Kinesthetic Teaching vs. Teleoperation," 2011.

[10] G. Ye and R. Alterovitz, "Demonstration-Guided Motion Planning," *Springer Tracts in Advanced Robotics*, pp. 291–307, 2016.

[11] M. Babamiri, R. Heidarimoghadam, F. Ghasemi, L. Tapak, A. Mortezapour, "Insights into the relationship between usability and willingness to use a robot in the future workplaces: Studying the mediating role of trust and the moderating roles of age and STARA," *PLOS ONE*, vol. 17, no. 6, 2022.

[12] J.T. Meyer, R. Gassert and O. Lambercy, "An analysis of usability evaluation practices and contexts of use in wearable robotics," 2021.

[13] N. Garcia, J. Rosell and R. Suarez, "Motion Planning by Demonstration With Human-Likeness Evaluation for Dual-Arm Robots," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 11, pp. 2298–2307, 2019.

[14] A.S. Morgan, K. Hang, W.G. Bircher, F. M. Alladkani, A. Gandhi, B. Calli and A. M. Dollar, "Benchmarking Cluttered Robot Pick-and-Place Manipulation With the Box and Blocks Test," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 454–461, 2020.