FACILITATING RELIABLE AUTONOMY WITH HUMAN-ROBOT INTERACTION

A Dissertation
Presented to
The Academic Faculty

By

Siddhartha Banerjee

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Interactive Computing
College of Computing

Georgia Institute of Technology

May 2021

FACILITATING RELIABLE AUTONOMY WITH HUMAN-ROBOT INTERACTION

Thesis committee:

Dr. Sonia Chernova *(advisor)*
School of Interactive Computing
Georgia Institute of Technology

Dr. Matthew Gombolay
School of Interactive Computing
Georgia Institute of Technology

Dr. Karen Feigh
Aerospace Engineering
Georgia Institute of Technology

Dr. Laurel Riek
Computer Science and Engineering
UC San Diego

Dr. Sean Andrist
Adaptive Systems and Interaction Group
Microsoft Research

Date approved: April, 2021

Learning from direct experience is more effective when coupled with reflection . . . We don't learn from experience, we learn from reflecting on experience.

Jonathan Doerr

ACKNOWLEDGMENTS

The work in this dissertation would not have been possible without any of the vast amount of support from my advisers, friends, and family. The oft-quoted, "standing on the shoulders of giants," metaphor rings true for this thesis: I simply add that all the giants who supported me provided an exceptionally stable foundation. I feel fortunate to have worked in such an environment and I cannot thank you all enough.

The first thanks must be given to my advisor, Dr. Sonia Chernova. I am not sure what she saw in the aspiring Ph.D. student (me) who met with her one afternoon in October 2015, but whatever it was, she invested a lot of time and effort into helping me grow and mature as a roboticist and an adult. She championed my extra-Ph.D. activities (e.g. letting me disappear for months during the Mongol Rally), helped improve my writing, supported my flights of fancy (e.g. funding me to attend a conference causal learning & inference), provided much needed perspective when I was lost in the weeds or unable to make sense of experimental results, and (most importantly for this document) encouraged me to not give up on this dissertation. She took great pains to create a lab environment where I was comfortable, and I truly appreciated her candor at times in answering questions that others might not have been as willing to answer. Sonia is a wonderful mentor, a great advisor, and a role-model for me to emulate as I continue beyond the Ph.D.

I have been lucky to have other brilliant mentors too. My committee members, Dr. Matthew Gombolay, Dr. Sean Andrist, Dr. Karen Feigh, and Dr. Laurel Riek, have all been extremely helpful and given me invaluable feedback for the work in this dissertation. I truly appreciate the time that all of them have devoted to activities ranging from brainstorming and designing experiments, to analyzing the data and presenting the results. My mentors at my internships—Sean and Dr. Dan Bohus at Microsoft Research, and Dr. Vivian Chu and Dr. Andrea Thomaz, with the rest of the Diligent team, at Diligent Robotics—shaped my perspective on robotics and improved my technical know-how. Finally, my mentors

in the GT IRIM faculty, such as Matthew and Dr. Seth Hutchinson, helped me thrive in the robotics community at GT and helped me integrate with the larger robotics community outside of the school. Thank you all.

My good fortune in friends provided an excellent complement to the guidance I received from my mentors: they created the environment in which I could do the work for this dissertation. My current and former labmates in the RAIL lab—Harish Ravichandar, Reza Ahmadzadeh, Vivian Chu, Kalesha Bullard, Tesca Fitzgerald, Lakshmi Nair, Asif Rana, David Kent, Andrew Silva, Angel Daruna, Jonathan Balloch, Weiyu Liu, Devleena Das, Glen Neville, among many others—kept me sane, enjoyed RAIL days with me, and took on challenges such as doling out Halloween candy or crushing FetchIt. Friends in my cohort and the broader GT graduate student community—Ian Stewart, Kelsey Kurzeja, Lara Martin, Shray Bansal, Samarth Brahmbhatt, Upol Ehsan, and many more—helped me settle into Atlanta by creating an environment that was fun and supportive. Even the encouragement of and conversations with my friends outside of Atlanta was instrumental in motivating me to complete this dissertation. Finally, I am especially grateful the support of my partner, Ceara Byrne, who was a bedrock during the COVID-19 pandemic and without whose help, every bit of the last year of this Ph.D. would have been more difficult.

Last but not least, I must thank my family, without whose unwavering support, I would not be where I am today. I know that I haven't been the most forthcoming of individuals when updating them on my progress through the Ph.D., but they have taken my reticence in stride and cheered even the smallest bits of news that they have received. Thank you for letting me follow my dream: I could not have been more lucky, and I definitely could not have done this without you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

MTBI Mean-Time Between Interventions

MTCI Mean-Time Completing Interventions

SUMMARY

Autonomous robots are increasingly deployed to complex environments in which we cannot predict all possible failure cases a priori. Robustness to failures can be provided by humans enacting the roles of:

- *developers* who can iteratively incorporate robustness into the robot system,
- *collocated bystanders* who can be approached for aid, and
- *remote teleoperators* who can be contacted for guidance.

However, assisting the robot in any of these roles can place demands on the time or effort of the human. This dissertation develops modules to reduce the frequency and duration of failure interventions in order to increase the reliability of autonomous robots, while also reducing the demand on humans. In pursuit of that goal, the dissertation makes the following contributions:

1. A development paradigm for autonomous robots that separates task specification from error recovery. The paradigm reduces burden on developers while making the robot robust to failures.
2. A model for gauging the interruptibility of collocated humans. A human-subjects study shows that using the model can reduce the time expended by the robot during failure recovery.
3. A human-subjects experiment on the effects of decision support provided to remote operators during failures. The results show that humans need both diagnosis and action recommendations as decision support during an intervention.
4. An evaluation of model features and unstructured Machine Learning (ML) techniques in pursuit of learning robust suggestions models from intervention data, in

order to reduce developer effort. The results indicate that careful crafting of features can lead to improved performance, but that without such feature selection, current ML algorithms lack robustness in addressing a domain where the robot's observations are heavily influenced by the user's actions.

CHAPTER 1
INTRODUCTION

Autonomous robots are rapidly becoming prevalent in homes [1, 2, 3], hospitals [4, 5], hotels [6], shopping malls [7, 8], factories [9], the military [10], and space [11, 12, 13]. In such diverse settings, the robots accomplish a wide array of tasks, including providing casual social interaction or reminders for medication [2, 3], delivering laboratory specimens [14, 15] and factory inventory [16, 17], providing directions and other services [18, 19, 7], exploring new worlds [12, 13], and assisting in disaster response [20]. In fact, the use of robots during the COVID-19 pandemic highlights both the widespread adoption of robots in the present, as well as opportunities for an even more ubiquitous adoption of robots in the future [21, 22]. In these circumstances, the reliability of the robots is of paramount importance.

There are two approaches to improving robot reliability, both of which we explore in this dissertation. The first solution is to develop more robust and intelligent autonomous systems that can handle a wider range of operating conditions and independently recover from errors. Such techniques reduce the frequency with which failures occur. However, no known framework exists that can eliminate all robot failures [23]. Therefore, the second solution is to reduce the duration of a failure when one does occur. In such cases, human assistance in the form of an *intervention* is required for the robot to resume autonomy [24, 25, 26, 27]. Interventions are burdensome on human operators because they may (1) interrupt the human in their ongoing task, (2) require humans to make an effort at diagnosing the robot's problems, and (3) demand human guidance to recover autonomy. As such, robot reliability should be improved by considering the additional pressures that the improvements might stipulate on the human stakeholders in the process, with the goal of reducing the human effort and time required during the resulting intervention. This dissertation seeks to

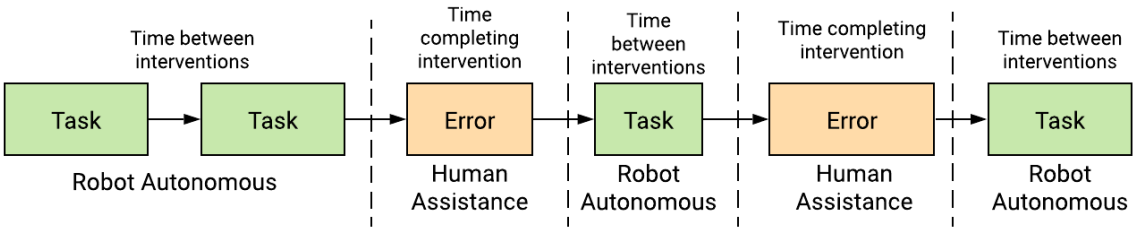


Figure 1.1: The frequency and duration of errors in tasks can characterize robot reliability. Metrics such as MTBI and MTCI respectively can then be used to quantify that reliability.

improve robot reliability with such considerations.

1.1 Dissertation Overview

In this dissertation, we characterize robot reliability through two properties – **frequency of failures** and **duration of failures** during robot task execution (Figure 1.1)[1].

The *frequency of failures* is measured by metrics such as the Mean-Time Between Interventions (MTBI), which is the mean time that the robot operates nominally without failures. Reliability is improved by reducing the frequency of failures, i.e. increasing MTBI, and it is dependent on the inherent autonomy of the robot agent or on the frequency of failures of system components [31]. As such, updating the components or the code of a robot can reduce the frequency of failures but the task requires *significant* human engineering effort. In fact, as evidenced by the robot failures in the DARPA Robotics Challenge, even after expending considerable effort in the development of robot systems [32, 33, 34], teams of experienced roboticists struggled to reduce the frequency of failures [35, 36, 37]. This finding motivates the need for agile methods of development, or for easy-to-implement system components, which reduces system engineering effort.

The *duration of failures* is measured by metrics such as Mean-Time Completing Interventions (MTCI), which is the mean duration of the failures during which the robot awaits

[1]We assume the ability to differentiate between moments when the robot is in an error and when it is not through the use of automated fault detection. Fault detection need not be straightforward and is an active area of robotics research [28, 29, 30]. However, regardless of how a fault or error is detected, the frequency and duration of the detected error(s) can characterize the reliability of the robot.

human assistance. Reliability is improved by decreasing the duration of failure, i.e. decreasing MTCI, which in turn is affected by the communication lag between the human and robot, the information available to the human or robot about the failure, etc. [31]. Therefore, reducing the amount of time between a failure and the robot's receiving of help for it, or improving the awareness of the failure for both the human and the robot, can help reduce the duration of the failure. Prior work has identified trade-offs in the appropriateness of requesting assistance from collocated human helpers vs. remote human operators, as well as in the quality of the assistance received from them [38, 39]. The finding thus motivates a need to investigate methods of garnering assistance from humans in order to reduce failure duration while also minimizing the costs to operators.

In the following chapters, we focus on reducing the frequency and duration of failures while also reducing any exigencies placed upon human stakeholders in the process. In particular, we focus on the demands made of human stakeholders who enact the following three roles identified above:

Developers are designers and engineers who help define a robot system's capabilities and its interface(s) with human operators. They often create new system components, improve component reliabilities, or orchestrate components together in new ways to improve autonomy or improve interactions with operators. As such, their efforts have the ability to reduce both the frequency and duration of failures.

Collocated bystanders are humans who are physically co-present with a robot in its operating environment and can help reduce the duration of a failure if they are approached by the robot for aid. We assume that bystanders have high situational awareness and can provide high-quality assistance, if approached. However, the request for assistance creates an interruption that can place unwanted demands on the bystander.

Remote teleoperators, by contrast, are humans who are physically removed from the robot's operating environment, but who can use teleoperation interfaces to decrease the dura-

tion of failure if they are contacted by the robot for aid. We assume that teleoperators are always available, but they have to face the demands of teleoperation, including low situational awareness, which can lead to low-quality assistance.

1.2 Thesis Statement

This dissertation develops modules to reduce the frequency and duration of failure interventions in order to increase the reliability of autonomous robots, while also reducing the demand on humans.

1.3 Contributions

The modules that we contribute are:

1. **Recovery-Driven Development** (Chapter 2): We formalized a development paradigm for robots that separates task specification from error recovery. The paradigm reduces the burden on developers while making a robot robust to failures. The paradigm, and the system developed to support it, enabled our robot to perform pick and place tasks autonomously for over 45 minutes without interventions at the FetchIt! Challenge at ICRA 2019 [40].

2. **An evaluation of a model for human interruptibility** (Chapter 3): We developed a model for classifying the interruptibility of collocated humans using the data from the on-board sensors of a robot [41]. A subsequent human-subjects study deployed the model online on a robot to show that the model was effective at gauging interruptibility and that the using the model can reduce the time expended by the robot in garnering assistance for failure recovery [42].

3. **An evaluation of decision support for remote operators** (Chapter 5): We conducted a large-scale human-subjects experiment on the effects of suggestion type

and suggestion accuracy when decision support is provided to remote operators during failure interventions. Our results show that providing operators with both diagnosis suggestions and action recommendations as decision support provides maximal benefit [43].

4. **An evaluation of the robustness of decision support models learned from data** (Chapter 6): In a bid to reduce developer effort, we conducted an evaluation of model features and unstructured Machine Learning (ML) techniques in pursuit of learning robust suggestions models from intervention data. Our results show that careful crafting of features can lead to improved performance, but that without such feature selection, current ML algorithms lack robustness in addressing a domain where the robot's observations are heavily influenced by the user's actions.

1.4 Outline of Dissertation Document

This dissertation is organized as follows. Chapter 2 introduces Recovery-Driven Development and focuses on reducing the *frequency* of failures while reducing the development effort of creating a reliable robot system. Chapter 3 transitions to focus on decreasing the *duration* of failure while remaining cognizant of the interruptibility of collocated bystanders. Chapter 4–Chapter 6 then shift the focus to ameliorating the effort of remote teleoperators and of developers while decreasing the *duration* of failures. Specifically, Chapter 4 introduces a human-subjects study that we conducted to characterize the complex interactions that occur between teleoperators and the decision support models that might be provided to them during an intervention. We then further investigate the interactions separately in the subsequent chapters. In Chapter 5, we investigate the effects of decision support on remote operator performance, and in Chapter 6, we investigate the effects of teleoperator behaviour on the robustness of decision support models. In all the chapters, from Chapter 2–Chapter 6, we introduce and discuss the relevant background and related works in order to contextualize our investigations. Finally, we provide concluding

remarks and discuss open questions in Chapter 7.

CHAPTER 2

RECOVERY-DRIVEN DEVELOPMENT FOR RECIPE-BASED ROBOT TASKS

Robot execution is fragile and often overfits to the development test bed [35, 37]. As such, robust robot architectures must rely on recovery behaviors in order to maintain autonomy when assumptions are violated [44]. Recovery during robot tasks is non-trivial, however, as resetting to a known state can be difficult [45] and knowing where to resume execution can be context dependent [46]. In addition, unforeseen faults create ambiguity in recovery strategies.

In this chapter, we look to improve robot autonomy (i.e., decrease the *frequency* of failures), while minimizing the development effort of doing so. Specifically, we address the development of robust recovery for recipe-based tasks—a class of robot tasks that dictate a pre-specified sequence of steps to accomplish a goal. Such tasks include common mobile manipulation tasks in unstructured environments, such as kit packing, machine assembly, table setting, or food preparation. Even for seemingly straightforward recipe-based tasks, the many interactions with the environment, and between robot components, lead to faults that are difficult to identify *a priori* [23], often resulting in systems that are inflexible or not robust to failures.

State machines [46], hybrid automata [47], and planning approaches [48] are common methods of sequencing robot execution that can be made robust to failures. However, robustness is often achieved at the cost of a complexity explosion in the task sequence specification or a loss of interpretability of the task recipe. Crucially, the increased complexity and the lack of interpretability negatively impact the iterative development of the main task and recovery processes, both of which are necessary in the face of potentially innumerable failure conditions.

We therefore propose *Recovery-Driven Development* (RDD), a development process for

recipe-based tasks couched in agile methodology. The key tenet of RDD is the separation of nominal task specification from recovery behavior definition. Another guiding principle of RDD is the support of hierarchical task specification, which both allows for re-use in the task recipe and provides higher-level context to recovery behavior selection. As such, the RDD methodology enables system developers to easily explore aspects of a robot's system design, such as those identified by Eppner et al. [47]—assumptions, generality, modularity, etc.

We define RDD as a 2-pronged iterative approach to developing robust task execution, in which designers can move back and forth between both prongs without risk of one phase interfering with the other:

1. **Specification** (Section 2.3.1): scripting a hierarchical task sequence incrementally from a task recipe, using strong assumptions

2. **Refinement** (Section 2.3.2): developing recovery behaviors by executing a situated task, noting a fault, specifying new recoveries, and repeating

The result is a development methodology that supports rapid task and recovery prototyping, without a noticeable loss in the robot's robustness when deployed.

In this work, we present our task execution and monitoring system as an example framework designed to enable RDD for recipe-based robot tasks[1]. We validate both the task system and the RDD workflow with our team's winning approach to the FetchIt! Challenge at the IEEE 2019 International Conference on Robotics and Automation (ICRA), in which our success was achieved mainly due to the robustness afforded to our system from the RDD methodology. Additionally, we provide details and open-source code for our complete system developed for the FetchIt! Challenge as a concrete example of a complex mobile manipulation system developed using RDD. We conclude with a discussion of lessons learned for the fast and robust development of recipe-based robot tasks.

[1] https://github.com/GT-RAIL/derail-fetchit-public/tree/master/task_execution

2.1 Related Work

Designing a robot's software is often application-dependent, requiring tradeoffs between multiple approaches [49]. In this section we enumerate common design choices that emerge across applications for robot architectures, situate our task execution framework within the design practices, and motivate the development of our approach.

Robot architectures are generally three-tiered with the following levels [49]:

- A *behavioral* level for highly reactive and highly situated robot execution. Modules at this level, sometimes termed skills, have a tight perception-action loop and are the focus of much research.
- An *executive* level that bridges low-level tasks (skills) and high-level tasks (goals). The executive is responsible for sequencing skills, monitoring execution, and handling exceptions.
- A *planning* level responsible for tasking the executive level with goals to achieve based on future objectives, robot constraints, environmental situations, etc.

Our primary contribution is in enabling the executive level to support an RDD workflow, and as such the remainder of this section examines executive level design and recovery. We discuss a behavioral implementation of mobile manipulation in Section 2.2 to provide context for our executive implementation. We also note that planning-level requirements are minimal for autonomous recipe-based tasks, although we return to this assumption at the end of Section 2.5.

2.1.1 Executive Level Design

The most informative consideration in executive level design is how dynamic or static the task should be. A task can be dynamic due to environments with uncontrolled agents

such as humans [50] or competing objectives [51]. There are four paradigms to behavior sequencing at the executive level, with differing levels of support for dynamic tasks:

- *Agent-based control* partitions control into separate, synchronized agents that maintain consistency with the global robot objective. This works well for dynamic environments, and was implemented through behavior trees in Playful [50] and resource agents in ROAR [52]. However, debugging and reasoning about the interactions between agents can be difficult.

- *Planning* is a principled manner of sequencing skills in dynamic environments, and was implemented by CRAM [48]. However, when designers know the exact sequence of skills they want, as in recipe-based tasks, the design process for planning can be non-intuitive or even counter-productive [46].

- *Finite State Machines* retain some of the autonomy in sequence specification provided by planning, and also allow system designers to explicitly specify state transitions a priori based on expected sub-task outcomes. Additionally, state machines support model verification and composition for incremental construction of complex behaviors [53, 46]. However, state machines, and the related method of hybrid automata [47], suffer from an explosion of transitions as the number of skills or the task complexity grows.

- *Scripting* allows for the compositionality of state machines with the simple declaration, rather than programming, of robot behavior [44]. Additionally, scripting provides easier error recovery to handle exceptions at the executive level than state machines. However, scripting puts the burden of sequence specification on the designer, raising scalability issues, especially for multi-objective tasks [46].

In this work, we focus on relatively static environments and recipe-based tasks that can be decomposed into subtasks. In order to facilitate the rapid prototyping and incremental

inclusion of error recoveries inherent to RDD, we sequence behaviors through hierarchical scripting. We address scripting's scalability issues by separating task specification and recovery.

2.1.2 Reactivity and Recovery

A robust robot executive level must include failover mechanisms that maintain autonomy when behavior design assumptions are not satisfied [44]. Therefore, recovery systems must address many challenges, including determining how to reset to a known state [45], handling context dependent execution resumption [46], and deciding on recovery strategies for unforeseen faults. A common recovery strategy for resetting to a known state is to re-attempt the entire task, as in [47], although such approaches are less reactive to failures.

Planning approaches maintain reactivity by recovering from seen and unforeseen failures by replanning [48]. Further, plans provide theoretical guarantees on robustness to unforeseen execution failures [54, 47]. Prior works have treated recovery as a planning problem with the goal of reaching any state where a diagnosed fault does not exist [55]. However, as noted earlier, planning approaches can be difficult to iteratively develop, or can result in not-easily interpretable task specifications. Our approach avoids planning at the executive layer in favor of scripting, to facilitate rapid and highly-interpretable behavior development.

In the absence of planning, reactively resetting to a good known state can be accomplished through fault forecasting, such as Failure-Modes Effects and Criticality Analysis (FMECA), which reasons about expected faults and the explicit recovery steps to address them [24]. However, such approaches are time consuming and not guaranteed to find all faults [23]. We instead take an empirical approach to fault discovery through situated task execution, exploiting the inherent structure of recipe-based tasks, allowing for pre-scripted recovery to intermediate task steps.

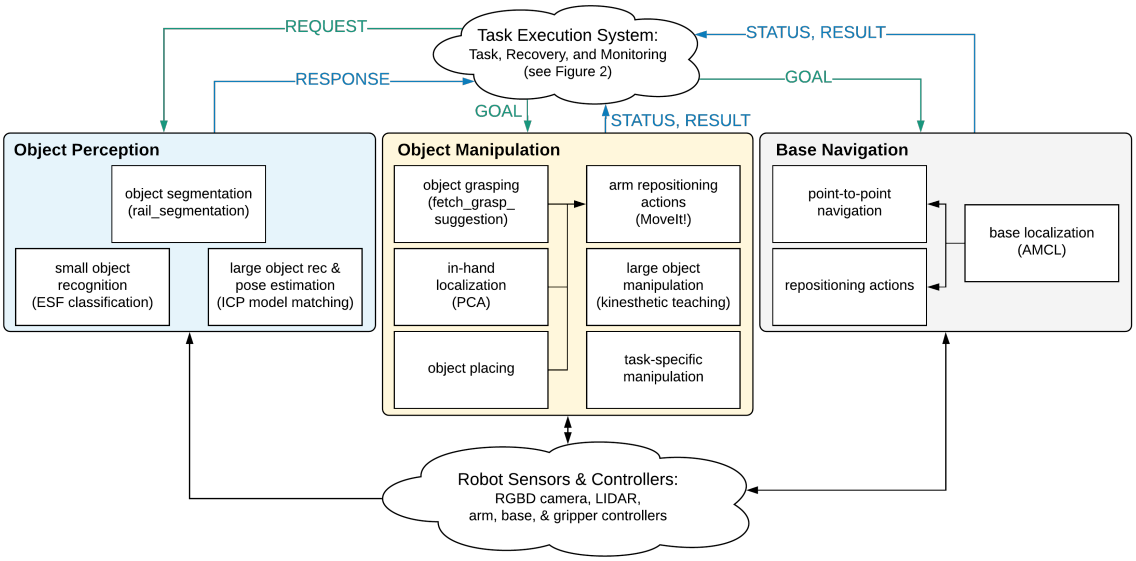


Figure 2.1: Mobile manipulation system overview. Arrows denote ROS information flow, through publishers, subscribers, services, and actionlib.

2.2 System Overview

Before presenting our task execution and recovery system, we first describe the behavioral level of our general mobile manipulation architecture, to serve two purposes: (1) to share our open-source challenge-winning mobile manipulation system developed to support RDD, and (2) to establish a task context and a set of robot capabilities that we will refer to throughout the chapter, grounding our discussion of RDD's benefits and drawbacks in a fully-realized robot system. The architecture consists of a set of independent mobile manipulation modules, implemented using the Robot Operating System (ROS) [56], shown in Figure 2.1. Object perception modules are implemented as ROS service servers, and object manipulation and base navigation modules are implemented as `actionlib`[2] servers. Each independent module can be called by the task executor, and provides feedback to the task executor and task monitor[3]. The modules consist of the following capabilities:

Object Perception. Our perception modules implement a perception pipeline for RGBD

[2]http://wiki.ros.org/actionlib

[3]Each module must necessarily provide feedback on its own faults so that the executive level can make relevant recovery decisions.

sensor data, using the Point Cloud Library (PCL) [57]. The object segmentation module uses the `rail_segmentation`[4] package to identify point cloud clusters-of-interest through table surface detection and Euclidean distance clustering. We divide our object recognition approaches between large and small objects. For large objects, we perform model matching using the `rail_mesh_icp`[5] package that uses an Iterative Closest Point (ICP) PCL pipeline, which also provides object pose detection. For small object recognition, we train an SVM classifier over Ensemble of Shape Functions (ESF) descriptors [58]. We do not need to perform pose estimation for small objects due to our object grasping approach, described below.

Object Manipulation. Most of our manipulation modules make use of MoveIt! to perform arm planning to either joint goals or end-effector pose goals using OMPL's `RRTConnect` motion planner [59]. This includes both general arm repositioning actions, which the task executor can call directly (e.g. to move the arm out of the way of the camera), and execution actions, called by other object manipulation modules. Object grasping calculates antipodal grasps over an object point cloud using the `agile_grasp` package [60], which are then ordered and executed using pairwise ranking through `fetch_grasp_suggestion` [61]. As objects can shift during the grasping process, we perform post-grasp pose detection using the in-hand localization module, which identifies the object point cloud by performing background subtraction on the robot's gripper, and calculates the object's pose based on its principal axes determined by Principal Component Analysis (PCA). Given a known object pose and a desired place location, object placing calculates and executes a pose goal for placing an object that ensures the gripper fingers and palm are out of the way of the object's fall trajectory.

We also include some manipulation modules that do not use MoveIt!, due to the limitations of sampling-based motion planning. For large object manipulation, such as lifting and placing kits of objects, we include a kinesthetic teaching module [62]. This allows

[4]http://wiki.ros.org/rail_segmentation
[5]http://wiki.ros.org/rail_mesh_icp

system designers to record and play back arm trajectories, either in full or as a set of waypoints. Additionally, we include task-specific manipulation actions to implement specific manipulation skills such as raising and lowering objects, using a Cartesian end-effector controller[6], and peg-in-hole insertion, using a controller with end-effector pose and joint effort as feedback.

Base Navigation. LIDAR-based localization uses AMCL provided by ROS's `nav_stack`[7] to localize the base with respect to a pre-collected 2D occupancy grid of the environment. Navigation is primarily done using point-to-point navigation between waypoints on the map, executed using a PID controller[8]. We also include local repositioning actions, which implement short movement primitives such as backing up from a table. The repositioning actions are implemented using a PID controller with gains tuned for shorter, more precise base goals.

With each module implemented, the navigation, perception, and manipulation actions can be sequenced in a robust manner to complete mobile manipulation tasks by the task execution system described in the next section.

2.3 Task Execution and Recovery

In this section, we describe our executive level, which consists of two packages seen in Figure 2.2: the *task executor* and the *task monitor*[9]. The task executor (Section 2.3.1) contains scaffolding to specify and incrementally develop a main task recipe. The task monitor (Section 2.3.2) contains the utilities necessary recover from general failures during task execution.

[6] Available at https://github.com/GT-RAIL/fetch_simple_linear_controller
[7] http://wiki.ros.org/navigation
[8] In complex environments, `nav_stack`'s global and local planners can be used instead.
[9] Stand-alone packages under development at https://github.com/GT-RAIL/assistance_arbitration

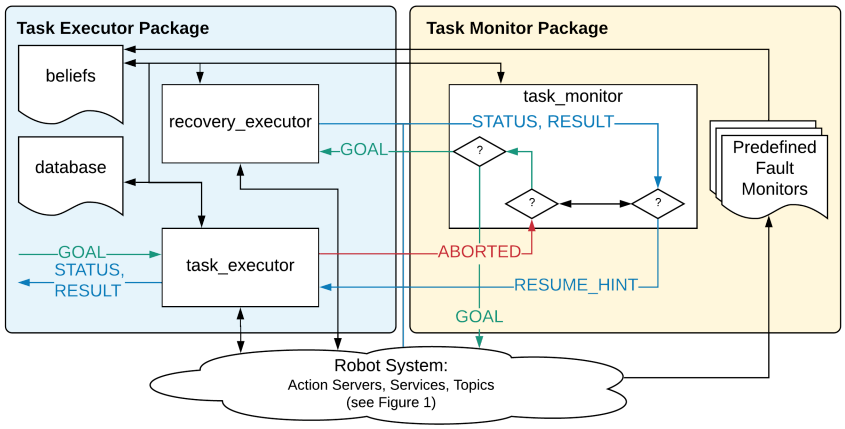


Figure 2.2: Overview of the two packages in our the executive level. Arrows denote ROS information flow, through publishers, subscribers, services, and actionlib.

2.3.1 RDD Specification: The Task Executor

During the *Specification* phase of RDD, developers translate the nominal behaviour of the robot executing a task recipe into a script. Crucially, developers should fulfill two objectives in this phase: (1) declare robot behavior under the strong assumption of perfect robustness in execution, and (2) provide structure to the specified script so that in the event of an error, it is easy to garner error context as well as resume execution once the error is resolved. The task executor package facillitates meeting such objectives.

Specifically, the task executor provides the following utilities to aid in rapid task prototyping and testing:

- A Python-based abstraction for specifying semantically meaningful interfaces to the robot's behavior layer, to form sequenceable primitive actions.
- A custom domain-specific language using YAML syntax for scripting recipe-based tasks, with a view towards facilitating hierarchical task declaration for code modularity and reuse.
- A consistent API to tasks and actions to facilitate testing in isolation and to enable easy invocation from other system components.

- A database to provide a common knowledge-base of task relevant information to all tasks and primitive behaviors.
- An automatically populated belief system encapsulating pertinent robot, environment, and task states, to provide additional context during recovery.

The following sections provide additional details on the above utilities.

Actions

Primitive actions are specified as Python objects derived from a common abstract class. They are implemented either as a client to individual robot components, such as to point-to-point navigation, or as a client to semantic groupings of robot components, such as to the grasp calculation packages.

Tasks

Tasks manifest as a Python class derived from the same abstract class as *actions*, but whose execution is specified using a custom domain-specific language, which uses YAML syntax, to allow loading and reloading of tasks from the ROS parameter server. The language allows tasks to:

- reuse other tasks for the creation of complex task hierarchies
- accept parameters for adaptation and compositionality in task specification
- create, maintain, and manipulate local variables for data transfer between actions and for adaptation to environmental or execution conditions
- utilize rudimentary control flow through conditional statements and loops, aiding in concise task specification

We present the formal task syntax in Listing 1, with example tasks used for large object pose estimation and for object picking at the FetchIt! Challenge (Section 2.4.1) shown in

Listing 1 Task Syntax

```
(task name):
  [params: [...]]
  [var: [...]]
  steps:
  - (action | task | op | choice | loop) : (name)
    [params: {...}]
    [var: [...]]

  [...]
```

Listing 2 Example Tasks

```
detect_schunk_pose_task:
  params:
  - look_location

  var:
  - chuck_approach_pose

  steps:
  - action: look
    params:
      pose: params.look_location

  - action: detect_schunk
    var:
    - chuck_approach_pose
```

```
pick_task:
  params: [object_idx, grasps, object_key]
  var: [grasped]
  steps:
  - action: pick
    params:
      object_idx: params.object_idx
      grasps: params.grasps
      object_key: params.object_key

  - action: verify_grasp
    params:
      abort_on_false: false
    var:
    - grasped
```

Listing 2. A task is defined as a dictionary entry with the required key of `steps`, which defines an ordered list of the steps in the task, and the optional keys of `params` and `var`, which define the task inputs and outputs. Each `step` in the task is named and can be one of five types: (1) `action`, invoking a primitive action, (2) `task`, invoking another task, (3) `op`, invoking a simple Python function for rudimentary data manipulation, (4) `choice`, to evaluate a boolean expression for control flow, and (5) `loop`, to loop while a boolean expression is `true`. All steps accept a dictionary setting values for their `params`, and return a list of `var` values that become local variables in the parent task[10].

Consistent API

Tasks and actions have a consistent API, which mimics that of ROS's `actionlib` interface. This consistent API enables (1) the use of JSON to specify inputs and outputs to tasks and actions easily in order to test them in isolation, and (2) the invocation of individual tasks from other ROS nodes, such as the recovery system, through the `actionlib` interface when required.

[10]For more details, see the README in our Github repository.

Database

Recipe-based tasks can be parameterized by semantically meaningful task variables which are then grounded to different values for particular environments or tasks. Example task variables are locations, robot poses, objects, or other real-world entities. The database is a YAML dictionary loaded into the ROS parameter server that provides a single source of truth for grounding all relevant task variables. Rapid environment adaptation is readily facilitated by modifying the values associated with known keys in the database definition.

Beliefs

Beliefs are key-value assertions about the robot or the state of the environment, e.g., `num_bolts_in_kit=1` or `robot_at_schunk=true`. They are included in the task executive layer for two reasons: (1) to provide context to recovery mechanisms in the event of a failure, and (2) to provide updates to a higher level planner, should one exist, about relevant states of the task, the robot, or the environment. For instance, the expected and actual state may become mismatched: transient localization and navigation errors during point-to-point navigation might compound to leave the robot at a location outside an expected tolerance for manipulation actions. Background monitors on the robot's location can indicate a mismatch, and in turn the recovery system can use this information to reposition the robot.

Discussion

The task executor package is optimized to facilitate rapid specification and testing of recipe-based tasks: developed task scripts are deterministic, easy to specify, interpretable, and readily allow the testing of components in isolation. Additionally, the scripting approach to task specification provides the implicit benefits of straightforward state tracking and an efficiency in task execution borne from overestimating the robustness of the robot's behaviors. Indeed, we do not check for most violations to the operating conditions of our

primitive behaviors until they report a failure.

We note that the determinism of our scripting approach and overestimation of the robustness of our behaviors leaves us susceptible to violations in assumptions of the environmental state (a susceptibility that reactive sequencing approaches do not share). However, instead of complicating the task scripts and in turn slowing down task specification, our RDD methodology relies on the incremental recovery development to achieve robustness and reactivity.

2.3.2 RDD Refinement: The Task Monitor

The primary objective of system development during the *Refinement* phase of RDD is to rapidly incorporate diverse recovery strategies for a specified task recipe in order to incrementally improve its robustness. As such, it involves addressing four challenges (mentioned in Section 2.1):

1. resolving ambiguity in the recovery policy for unforeseen faults
2. taking actions to reset to a known state in the event of a fault
3. deciding how to resume execution once a fault is addressed
4. trying diverse strategies when recovering from a repeated fault

The task monitor package, which provides execution monitoring and error recovery to the task executor, is designed to address each of the above challenges.

Handling Unseen Errors

In the event of an unseen error during development[11], the monitor immediately exits from the task, displaying the entire context of the error in a consistent manner and logging all

[11]The system can detect unseen errors in three ways: (1) the behavior level can propagate reported faults (i.e. action servers aborting, nodes crashing, etc.), (2) recipe steps can explicitly check for expected errors, or, in the case of unexpected errors, (3) the developer can stop system execution and write a new error detection module.

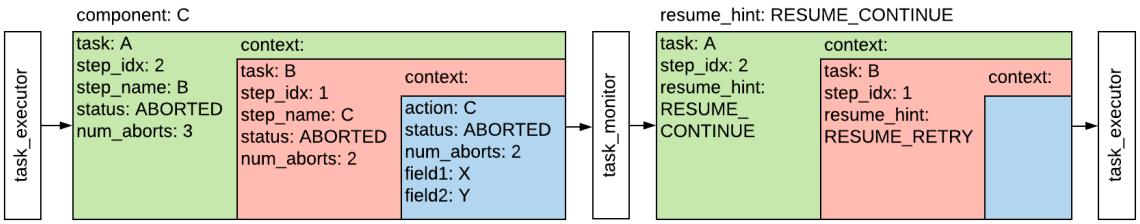


Figure 2.3: Metadata passed between the task executor and the task monitor, used to facilitate error diagnosis and task resumption after fault resolution.

possible causes. Developers can then inspect the logs to create a tailored (set of) recovery mechanism(s) for such errors, thereby making them "known" errors during future failures. In practice, we quickly accumulate a list of errors that our developed recovery strategies know to address.

During deployment, unseen errors can be dealt with under a domain-dependent context-relevant policy of always exit, always retry, or some combination thereof.

Taking Actions

Rapidly developing and testing actions to take in the event of a particular failure requires the presence of (1) a means of determining the diagnosis of an error, and (2) an easy mechanism to invoke actions or subsets of actions. The monitor and task executor are designed to facilitate both.

The structure and compositional design of our main task recipe aids in fault diagnosis, given the current task state. Specifically, in the event of an error, tasks in the task executor provide a consistent context of their state in a recursive dictionary containing all tasks in a task hierarchy until the primitive action, and primitive actions can also provide error context through custom data fields, as shown in Figure 2.3. Further, the task executor's beliefs (Section 2.3.1) can additionally inform error recovery.

When an error is diagnosed, the consistent API for invoking tasks and actions facilitates the monitor in resolving the problem. The monitor uses a redundant instance of the task executor, called the recovery executor (seen in Figure 2.2), to execute simple task recipes

for recovery.

Resuming

We have identified five strategies for resuming task execution that have applied to the errors we have encountered:

1. `RESUME_NONE`: stop executing the task.
2. `RESUME_CONTINUE`: resume task execution from the failed step.
3. `RESUME_RETRY`: restart a subtask, or the whole task; useful if, for example, the environment changed during recovery and thus perception must be rerun.
4. `RESUME_NEXT`: resume execution at the next step; useful if the recovery process accomplishes the failed step.
5. `RESUME_PREVIOUS`: resume execution at the previous step; useful if failure assumptions change, but the entire subtask does not need to be restarted.

To enable full flexibility in the recovery mechanism on how tasks[12] are resumed, any of the tasks in a hierarchy can be resumed using any of the above five strategies. An example of the context for task resumption is shown in Figure 2.3.

Recovery Diversity

Due to the larger context of some errors, the same recovery actions taken during the same fault diagnosis can fail: for example, recalculating grasps on a small object when sampling-based arm motion planning fails to pick it up may be insufficient due to an arm workspace limitation, and instead the error should be resolved by repositioning the robot base or by moving the arm to a different start configuration. As such, the context dictionaries included

[12]Resuming execution from arbitrary stopping points in primitive actions is hard [45], but depending on the implementation of the robot system, might be unnecessary.

for diagnosis and resumption support development of diverse recovery strategies for the same faults, based on factors such as task hierarchy location, primitive action failure count, or hierarchical task failure counts.

Discussion

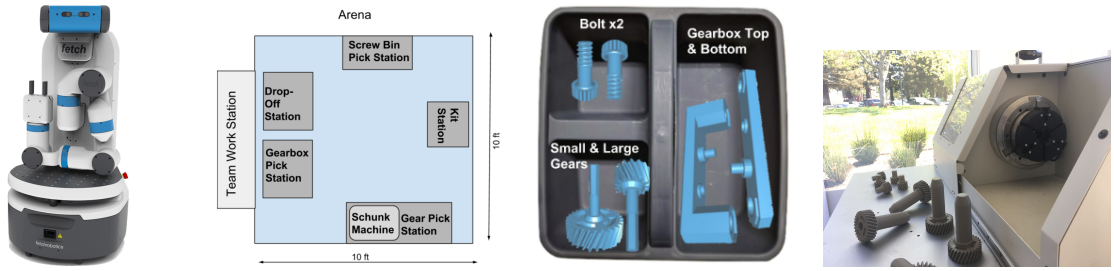
The philosophy behind RDD's *Refinement* phase, i.e. incremental and independent recovery development, necessitated a recovery system that is deterministic, easy to specify, interpretable, and readily allows testing of individual recoveries in isolation. Although the current version of our implemented recovery system is not robust to failures during the recovery process, such robustness can either be added in a future iteration of our system, or can be left to the purview of a higher-level planner in the robot system. Finally, we note that our current rule-based system of recoveries does not easily lend itself to analysis or verification, but such a feature can be integrated in the near future. In the meanwhile, the easy testability of individual recoveries mitigates the lack of verification in the system.

2.4 Validation

Our task execution approach formed our executive level for the FetchIt! Challenge at ICRA 2019. The challenge's goal was to advance autonomy and robustness by using a mobile manipulator to perform an industrial kit assembly task in an unstructured environment. As such, the challenge was a good opportunity to validate our system and development methodology in a real-world, time-sensitive scenario. We provide a brief description of the FetchIt! Challenge, followed by quantitative and qualitative observations of the RDD workflow and the recovery mechanisms we developed for our competition-winning robust task executor.

As further context, and to provide a continuous example of recoveries over a 45-minute autonomous task, we provide a video of our final competition run[13].

[13] https://youtu.be/G_ur71h4CNQ



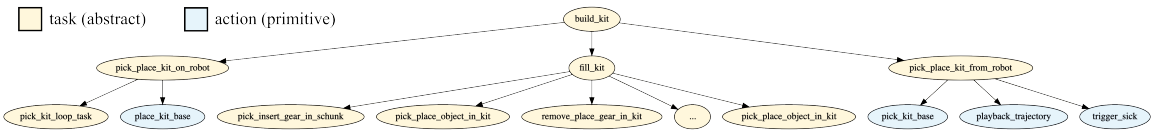
(a) Fetch Robot (b) Challenge Arena (c) Assembled Kit (d) Schunk Machine

Figure 2.4: FetchIt! challenge hardware and specifications.

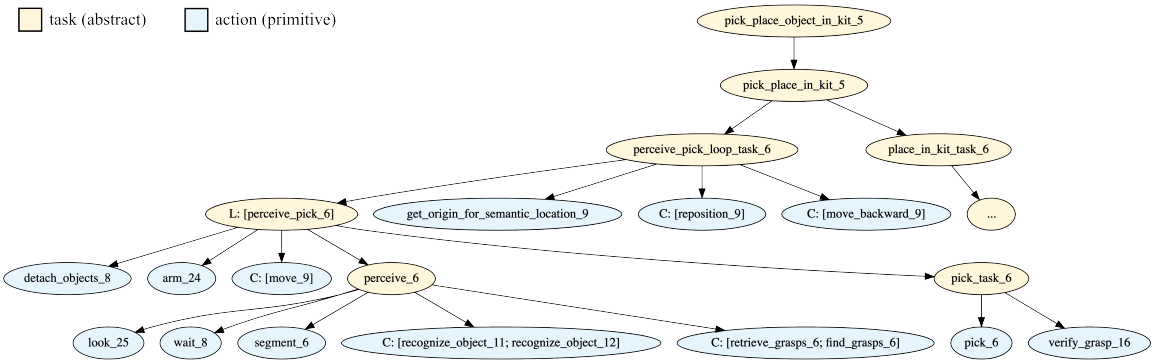
2.4.1 FetchIt! Challenge Overview

The FetchIt! Challenge was a mobile-manipulation challenge focused on autonomously completing combined manipulation, perception, and navigation tasks on a mobile-manipulator platform[14]. Specifically, the goal of the competition was to have a Fetch mobile manipulator [63], equipped with an RGBD camera and a 2D LIDAR (Figure 2.4a), autonomously assemble kits (Figure 2.4c). In order to assemble each kit, the Fetch had to navigate a challenge arena (Figure 2.4b), perceiving and picking the various parts from tabletops and bins. In addition to pick and place, the Fetch had to operate machinery in the arena via physical manipulation and wireless interfaces as part of the assembly process. For instance, it had to insert the "Large Gear" in Figure 2.4c into a small opening shown in Figure 2.4d on a simulated milling machine. Perfectly completed kits (as in Figure 2.4c) scored seven points, with no points awarded for incomplete kits (i.e. any parts missing).

The robot was required to run autonomously with no intervention for an allotted time of 45 minutes, completing as many kit assemblies as possible. The strict scoring requiring fully assembled kits made robust task execution one of the largest challenges of the competition.



(a) The top three levels in the hierarchical task tree. Repetitions of $pick\_place\_object\_in\_kit$ are omitted and denoted with an ellipsis for brevity.



(b) Full expansion of the fifth $pick\_place\_object\_in\_kit$ task. The suffix n for each node indicates the n^{th} invocation of an action or task over the full task tree (n is often higher in practice due to recovery execution). Details of $place\_in\_kit\_task$ are omitted for brevity. C denotes a choice node that denotes conditional execution, and L denotes a loop node.

Figure 2.5: Hierarchical task tree for the FetchIt! challenge.

2.4.2 Validation of Recovery-Driven Development

Figure 2.5a shows the high-level structure of the task implemented for the FetchIt! Challenge—an easy to understand script. Figure 2.5b demonstrates the task complexity, showing an expansion of one of the abstract tasks from Figure 2.5a. This complexity is manageable thanks to hierarchy-enabled action and task reuse–our task script reuses the look action at least 25 times and the perceive task at least 6 times. The modular nature of the task specification simplified the design process and allowed for independent testing of task recipes.

Further, the RDD requirement of separating specification and refinement allowed us to safely develop new recovery behaviors in high-pressure moments between competition runs. For instance, the simulated milling machine required gear insertion into a smaller hole than we had previously tested with, which caused new faults resulting from false positives in the robot's evaluation of the insertion. With only 45 minutes to test between runs, we

[14]https://opensource.fetchrobotics.com/assets/Rulebook2019.pdf

were able to quickly update existing recovery strategies to identify insertion failure and retry the task, without risk of disrupting the previously tested nominal task flow.

2.4.3 Evaluation of Task Robustness

The FetchIt! Challenge provided an opportunity to gather quantitative data on our recovery strategies, allowing us to evaluate the error recovery utilities provided by the task monitor (Section Section 2.3.2). We also provide a representative example to qualitatively highlight those utilities.

Quantitative Observations

We provide a breakdown of the recovery strategies we implemented for the FetchIt! Challenge. In total, we implemented 18 strategies, which often included multiple sub-strategies to handle dynamic execution under differing fault conditions.

To highlight the value of easy rule specification for recoveries and the ability to act upon a rule-based diagnosis, we define the following three situations:

1. *Shared Recovery*: different faults use the same rules for diagnosis and recovery
2. *Immediate Action*: recovery directly invokes a primitive action
3. *Dynamic Recovery*: in the same error diagnosis, error context determines different parameters for recovery actions

Figure 2.6a shows the occurrence of these three situations in our developed recoveries. We most frequently use *Immediate Actions*, to create short and responsive recovery actions to bring the task back to a known state. While less frequent, *Shared Recoveries* aided in rapid development and *Dynamic Recoveries* were crucial in creating a reactive system to deal with diverse faults.

The task executor and monitor use the following factors in the metadata context to determine what recovery to perform:

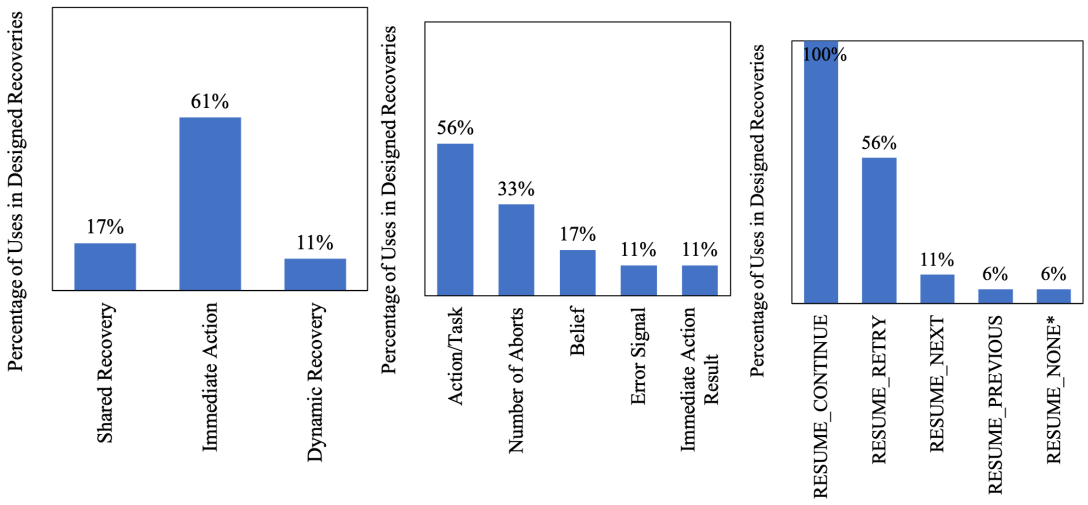
1. *Action/Task*: the location of the error in the task hierarchy
2. *Number of Aborts*: how many times the error has occurred without resolution
3. *Belief*: a subset of the robot's belief of the task, robot, or environment state
4. *Error Signal*: a specific error signal returned by a primitive action
5. *Immediate Action Result*: the result of actions taken for recovery execution

Figure 2.6b illustrates that localizing the error within the task hierarchy was especially important in determining recoveries because the task and action reuse for different situations often required different recoveries. Overall, the use of a diverse factors shows that robust recovery requires a wide range of context.

Finally, Figure 2.6c demonstrates that all resumption strategies described in Section 2.3.2 were necessary for designing a robust recovery system. The diversity in resumption strategies showcase the possibility of resuming from a task beyond simply re-attempting it entirely, and the use of resumption strategies other than `RESUME_CONTINUE` show that resumption cannot always directly return to where the error occurred.

Representative Example

We describe here recoveries for the $pick$ action (Figure 2.5b) to provide concrete examples for the features mentioned above. In our system, $pick$ and arm can fail due to a common cause—errors in the MoveIt! Motion Planning Framework. Therefore, the default recoveries for these actions, e.g. reinitializing a 3D obstacle map followed by a `RESUME_CONTINUE`, are examples of *Shared Recoveries*. However, depending on the context, the fault sometimes requires additional recovery steps. For instance, after the third consecutive failure of both actions in the $pick\_task$, a short upward arm move jogs the system out of its error. When this is not enough, the cause of failures can be a limitation in the arm's workspace, and so the robot repositions itself based on beliefs about the task and



(a) Properties of the recovery (b) Factors used in diagnosis (c) Resumption strategies

Figure 2.6: Percentage of times that recovery uses each utility of the task monitor, for the 18 main recovery strategies designed for FetchIt! Challenge. In (c), note that `RESUME_NONE` is also the default strategy for unseen errors.

environment state. All faults that occur within the context of the $perceive\_pick$ task retry that task (`RESUME_RETRY`) in order to account for scene changes resulting from recovery execution. We show a selection of these $pick$ recoveries in action, as well as other example recovery behaviors selected from our FetchIt! competition runs, in a video supplement to this chapter[15]. At the competition, we recovered from all errors in the $pick$ task, thanks to the task monitor's utilities.

2.5 Discussion

We conclude with a discussion of lessons learned using our RDD-inspired task execution and monitoring system at the FetchIt! Challenge.

Testing early and often. The ability to repeatedly test the robot system in its target environment is a critical requirement for the robustness benefits of RDD—while RDD can be implemented in simulation, simulation alone will likely not lead to the same level of robustness. As such, RDD is not suited to hazardous or remote environments, such as

[15]https://youtu.be/AcOdT10q_94

space robotics. However, many target environments for robots are neither inaccessible nor catastrophically hazardous, and are therefore compatible with RDD.

Stochasticity in behaviors. As mentioned in Section 2.3.2, failure recovery requires a high degree of variety in recovery mechanisms. We have found that a degree of non-determinism at the robot's behavior level facilitates such recoveries. For instance, our sampling-based ranking approaches to object selection, grasp calculation, and place pose calculation provided the robot with successful alternatives when retrying actions after previous action attempts failed.

Planning layer integration. Recipe-based tasks can admit multiple recipes, which need to be selected or rescheduled at runtime based on factors such as time constraints or major execution errors. Predefined scripts, such as those created during RDD specification phases, cannot easily handle such situations. The shortcoming can, however, be addressed by the higher level planning layer in the robot architecture. We found that the level of abstraction used in our hierarchical executive layer recipe scripts made the specification of our planning layer almost trivial. Additionally, the executor and monitor utilities we developed (e.g. beliefs) were a great help in the planning layer.

In conclusion, the RDD methodology of separating the nominal task specification from recovery specification provides numerous benefits, which our team validated at the FetchIt! Challenge. Our use of RDD (1) allowed **rapid development of the task and recoveries**, (2) enabled **independent testing and efficient re-use through abstraction** for tasks and recoveries, (3) necessitated the **development of system utilities** that ultimately proved valuable in other aspects of system development, and most importantly, (4) afforded our system a **level of robustness** that would have been more difficult or time-consuming to achieve through other means.

The reduced frequency of failures, and hence increased reliability, of a robot system achieved through RDD involves transferring the experiences of past interventions into a policy of actions for future failures, albeit through developer-implemented system updates.

As such, it implies the possibility of accomplishing such a transfer without developer effort, and it is therefore a question that we intend the revisit in Chapter 6. Before addressing it, we examine how modules developed for the benefit of collocated and remote human operators might also reduce the *duration* of robot failures.

CHAPTER 3
THE INTERRUPTIBILITY OF COLLOCATED HUMANS

We begin this chapter by noting that unless a robot is actively monitored by a human operator, it must find and *interrupt* a human in order to notify them of an error and to solicit help. Interruptions are distracting, potentially leading to task performance penalties [64, 65], stress [65, 66], antipathy [27], and even catastrophe [67, 68], depending on context. As such, an inappropriate or incorrect interruption from a robot can greatly increase the *duration* of robot failures if the interrupted humans ignores the interruption, takes more time to intervene, or intervenes incorrectly.

In the context of technology-driven interruptions, a large body of work in human factors engineering (HFE) and human-computer interaction (HCI) research has specifically identified the appropriateness of the *timing of an interruption* as one of the most important factors dictating interruption consequences [69, 64, 65, 68]. The appropriateness of timing is referred to as *interruptibility* [70] and it is itself the focus of much research [71]. Low interruptibility signifies a person's desire to not be disturbed, while high interruptibility signifies that the person could be amenable to an interruption.

Today's robots have no interruptibility awareness, despite the fact that interactive robots are increasingly deployed in human environments. Many robot control architectures being developed in the research community for interactive applications enable robots to not only follow human instructions, but also to actively engage with a person to offer a service [19] or to ask for help [72, 73]. As a result, robots performing deliveries, taking store inventory, organizing warehouses, and collaboratively working alongside humans on factory production lines increasingly have the potential to interrupt people, without any measure of the appropriateness or costs of such interruptions. Extrapolating results from prior research [74, 75] to the domain of embodied robot interactions, suggests that inappropriate interruptions

may have significant effects on many factors, including

- negatively impacting human task performance, if people are interrupted at inappropriate times,
- negatively impacting robot task performance, as the robot wastes time attempting to interact with a person not receptive to the interaction, and
- negatively impacting a person's social perception of the robot, and ultimately their willingness to use it.

In order to develop robots that appropriately handle interruptions, it is important to determine *when* a robot should interrupt, and *how* it should behave during an interruption. Prior work has explored how a robot should behave during interruptions by studying multiple approaches for engaging people [76, 77]. In this chapter, we address the former question. Expanding on results previously presented in [41] and [42], we describe a self-contained interruptibility-aware mobile robot system and present a detailed analysis of the effects of interruptibility-aware behavior on the factors listed above.

We begin by examining the following research questions:

RQ1 Which computational features are useful in allowing a robot to classify interruptibility in an unstructured world?

RQ2 What is a robust model for obtaining interruptibility estimates from the proposed features?

In our examination, we first contribute an ordinal scale of interruptibility that can be used to rate the interruptibility of a person and to influence decisions on whether or not to interrupt them (Section 3.2). Second, derived from factors used by humans to gauge interruptibility [68], we propose using features for *person state*—motivated by prior work in robotics on the closely related problem of estimating human engagement [78]—and features for *interruption context*—inspired by cues to interruptibility context used in prior work [79]—to

classify interruptibility (Section 3.3). Last, we introduce the non-temporal and temporal models that we evaluated (Section 3.4) and the dataset of person observations that the models were evaluated upon (Section 3.5).

Our results (Section 3.6) show that (1) features for *person state* and *interruption context* are indeed useful for classifying interruptibility, (2) in the absence of robust detectors for *person state*, more robust detectors for *interruption context* are a good substitute, and (3) Random Forest (RF) [80], Multi-Layer Perceptron (MLP) [81], and Latent-Dynamic Conditional Random Field (LDCRF) [82] classifiers outperform all other classifiers in interruptibility classification, remaining robust to feature noise. The results inform our development of an interruptibility-aware robot system (Section 3.8).

We evaluate our robot system in a 42 participant user study, introduced in Section 3.7 and described in Section 3.9, to answer the following research questions:

RQ3 Can we use measures of the robot's behavior to show that our models accurately estimate interruptibility online on a robot platform?

RQ4 How does interruptibility-aware robot behavior affect human task performance when a robot regularly needs assistance?

RQ5 How does interruptibility-aware robot behavior affect robot task performance when relying on humans for assistance?

RQ6 Does a robot appear more socially adept if it interrupts humans at appropriate moments?

Our results (Section 3.10) show that (1) our integrated system is effective at predicting interruptibility at high accuracy, (2) an interruptibility-aware robot interrupts less often but at more appropriate times thereby increasing its efficiency, (3) better timed interruptions have no significant effect on human task throughput in skill-based tasks (perhaps as a result of participants self-regulating their schedule by ignoring badly timed interruptions), and

(4) users have a higher opinion of the interruptibility-aware robot. These results highlight key findings for face-to-face robot interruptions, underscore the social and task benefits of interruptibility-aware robot behaviours, and present directions for future research.

3.1 Related Work

People are generally very adept at gauging the interruptibility of others from observation: when deciding the moment to interrupt, they naturally take into account another person's projected level of "busyness" (demeanor) and availability, the context and conditions of the interruption, and their knowledge of the consequences of the interruption [68]. In the following subsections, we highlight some of the pertinent prior works that detail computational methods for estimating a person's availability and the context for an interruption, as well as prior research into evaluating the consequences of interruptions.

3.1.1 Estimating Availability and Interruption Context

Existing work has explicitly modeled a person's availability in one of two ways. The first category of techniques relies on *task* and *experiential* knowledge. In HCI, known task models, for instance detailed as GOMS (Goals, Operators, Methods, Selectors) structures [83], have been used to estimate interruptibility [84, 85]. Meanwhile in robotics, cognitive architectures such as ACT-R/E [86, 87] have been used to predict if humans might need assistance in resuming a task post-interruption by another human, a technique that easily extends to determining the moment to interrupt. However, these approaches require domain knowledge of a human's task and constant surveillance of its execution, which is often unavailable in a general-purpose mobile robot deployment. Others have modeled human availability based on past experiences of room occupancy, assuming that an open office door indicates the occupant's willingness to be interrupted [73], but this assumption ignores both the social cues and task state to greatly simplify the interruptibility problem.

The second category of techniques for explicitly estimating availability leverages a per-

son's demeanour, focusing on immediate *social* cues of availability. Social cues, such as eye contact, are largely task-independent, and as a result, models based on social cues are more easily generalizable across a wider set of applications: in robotics, the methods have been used to estimate related measures of a person's "intent-to-engage" and awareness of the robot in applications ranging from companion robots [78, 77], shopping mall assistants [88, 89, 90, 19], receptionists [18], and bartenders [91]. Some prior work has relied on external sensors such as motion capture systems, ground-mounted LIDAR, and ceiling cameras [88, 89, 90, 19], which can be expensive and difficult to deploy in support of mobile robots traversing a large space. Other work has used onboard sensors to detect social cues of engagement [18, 77, 78, 91]. Although engagement estimation is a separate problem from interruptibility estimation (because interruptibility can be high even when engagement is low), the problems are closely related, and we take inspiration from the work of Mollaret et al. [78] and Chiang et al. [77] in both our selection of audio-visual features for classification and in validating the use of Hidden Markov Models to estimate interruptibility.

Existing work has also implicitly modeled a person's availability through methods termed "contingency detection" [92, 93, 94]. These methods often assume the person as available, perform a probe action (or sequence of actions), and then reassess the person's availability based on the person's response. Assessing a person's availability through contingency detection is complementary to estimating availability explicitly: the latter can inform the execution of probe actions for the former.

Meanwhile, interruption context has been extensively studied in HCI [71], where context is often captured through features that describe the user (e.g., personality traits) [70, 95], the task [84, 85], the environment [96, 95], the interruption [97], and the relationships between these when the interruption is presented [65]. In robotics, interruption context has been studied by Nigam & Riek [79], where the authors use only global audio-visual descriptors—such as GIST [98] features and audio frequency & volume features—as cues

to context in classifying interruptibility (termed an *appropriateness function*) on their collected dataset. In our work, we instead leverage advances in computer vision to garner localized, explicit, high-level environment context from the labels of objects that a person might be interacting with. We also deploy our best model for online classification in a user study for further evaluation.

3.1.2 Evaluating Interruption Consequences

In HCI and HFE, the cost of an interruption on-screen has been evaluated with quantitative metrics such as time on task [74, 84, 66, 99], the number of tasks completed [74], the number of incomplete tasks [100], the number of errors [74, 101], switching time [85, 75, 99], and workload [84, 66]; and qualitative metrics such as respect [84], and preference [74]. In embodied settings, researchers have also used structured interviews [88, 102, 103] and ethnographies [27, 102, 103, 104] to evaluate long term interruption costs.

However, the evaluation of face-to-face robot interruptions, in which a robot is co-present with the human, has often been limited to qualitative measures to gauge the effectiveness of the interruption. For instance, Saulnier et al. [76] base their evaluations on participant self-assessed "interruptedness", while Chiang et al. [77] evaluate whether an interruption successfully captured the attention of a participant, without consideration for the appropriateness of interruption timing. While the recent work of Short et al. [92], does quantitatively evaluate the effectiveness of robot interruptions through a measure of the number of survey responses started by interrupted humans, there is no prior work that quantitatively studies the *task effects* of embodied robot interruptions on both the human's and robot's performance.

Prior research shows a strong effect of interruption handling mechanisms on the potential costs of interruptions [102]. In HCI settings, research has shown that people subject to on-screen mandatory interruptions experience significant loss in task performance [74, 84, 75]. However, when such interruptions can be deferred by the participant, as in [74], or

when they do not consist of an actual task, as in [75], the loss in task performance is not as significant; a result predicted by the Goal-Activation model of interruption handling [105]. Similarly, in embodied settings, when people defer an incoming interruption, they are more likely to complete their original task [100]; a result predicted by Prospective Memory [106] models of interruption handling [102]. Recent results from HFE continue to show that performance loss is not noticeable with tasks that are embodied or skill-based, even when the interruptions might be computer mediated as in the work of Lee & Duffy [101] and Kolbeinsson et al. [99]. These authors, in particular, reason that performance loss is absent in an embodied setting because it is impossible to occlude the main task, which allows people to optimize common sub tasks and choose when to switch to an interruption. In this work, we explore whether the results from HFE research generalize to robotic systems.

3.2 Interruptibility Classification

Interruptions are defined as "externally generated, randomly occurring, discrete events that break the continuity of cognitive focus on a certain task" [64], and the *interruptibility* of a person at any given point in time is defined in terms of their receptiveness to interruptions at that moment [70]. A person focused on their current task and not amenable to an interruption is said to have low interruptibility; meanwhile, a person amenable to interruptions is said to have high interruptibility. Hence, the interruptibility classification of any given person-of-interest can be a binary classification task, with 0 denoting the person as busy and 1 denoting them as interruptible.

Binary interruptibility classification provides an intuitive mechanism for deciding when to interrupt a person, but it is important to distinguish interruptibility from the decision to interrupt. The interruptibility of a person quantifies the *disturbance* that a person might experience as a result of an interruption, while the decision to interrupt depends upon a person's interruptibility as well as other factors, such as the urgency and characteristics of the interrupting task [68]. In this work, we focus on the classification of interruptibility and

its use on a robot, with the goal of incorporating the classification later within a broader framework for deciding when to interrupt.

In some applications, it can be useful extend the binary interruptibility classes to a higher fidelity in order to further help with the robot's decision-making process. Such situations may arise when the robot needs assistance from one person when multiple people, potentially in different states of interruptibility, are present, or if the robot should behave differently depending on the person's level of interruptibility. To support these capabilities, we propose the following interruptibility scale:

INT-4 **Highly Interruptible**. The person is not busy and they are aware of the robot's presence.

INT-3 **Interruptible**. The person is not busy, but they are unaware of the robot's presence.

INT-2 **Not Interruptible**. The person is busy, but the robot may interrupt if necessary.

INT-1 **Highly Not Interruptible**. The person is very busy, the robot should not interrupt.

INT-0 **Interruptibility Unknown**. The robot is aware that a person is present, but does not have sufficient sensory input to analyze interruptibility.

Values 1-4 in the scale capture the full range of interruptibility states that can help guide the robot's decision making process. We include the rating of 0 to represent states in which the robot does not yet have sufficient information about the person, such as when the person is too far away or out of view. In this case the robot may choose to approach another person, or take actions to improve sensing quality.

3.3 Perceiving Interruptibility

Interruptibility can be characterized based on two sources of information—*person state* and *interruption context* (Figure 3.1).

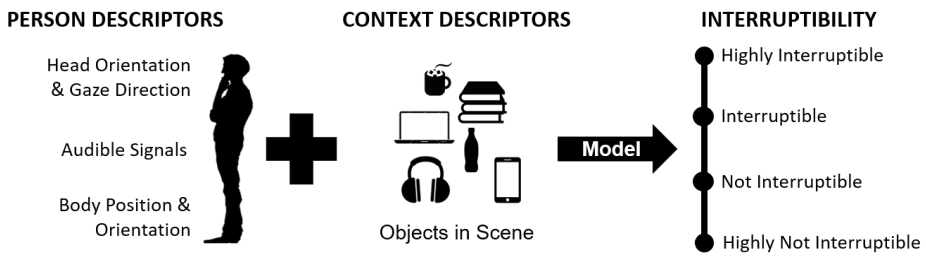


Figure 3.1: The level of interruptibility of a person is represented on a four point scale. In order to arrive at a value on this scale, we use information about person state and interruption context. In this work, we use object labels as a cue to the context.

Person state is widely used to model engagement and human awareness in robotics [78, 77]. Although classifying interruptibility poses its own research problem, because interruptibility can be high even when a person shows neither intent-to-engage or awareness of the robot, we propose the cues of person state from the engagement modeling literature can be informative for interruptibility. Following prior work, person state includes the following information categories:

- The **position and orientation** of a person within the environment. This includes where they are located as well as how their body is oriented with respect to the robot.
- The **head orientation and gaze direction** of the person.
- The **presence and orientation of sound** within the environment.

We infer person state from laser, video, and audio sensor data.

The context of an interruption includes known information about the user, the task, the environment, and the type of interruption [71]. Following the definition in prior work [79], we consider interruption context to include visually observable cues from the environment that may inform the robot of a human's interruptibility. In particular, we use:

- The **labels of objects** that are being used by the person, or those that lie near them.

We infer the object labels from robot camera video and propose that the object cues to a person's activity can provide additional useful information for classifying interruptibility.

For example, an individual drinking from a coffee mug in a lounge is judged to be more interruptible than someone engaged with a laptop in the same setting. Although objects may not be a valid substitute for person state, or even activity recognition for interruptibility estimation, object recognition is widely available on robotic systems.

In the following section, we describe how this information can be leveraged in multiple computational models.

3.4 Models for Interruptibility Classification

Based on our survey of prior literature, we consider both non-temporal and temporal models for interruptibility classification given data inputs of the form in Section 3.3. Non-temporal models provide interruptibility estimates based on data from a particular moment. Informed by the survey of Turner et al. [71], which details the various classification models commonly used for interruptibility classification in HCI, we explore the use of Random Forests (RFs) [80], Support Vector Machines (SVMs) [107], K-Nearest Neighbors (KNN) [108], and Multi-Layer Perceptrons (MLPs) [81].

In contrast, temporal models, use a sequence of data within a time window to generate the interruptibility estimates. Recent work by Foster et al. [91] on engagement modeling has shown that although non-temporal models are more accurate at a classification task, temporal models tend to work better on a robot due to greater stability in classification output. Informed by the successful use of Hidden Markov Models (HMMs) for engagement detection on robots in the works of Mollaret et al. [78] and Chiang et al. [77], we use HMMs as one of our temporal models. We also explore Conditional Random Fields (CRFs) [109] and derivatives thereof, Hidden Conditional Random Fields (HCRFs) [110] and Latent-Dynamic Conditional Random Fields (LDCRFs) [82], as alternate temporal models to classify interruptibility. We expect the CRF variants to outperform HMMs for interruptibility classification because of their discriminative nature and more expressive representation.

In this section, we introduce and overview the non-temporal and temporal models that we evaluated to classify interruptibility.

3.4.1 Non-Temporal Models

Here, we provide a brief overview of each non-temporal model, the hyperparameteres we used, and the reasons to expect success with each model. Each of the models is implemented with the scikit-learn framework [111].

Random Forests

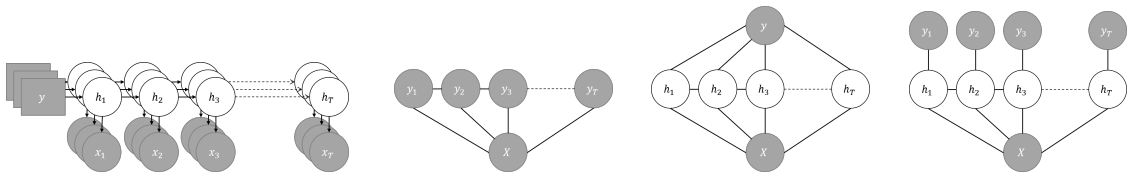
RFs have been shown to be powerful models for activity recognition [112]. An RF [80] models data by creating several decision trees, and allowing each of them to ”vote” on test cases. Each decision tree is trained on a subset of the training data, equal to the original dataset size and drawn randomly with replacement. We varied the number of trees in our RF and ultimately found that 10 estimators provided the most accurate and generalizable model.

Support Vector Machines

SVMs have also been successful in many applications of supervised learning and classification, including activity recognition [113] and gaze estimation [114]. An SVM [107] employs hyperplanes to attempt to partition training data into separate classes, after casting it to a higher dimension using a kernel function. We experimented with different kernel functions and multi-label strategies to determine that the radial basis function kernel and the one-vs-one classification strategy worked the best for our data.

K-Nearest Neighbors

KNN [108] is a model that makes use of similarity in data points to classify unseen data. The most important parameter for KNN is the number of neighbors to examine, which we



(a) HMM (b) CRF (c) HCRF (d) LDCRF

Figure 3.2: Graphical representation of each of the temporal models in this chapter. Gray elements represent observed variables, and white elements represent hidden variables.

set to 5 after a short experimental search. Classification of unseen data is performed by examining the 5 closest data points in our training data, and returning the class label with the most votes.

Multi-layer Perceptron

An MLP [81] is model that trains iteratively on each example in the dataset, using partial derivatives from a predefined loss function to update weight parameters that are used for each prediction. While there are many parameters and architecture choices to make for an MLP, we used a log-loss function, a ReLU [115] activation function, and the Adam [116] optimizer. Our architecture is a 2-layer network with 100 units in each layer, and our learning rate is set to .001. We allow training to continue until the loss stops decreasing by more than .0001.

3.4.2 Temporal Models

Here, we provide an overview of each temporal model and provide motivations for its use in our research. We explain temporal models in greater detail than non-temporal models because of their relative rarity in robot research.

Hidden Markov Models

An HMM [117] models two stochastic processes. The first process is a Markov chain through a sequence of discrete hidden states, while the second process produces observable

continuous or discrete emissions given a hidden state (Figure 3.2a). HMMs have found widespread use in areas such as natural language processing and speech recognition, and in the context of human-robot interaction have been used for tasks such as activity recognition and human engagement detection [77, 78].

The HMM is characterized through five parameters

$$\lambda = (N, M, A, B, \pi)$$

where each of the parameters has the following significance:

N is the number of hidden states in the model. Although it is common for the hidden states to have some physical significance, this need not be the case.

M is the number of distinct observation symbols per state if the observation sequence is discrete valued. In the case of continuous observation sequences, M denotes the number of mixture components that contribute to producing an observed value.

A is an $N \times N$ state transition matrix where each element of the matrix signifies the probability of transitioning from one hidden state to another.

B is the observation symbol probability distribution for all hidden states. In the case of discrete emissions, B is an $N \times M$ matrix; in the case of continuous emissions, B is a parameterized specification of M mixtures (usually Gaussian) for each of the N hidden states.

π is the initial state distribution over the hidden states.

To classify interruptibility, we train separate ensembles of HMMs for each of the five different interruptibility classes that we have defined. Within each ensemble, we train a separate HMM for each of the features in the data sequences that we use. We use a uniform initial distribution over all hidden states, and we model the continuous valued features

using Gaussian Mixture Models. The HMMs are implemented with the GHMM library[1] and trained using Baum-Welch. We vary the number of hidden states, N, from 2–4 and the number of mixtures in the models, M, from 1–4.

Given a sequence of data, each of the trained HMMs in each ensemble runs the Forward algorithm to return a log likelihood of the data being generated by the HMM; the log likelihood result for the ensemble is taken to be the sum of log likelihoods from each HMM within the ensemble. The interruptibility label derived for the given data sequence is then determined on the basis of the maximum log likelihood from each of the different ensembles.

Conditional Random Fields

Represented as an undirected graphical model, a CRF [109] models the probability of a label sequence conditioned on the entire observation sequence (Figure 3.2b), as opposed to an HMM which models the joint probability of both the hidden state and the observation at any timestep. The change allows the CRF a richer specification, using prior domain knowledge, of the relevant factors within the model by incorporating information over multiple timesteps within the observation sequence and linking state transitions within the model directly to the observations. Previous work has successfully demonstrated the superiority of CRFs over HMMs in the realms of Activity Recognition [118] and Natural Language Processing [109], leading us to hypothesize that CRFs hold promise for gauging interruptibility.

Concretely, the CRF model provides

$$P(Y|X) = \frac{1}{Z}\prod_{t=1}^{T}\Psi_t(Y_a, X) \qquad Z = \sum_{Y_a}\prod_{t=1}^{T}\Psi_t(Y_a, X)$$

where $Y = \{y_1, y_2, ..., y_T\}$, each $y_i \in \mathcal{Y}$, is the label sequence, $\mathcal{Y}$ is the set of possible

[1] http://ghmm.sourceforge.net/index.html

labels, X is the observation sequence, Z is a normalization function, and T is the length of the observation sequence. Y_a is a subset of the label sequence considered for Ψ_t, a local feature function dependent on time that contains the parameters to be trained for the CRF. In our work, $\mathcal{Y} = \{0, 1, 2, 3, 4\}$, the set of possible interruptibility labels, and we use two types of feature functions—*windowed* observation feature functions and *edge* observation feature functions.

Windowed observation feature functions include a window parameter, ω, that defines the number of past and future observations to use when predicting a label at time t. These feature functions are of the form:

$$\Psi_t(Y_a, X) = exp\{\sum_{k=1}^{K} \theta_k f_k(y_t, x_{t-\omega}, x_{t-\omega+1}, ..., x_{t+\omega}\} \tag{3.1}$$

where y_t is the label at time t, x_i is an observation value at time $t = i$, and K is the number of feature functions, f_k; in our case K is the same as the number of attributes in the data. The parameter θ_k is a parameter that is trained using gradient descent.

Unlike windowed observation feature functions, edge observation feature functions model transitions from one interruptibility class to another. These feature functions have the form:

$$\Psi_t(Y_a, X) = exp\{\sum_{k=1}^{K} \theta_k f_k(y_{t-1}, y_t)\} \tag{3.2}$$

where all the variables have the same meaning as they did in Equation 3.1 and the value of K is the number of possible transitions, 25, from one interruptibility class to another.

In our work, the feature functions are specified using the implementation of CRFs in the HCRF library[2], and we train the parameters θ_k using the BFGS gradient descent method. Unlike with the HMMs, we do not train separate CRFs for each of the interruptibility classes; instead we train the CRF to perform multiclass classification. We vary the value of the hyperparameter ω from 0–4.

[2]https://sourceforge.net/projects/hcrf/

Hidden Conditional Random Fields

The HCRF [110] extends the CRF by including hidden state variables to more accurately model intra-class variation within observation data. In addition, the HCRF provides a single label for the entire sequence (Figure 3.2c) and thus prevents the need for an a-priori segmentation of the observed sequence into substructures. Prior work has successfully used the HCRF for Gesture Recognition [110], and thus we consider it a good candidate for modeling interruptibility.

Mathematically, the HCRF is formulated in a similar manner to the CRF:

$$P(y|X) = \sum_{H} P(y, H|X) = \frac{1}{Z} \sum_{H} \prod_{t=1}^{T} \Psi_t(y, H, X) \qquad Z = \sum_{y} \sum_{H} \prod_{t=1}^{T} \Psi_t(y, H, X)$$

where $H = \{h_1, h_2, ..., h_T\}$ each $h_i \in \mathcal{H}$, is a sequence of hidden states that capture the underlying structure of class y, and $\mathcal{H}$ is the set of possible hidden states. Correspondingly, $|\mathcal{H}|$ is the number of hidden states that the HCRF can use; this hyperparameter is optimized during training.

In our work, the feature functions in Equation 3.1 and Equation 3.2 are modified so that y_t and y_{t-1} are replaced with h_t and h_{t-1}, where h_t and h_{t-1} are the hidden states at time t and $t-1$ respectively. We also create an additional feature function to model the association of a hidden state to the interruptibility class label for a sequence. This feature function is of the form:

$$\Psi_t(y, H, X) = exp\{\sum_{k=1}^{K} \theta_k f_k(y, h_t)\} \tag{3.3}$$

where all the variables have the same meaning as they did in Equation 3.1. The value of K equals $|\mathcal{H}| \times |\mathcal{Y}|$, which is the number of hidden states per interruptibility class.

The feature functions are implemented using the HCRF library [2] and training is performed using BFGS. As with the CRF, we train the HCRF to perform multiclass classifica-

tion and vary the value of the hyperparameters ω from 0–4 and $|\mathcal{H}|$ from 2–4.

Latent-Dynamic Conditional Random Fields

The LDCRF [82] offers several advantages over CRFs and HCRFs by modeling both extrinsic dynamics between interruptibility classes as well as the intrinsic substructure within an interruptibility class. It does so by using hidden states, as the HCRF, and at the same time by removing the need to label an entire sequence with a single interruptibility class label (Fig. Figure 3.2d). In prior work, the LDCRF has been shown to outperform both the CRF and HCRF in Gesture Recognition [82], and therefore we consider it a good candidate for classifying interruptibility.

Mathematically, the LDCRF assumes that each sequence label y contains a corresponding set $\mathcal{H}_y$ of hidden states to capture intra-class substructures. Therefore, the LDCRF evaluates the following conditional model

$$P(Y|X) = \sum_{H} P(Y|H, X)P(H|X)$$

where $H = \{h_1, h_2, ..., h_T\}$ is a sequence of hidden states and each h_i belongs to the hidden state set $\mathcal{H}_{y_i}$ of its corresponding label y_i. To keep training and inference tractable, these sets are assumed to be disjoint for each class label. With the disjoint assumption, the conditional probability evaluated by the LDCRF reduces to

$$P(Y|X) = \sum_{H:\{h_1,...,h_T\}, h_i \in \mathcal{H}_{y_i}} P(H|X)$$

where $P(H|X)$ can be derived using the CRF formulation:

$$P(H|X) = \frac{1}{Z} \prod_{t=1}^{T} \Psi_t(H_a, X) \qquad Z = \sum_{H_a} \prod_{t=1}^{T} \Psi_t(H_a, X)$$

In our work, we use the same feature functions that we have for the CRF (Equation 3.1

and Equation 3.2), with suitable updates to the variables. The feature functions are again implemented using the HCRF library [2] and training is performed with BFGS. As with the HCRF and CRF, the LDCRF is trained to perform multiclass classification. We vary the value of the hyperparameters ω from 0–4 and $|\mathcal{H}|$ from 2–4.

3.5 Dataset for Interruptibility Classification

In this section we describe describe a dataset that we collected to evaluate the models introduced in Section 3.4 on their accuracy and robustness in interruptibility classification. Specifically, we seek to answer:

RQ1 Which computational features are useful in allowing a robot to classify interruptibility in an unstructured world?

RQ2 What is a robust model for obtaining interruptibility estimates from the proposed features?

Therefore, our dataset contains different subsets of the information categories presented in Section 3.3, each of which contains varying levels of information and noise.

3.5.1 Feature Subsets

Each of the sets of features in the dataset is additive in the features it is comprised of. Ultimately, the sets increase the amount of information presented to our models but at the cost of a corresponding increase in noise in those features.

Person State Features

We define the primary interruptibility cues about a person include head orientation, body position, and audible signals (Section 3.3). Since the recognition of some of these cues by a mobile robot in a public space can be noisy, we consider three subsets of features—*Minimal* (Min), *Standard* (Std), and *Extended* (Ext)—which are summarized in Table 3.1.

Our goal in this part of our work is to explore the robustness of the classification models to additional data and noise; we do not propose that any of the subsets is the best set of features for characterizing *person state* in general.

Minimal Feature Set We speculate that the most informative features for gauging interruptibility are the position of a person and an indication of whether they are looking at the robot or not. Therefore, we use *Min* to test our model performance when rich, but possibly noisy, data from other sensors (such as microphones), or from additional visual detectors (such as upper body detectors), is unavailable. This set contains:

Body Position: Tuple, (x, y), denoting the position of the body in the environment relative to the robot base.

Face Gaze: Boolean, *True* when a face is detected and the head is oriented towards the robot, *False* when a face is detected but the head is not oriented towards the robot or if the eyes are shut, and NaN when no face is detected.

Standard Feature Set This set of features represents the full breadth of information enumerated in Section 3.3 and is most similar to the features used in [78, 77]. In addition to *Min*, the set contains:

Body Orientation: Tuple, (z, w), of the quaternion, (x, y, z, w), denoting the rotation of a person's upper body relative to the robot's base frame. The (z, w) values specify rotation estimates about the upright axis and are thus the only meaningful values in the quaternion.

Audio Angle: Angle, in radians, to the dominant source of detected sound, calculated by a Kinect.

Audio Confidence: A $[0, 1]$ confidence measure for the *Audio Angle* estimate.

Feature	Min	Std	Ext
Body Position	×	×	×
Face Gaze	×	×	×
Body Orientation*		×	×
Audio Angle		×	×
Audio Confidence		×	×
Audio Angle Near Position*			×
Within Camera Field-of-View			×
Body Distance Thresholds			×
Linear Velocity			×
Quaternion Rate of Change*			×
Face Bounding Box*			×
Body Bounding Box*			×
Body Bounding Box Area*			×

*Unreliable data either due to sensor noise or unreliability.

Table 3.1: Membership of each *person state* feature to the different feature sets—Minimal (Min), Standard (Std), and Extended (Ext).

Extended Features Set In the final feature set we add additional features, some of which are noisy, to study the effects of extra data on model performance. The features are either obtained from the outputs of intermediate processing steps, such as the body bounding box, which is a supplementary output of the upper body detector, or are obtained through additional post-processing of *Std*, such as the field-of-view boolean, which maps a point in (x, y) to a boolean value indicating whether the point is in the field-of-view of the camera. These features have not been used in prior works but are added with the aim of making explicit some of the decision variables that we think might be useful for interruptibility. We surmise that the presence of the explicit decision variables will help the models, regardless of the effects of the noise. The variables include:

Audio Angle Near Position: Boolean, *True* when the *Audio Angle* estimate equals the angle from the camera to a detected person (within some tolerance), *False* when this is not the case.

Within Camera Field-of-View: Boolean, *True* when a detected person is within the field-of-view of the camera and *False* otherwise.

Body Distance Thresholds: Three booleans, each *True* if a detected person is beyond the boundaries of Hall's proxemic distances [119], and *False* if not. The boundaries considered are those of Personal Distance (0.46 m), Social Distance (1.22 m), and Public Distance (3.66 m).

Linear Velocity: Tuple, (v_x, v_y), obtained from the rate of change in *Body Position* between data segments.

Quaternion Rate of Change: Tuple, (v_z, v_w), obtained from the rate of change in *Body Orientation* between data segments.

Face Bounding Box: Four continuous values—*x*, *y*, *width*, and *height*—for the bounding box around a detected face.

Body Bounding Box: Four continuous values—*x*, *y*, *width*, and *height*—for the bounding box around a detected body.

Body Bounding Box Area: Area of the *Body Bounding Box*.

In all models but the HMM, continuous multivariate features, such as the *Body Position* tuple, are treated as separate vectors of univariate features. In the HMM, the features are left as multivariate because doing so provides us with the largest log likelihood values post-training. Similarly, combining the *Within Camera Field-of-View* boolean feature with the *Body Distance Thresholds* boolean features, and combining the *Audio Angle* feature with the *Audio Angle Confidence* feature, provides us with the highest log likelihood values for the HMM, and therefore these combinations are used in that model.

Interruption Context Feature

In order to evaluate the use of object recognition as a means of conveying the context of a scene, we additionally define an object label feature which can be added to any of the above feature sets. The object feature is defined as a set of boolean values, each of which is *True* or *False* if the corresponding object is present or absent within the scene. Objects are human-annotated (Section 3.5.2) and as such, we have perfect object labels. Therefore we simulate the noise expected from automated object recognition by randomly corrupting the boolean values in approximately 10% of the data segments of each interruptibility class label.

3.5.2 Dataset Creation

Our dataset contains robot sensor data from scenes containing small groups of people acting out staged scenarios in a public space (Figure 3.3).

Robot Sensors and Software

The robot used to collect the dataset was outfitted with a Hokuyo laser scanner, a Kinect One RGB-D camera, and an ASUS Xtion Pro Live RGB-D camera. The Kinect directional microphone array was used to collect audio data. We used the STRANDS perception pipeline [120] for people tracking at approximately 10 Hz and the Sighthound Cloud API[3] for face detection and tagging at 3–4 Hz.

Data Collection and Processing

During the data collection process, five people (not the author) were asked to take part in everyday activities in a common area of the building. Five data collection runs were conducted, each with 3–5 participants in the scene engaged in activities such as drinking coffee, having a conversation, or working on their laptops (Figure 3.3). The common area

[3] https://www.sighthound.com/products/cloud

Figure 3.3: Example scenes from the five data collection runs in the dataset in Section 3.5.2. The blue bounding box denotes individuals identified in the scene and the green bounding box denotes a face identified by the face recognition component. The interruptibility label of the identified individuals is also shown.

and activities were chosen because they allowed for a wide range of likely activities and a variety of visual scenes with different numbers of people and varying levels of occlusion. During each run, the robot was teleoperated through a preset series of waypoints that enabled it to observe the group from different perspectives; each run lasted an average of 108 seconds.

Following recording, the data was processed into segments that could be annotated with a person's interruptibility. Due to motion blur during navigation, only data from stationary robot observations was used. First, data from all sensor streams was segmented into 250 ms non-overlapping windows. For each sensor stream, the window of data was condensed into a single value consisting of the last recorded value for that sensor stream (if available). A Euclidean distance heuristic was then used to merge data for each detected person across all sensor streams. The result of this process was the creation of 1516 data segments, each of duration 250 ms, and each of which contained all the information, represented as features, available about a single person detected within the environment. Each segment was then annotated with ground truth interruptibility labels (details below).

Post-annotation, consecutive data segments were concatenated into sequences of minimum length 4 (1 second) and maximum length 8 (2 seconds), which resulted in the creation of 671 sequences. In the event of missing data (e.g., face recognition failure), missing values were filled in through linear interpolation for continuous valued features, or by propagating the last known value for boolean valued features. If neither approach was available, such as in the case where the beginning segment of the sequence was missing required

data, features were assigned a value of NaN to distinguish them from other valid values in the domain. During training and evaluation, the non-temporal models used the empirically determined value of -5 instead of NaN; the temporal models were modified to ignore NaN values. Additionally, during evaluation, and for training HCRFs, we defined the interruptibility label of a sequence to be the interruptibility label of the last segment in the sequence.[4] Non-temporal models were trained and evaluated on the data and label of the last segment in the sequence.

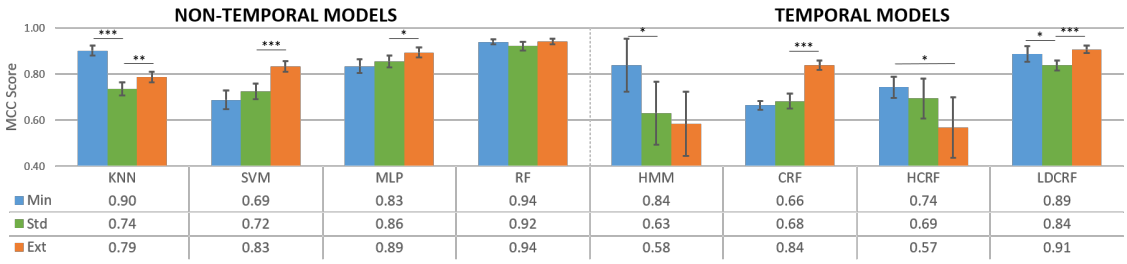
Annotation

We used the extended 5-point interruptibility scale from Section 3.2 to annotate each of the 250 ms data segments. Additionally, two independent coders were each asked to annotate a random subset consisting of approximately 40% of the data. To verify label consistency we calculated the Cronbach's Alpha measure of inter-rater reliability between our annotations and those of the other annotators, resulting in scores of 0.81 and 0.96. The high level of agreement highlights not only label reliability, but also the fact that humans are generally very consistent in judging the interruptibility of others.

We also annotated the data in this dataset with the labels of objects in the scene. The labels included *unknown, none, laptop, bottle, book, headphones, mug, phone_talk,* and *phone_text*[5]. The label *unknown* was frequently used in conjunction with the interruptibility label 0, which was used in situations when the person-of-interest was outside the camera field-of-view but detected by the laser and audio (leftmost example in Figure 3.3). Separate labels were assigned to phone use for speaking or texting (*phone_talk* and *phone_text*) because the activities correspond to different visual features and because the associated interruptibility of the person would likely also be different.

[4]No significant difference was observed in using alternate sequence labeling methods, such as mode of all segment labels.

[5]The last two features can be the output of activity recognition in addition to object detection. Activity recognition is another valid source of contextual features for interruptibility.



	KNN	SVM	MLP	RF	HMM	CRF	HCRF	LDCRF
Min	0.90	0.69	0.83	0.94	0.84	0.66	0.74	0.89
Std	0.74	0.72	0.86	0.92	0.63	0.68	0.69	0.84
Ext	0.79	0.83	0.89	0.94	0.58	0.84	0.57	0.91

Figure 3.4: Average MCC (MCC_{avg}) performance of each model in 10 fold cross-validation as a function of the feature sets. In Figure 3.4, Figure 3.5, Figure 3.6, and Figure 3.7, error bars indicate the 95% confidence interval and asterisks indicate level of statistical significance after Wilcoxon rank-sum test on MCC scores in each fold of cross-validation: * $p < 0.05$, ** $p < 0.01$, ***$p < 0.001$.

3.6 Evaluating Features and Model Robustness

In this section, we present a comparison of the classification models in the estimation of interruptibility based on the different feature sets introduced in Section 3.5.1 and then show the impact of adding contextual data in the form of object labels. In order to train the parameters for our models, we performed 10 fold cross-validation with 80% of the data in a fold used for training and 20% for testing. Results with the best performing hyperparameters for each model are reported using a Matthew's Correlation Coefficient (MCC) score for multiclass classification. The score, with a maximum value of 1.0 and with 0.0 indicating a performance no better than random, reflects a model's predictive power in a classification task in the presence of unbalanced class labels. Significance results are presented using a Wilcoxon rank-sum test using the MCC scores across the different folds of cross-validation.

3.6.1 Robustness to Noise

Figure 3.4 compares the performance of the classification models across the three feature sets without the inclusion of object context data. The results for each of the feature sets is analyzed below.

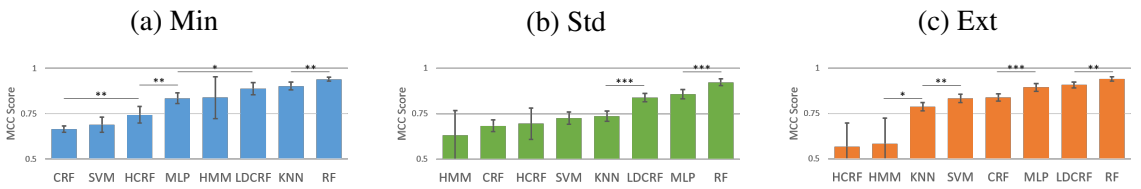


Figure 3.5: The classifiers ordered in increasing order of MCC_{avg} (Figure 3.4) for each of the feature sets.

Minimal Feature Set Figure 3.5a orders the eight classifiers in order of improving performance on the classification of interruptibility using the *Min* feature set. We note that the RF classifier is the overall best performing classifier with an MCC_{avg} of 0.94, and the LDCRF is the best performing temporal classifier with an MCC_{avg} of 0.89. The KNN classifier performs on par ($\Delta MCC = 0.01, p = .8$) with the LDCRF classifier, achieving a MCC_{avg} of 0.90. The HMM's performance, with an MCC_{avg} of 0.84, is also not significantly different than that of the LDCRF ($\Delta MCC = -0.05, p = .38$), but the high variance in its classification accuracy also does not differentiate it from the MLP ($\Delta MCC = -0.01, p = .68$), which achieves an MCC_{avg} of 0.83. Meanwhile, the MLP is noticeably less accurate than the LDCRF ($\Delta MCC = -0.06, p = .023$) and significantly more accurate than the HCRF ($\Delta MCC = 0.09, p = .0039$), which has an MCC_{avg} of 0.74. Finally, while the HCRF is not significantly better than the SVM ($\Delta MCC = 0.05, p = .11$), which achieves an MCC_{avg} of 0.69, the HCRF is significantly better than the CRF ($\Delta MCC = 0.08, p = .0029$), which achieves an MCC_{avg} of 0.66.

Standard Feature Set Figure 3.5b orders the eight classifiers in order of improving performance when using *Std*, which includes three additional features beyond the minimal set (*Body Orientation*, *Audio Angle*, and *Audio Confidence*). We observe that the RF classifier remains the best performing classifier for interruptibility classification, with an MCC_{avg} of 0.92. Similarly, despite a noticeable drop ($\Delta MCC = -0.05, p = .023$) in the performance of the LDCRF to an MCC_{avg} of 0.84, it continues to be the best performing

temporal classifier. The drop in LDCRF performance, coupled with an insignificant change ($\Delta MCC = 0.02, p = .17$) in the performance of the MLP, puts the performance of the LDCRF on par with that of the MLP ($\Delta MCC = 0.02, p = .22$), which achieves an MCC_{avg} of 0.86.

Overall, we notice that the use of the *Std* features either leaves the performance of the models unchanged, or causes a significant drop in classification accuracy. This drop is particularly evident in the case of the KNN ($\Delta MCC = -0.16, p < .001$) and the HMM ($\Delta MCC = -0.19, p = .014$). Although the Curse of Dimensionality [121] is a likely contributor to the observed penalty in the case of the KNN, the noise in the features of *Std* also plays a non-trivial role in the observed results. The *Body Orientation* feature, for example, is extremely noisy, with orientation values in some segments deviating by 90° or more from the ground truth. Most of the models show some sensitivity to this noise, with the HMM and KNN proving to be particularly susceptible. Although there is an insignificant change ($\Delta MCC = -0.05, p = .74$) in the performance of the HCRF, with an MCC_{avg} of 0.69, the increased variance in model's performance and the lower average score indicate that the HCRF might also be particularly susceptible to noise in its features.

Extended Feature Set Figure 3.5c presents the classification models ordered on performance with *Ext*, which includes eight features beyond *Std*. Several of these features contain additional information, such as the *Body Distance Threshold* booleans and the *Linear Velocity* tuple, but significant noise (see Table 3.1). Overall, the trends we observe in performance with *Std* hold with *Ext*. RF remains the best performing classifier with an MCC_{avg} of 0.94, outperforming the LDCRF ($\Delta MCC = 0.03, p = .0021$), which remains the best performing temporal classifier with an MCC_{avg} of 0.91. The MLP continues to perform on par ($\Delta MCC = -0.02, p = .32$) with the LDCRF, with an MCC_{avg} of 0.89, thereby preserving the ordering of the models observed with *Std*.

The performance on *Ext* reveals the HCRF and HMM sensitive to noise, with significant

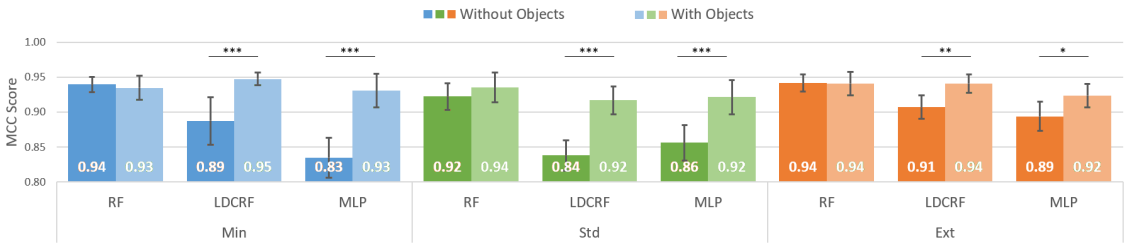


Figure 3.6: Effect of adding object labels as features to the different feature sets.

drops in the performance of both models relative to their performance with *Min* ($\Delta MCC = -0.17, p = .029$ for the HCRF, and $\Delta MCC = -0.26, p = .01$ for the HMM). Meanwhile, the remaining models reveal themselves to be tolerant to noise, with the CRF ($\Delta MCC = 0.16, p < .001$) and SVM ($\Delta MCC = 0.11, p < .001$) in particular showing significant improvement in their MCC_{avg} scores with the addition of more information with *Ext*.

Summary In summary, we first note that, as a partial answer to the question of what features might be relevant to the classification of interruptibility (**RQ1**), the hypothesized *person state* features mentioned in Section 3.3 are relevant because our models achieve high MCC scores with all subsets of those features—*Min*, *Std*, and *Ext*. Next, to answer the question of finding a robust model for interruptibility classification (**RQ2**), we find that the RF model consistently outperforms all other models across all feature sets, remaining robust to noise in features but also remaining unaffected by any additional information in them. In contrast, the MLP and LDCRF perform comparably to RF, especially with *Ext* features, and both show an ability to learn from the additional information available in the features while also remaining robust to noise.

In the following subsection, we complete our investigation into the features relevant for interruptibility classification by examining the performance of the RF, MLP, and LDCRF with the addition of object label features, which provide information about the interruption context (Section 3.3).

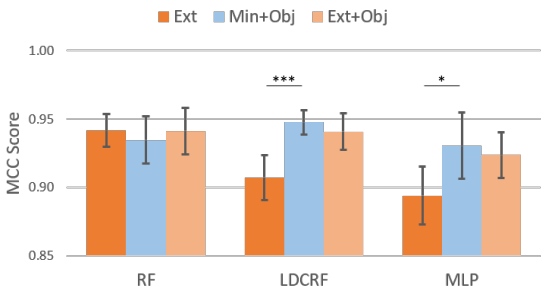


Figure 3.7: Comparison of RF, LDCRF, and MLP performance with *Ext* features to their performance with *Min* and *Ext* features augmented with object labels.

3.6.2 Adding Object Context

Figure 3.6 presents the classification performance of the RF, LDCRF, and MLP classifiers after adding object recognition features to each of the three feature sets (*Min*, *Std*, and *Ext*). Overall, we note that the addition either improves classification performance or leaves it unchanged, thereby implying that object labels are a good cue to interruptibility. In the case of RF, we find that the object labels do not affect classification performance, which is similar to the trend observed in the Figure 3.4 where the inclusion of additional features from *Min* to *Ext* does not affect the classification performance of the RF model. Conversely, the LDCRF experiences consistent gains in classification performance from the addition of object label features to *Min* ($\Delta MCC = 0.06, p < .001$), *Std* ($\Delta MCC = 0.08, p < .001$), and *Ext* ($\Delta MCC = 0.03, p = .0021$). The MLP also experiences a significant improvement in classification performance with the addition of object labels to *Min* ($\Delta MCC = 0.10, p < .001$), *Std* ($\Delta MCC = 0.07, p < .001$), and *Ext* ($\Delta MCC = 0.03, p = .043$).

In fact, as shown in Figure 3.7, we find that in lieu of adding a large set of somewhat noisy features to *Min* (as we do with *Ext*), adding the more precise object label features (*Min+Obj*) leads to better classification of interruptibility, particularly for the LDCRF ($\Delta MCC = 0.04, p < .001$) and the MLP ($\Delta MCC = 0.04, p = .043$). Additionally, we find that the object labels provide sufficient information for interruptibility estima-

tion, with no significant improvement in interruptibility classification performance between *Min+Obj* and *Ext+Obj* for any of the three models, RF ($\Delta MCC = 0.01, p = .44$), LDCRF ($\Delta MCC = -0.01, p = .32$), and MLP ($\Delta MCC = -0.01, p = .97$).

Therefore, to complete an answer to the question of what features might be relevant to the classification of interruptibility (**RQ1**), we can state that features about the interruption context, such as object labels, are also relevant.

3.6.3 Conclusions

In this section, we answer our first two research questions and find that:

1. As proposed in Section 3.3, we can use both *person state* features, and cues of the *interruption context*, such as object labels, to classify interruptibility in an unstructured world.

2. In the absence of robust detection of social cues (for example, noisy upper body detection), robust context detection (for example, accurate object detection) can be a good substitute.

3. The RF, LDCRF, and MLP classifiers are good candidates for robust interruptibility classification from the proposed features.

The second finding is especially significant in robot application domains where it might be difficult to obtain reliable person tracking information, but easier to obtain contextual signals in the form of object detection.

To answer our remaining research questions and to fully evaluate our findings from this section, we followed the above evaluation with a user study in which we deployed a model for online interruptibility classification on a robot. The following sections introduce the user study, elaborate on the system we designed for online interruptibility classification, and then present results from using the system in the user study.

Figure 3.8: The robot interrupts a participant engaged in a building task.

3.7 Effects of Interruptibility Classification: User Study

Our research seeks to develop interruptibility-awareness in robots and to evaluate the effects of this capability on human task performance, robot task performance, and on the human's interpretation of the robot's social aptitude. Therefore, we also focus on the following research questions:

RQ3 Can we use measures of the robot's behavior to show that our models accurately estimate interruptibility online on a robot platform?

RQ4 How does interruptibility-aware robot behavior affect human task performance when a robot regularly needs assistance?

RQ5 How does interruptibility-aware robot behavior affect robot task performance when relying on humans for assistance?

RQ6 Does a robot appear more socially adept if it interrupts humans at appropriate moments?

In order to evaluate these questions, we conducted a between-subjects user study in which human participants took part in a mock manufacturing assembly activity. Participants were given construction tasks while a robot with tasks of its own would occasionally interrupt them to request assistance (Figure 3.8). The study had three conditions in which

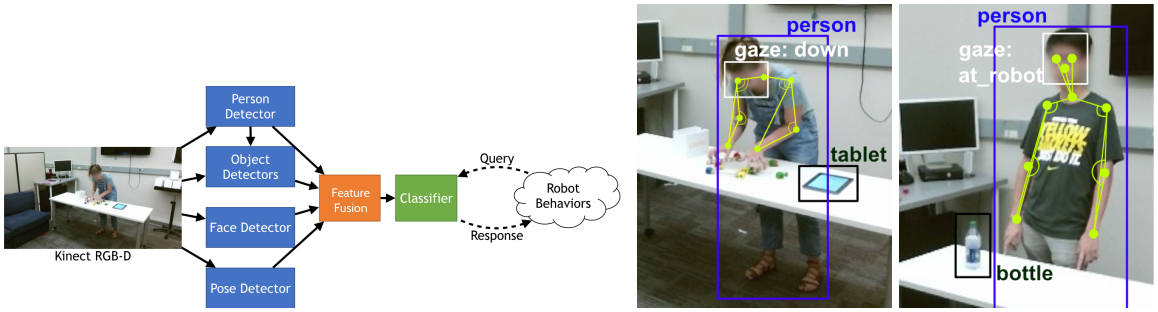
we varied the mechanism used by the robot to select an appropriate moment to interrupt the participant.

Random interruptions (RND) The robot interrupted participants after it waited for a random amount of time, reflecting the current behavior of interruptibility-unaware robots. For example, the robots evaluated by Mutlu and Forlizzi [27] operated in the same environment as hospital staff, interrupting them randomly to gain attention as needs arose. In our study, the robot's algorithm tried to emulate this behavior by randomly selected a wait time from a uniform distribution in the range [0,30] sec; after which, it flipped a fair coin every 0.5 sec to decide whether to interrupt. Wait times in the study ranged from 2 to 37 sec.

Wizard-of-oz interruptions (WOZ) The robot interrupted participants when a human (wizard) signaled it was an appropriate time. Wizards were provided with a real-time video feed from the robot's camera and, during pilot trials, were instructed to make moment-by-moment decisions to interrupt the participant or not, simulating the decision made by our interruptibility models[6]. Once the decision to interrupt was made, the wizards could perform no more actions until the next robot incursion into the study space. During study trials, there was no interaction between the experimenters and the wizard. We recruited two wizards and observed that, despite similar instructions, differing social norms and attitudes among individuals led one wizard to be more conservative in their interruptions than the other. We therefore had each wizard participate in 50% of WOZ trials to help account for this effect.

Model-based interruptions (MDL) The robot interrupted participants based on output from an LDCRF classifier implemented within a system to perform online interruptibility classification. We chose the LDCRF over the RF and MLP due to our evaluations in

[6]The wizards were asked to (1) treat images from the video at each moment as a static image to decide whether they would interrupt the participant at that moment, (2) specifically ignore the screen on the participant's tablet and the task schedules that they were becoming accustomed to (to the extent possible), and (3) give the robot and human tasks equal importance.



(a) Classification pipeline for interruptibility. (b) Visualization of the detected features.

Figure 3.9

Section 3.8.2.

In the following sections, we first describe our computational framework to enable online interruptibility classification and our process for choosing the appropriate classification model. We then present the design of and results from the user study to answer the above research questions.

3.8 Computational Framework

Our computational framework consists of two principal components: the perception system that identifies people in a scene and extracts feature vectors characterizing their state, and the classification model that classifies the interruptibility state of each person in the scene. Figure 3.9a summarizes the computation pipeline, which runs on our mobile robot equipped with a Microsoft Kinect RGB-D camera.

3.8.1 Perception System

The perception system of the robot (1) detects people in the scene, (2) uses a series of detectors to analyze the state of each individual, and (3) merges the output of the detectors into a feature vector for processing by the classification model. The feature vector, its features enumerated in Table 3.2, is emitted by the perception system at about 2.5 Hz.

Features	Detector(s)
Face Gaze Estimate: $at\_robot\|left\_right\|down$	Face
Skeletal Angles & Vectors: $angle\_left\|right\_elbow$, $angle\_left\|right\_wrist$, $angle\_left\|right\_shoulder$, $angle\_left\|right\_eye$, $nose\_vec\_x\|y$	Pose
Object Counts: $book$, $bottle$, $bowl$, cup, $laptop$, $cellphone$, $tablet$	Objects

Table 3.2: The features emitted from the perception system to classify the interruptibility of observed people.

Person Detector: We use the You Only Look Once (YOLOv2) [122] deep neural network to detect people in the scene. This detector was chosen for ease of use and setup, and for its accuracy and speed. It never missed a person in our user study, and published person detections at $>$10fps.

Feature Detectors: Once a person has been identified in the scene, we employ several deep networks to extract interruptibility-relevant features about the person. We include features from the prior work, such as the coarse gaze estimate of a person and the objects associated with them, and introduce the skeletal data for improved classification.

Face Detector: We use a cascaded deep network [123] for face detection and coarse gaze estimation. The detector returns facial keypoints, which we translate into an enumerated gaze estimation variable. Features: *at_robot*|*left_right*|*down* (Enum). Framerate: 7-10fps.

Object Detector 1: We use another implementation of YOLOv2 that runs over higher resolution images which are cropped to include regions around people in the scene—information that is obtained from our person detector. This detector was trained on MSCOCO [124] and returns counts of objects and their positions. Features: *book*, *bottle*, *bowl*, *cup*, *laptop*, and *cell phone*. Framerate: $>$10 fps.

Object Detector 2: We use Faster-RCNN [125] fine-tuned to identify study-related objects on the table, in our case the tablet participants used throughout the study. As with our

other object detector, this returns counts and positions of detected objects. Features: *tablet*. Framerate: $>$10 fps.

Pose Detector: We use a convolutional pose machine (CPM) [126] to infer a person's skeletal keypoints. These keypoints are then refined into joint angles and vectors for our classifier. Features: *nose_vec_x|y*, *angle_left|right*: *elbow*, *wrist*, *shoulder*, *eye*. Framerate: 5-7 fps.

Feature Fusion: Each of the above detectors runs in parallel and at different rates. The Feature Fusion module uses the Euclidean distance heuristics mentioned in Section 3.5.2 to aggregate the output of the various detectors into a single feature vector describing the most up-to-date estimate of the scene. Concretely, the module polls the person detector at a rate of 2.5 Hz to track every person identified by the detector. It then uses its heuristics to associate the latest data from the other detectors to its database of tracked people. If a detector does not contain information about the person of interest, the fusion module inserts a placeholder value of NaN for the corresponding attribute in the feature vector described in Table 3.2.

3.8.2 Classification Model

The classification model outputs the interruptibility of a person of interest given a stored buffer of feature vectors from the Perception System[7]. Based on our evaluations in Section 3.6, we considered the RF, LDCRF, and MLP classifiers when building our computational framework; further evaluations, which we summarize below, led us to choose the LDCRF over the other two classifiers. In this subsection, we introduce both the dataset we used to evaluate the different models and the evaluation we performed to select the final model for our study.

[7]During our user study, we stored a buffer containing 4 secs of feature vectors and used the buffer for data imputation, as described in Section 3.5.2.

Figure 3.10: Example timeline of a trial with the tablet ground truth, the human annotations, and the model predictions. Orange shows uninterruptible (0) while blue shows interruptible (1); gray indicates that there was insufficient data for the model to make a classification. Black indicates breakpoints between different moments of observation by the robot during the course of the trial.

***Dataset**:*

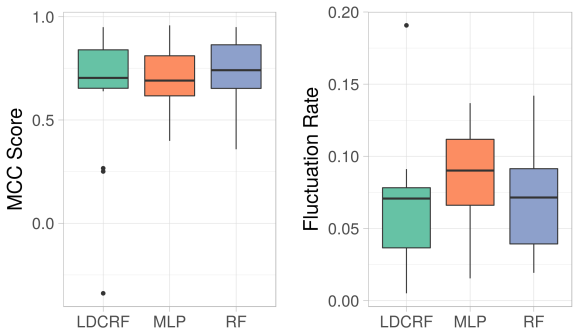
We evaluated the models on data collected over the course of 4 pilot runs and 11 runs of the RND condition of the study introduced in Section 3.7. Two of the coauthors annotated the collected data on a binary scale of interruptibility[8], with 0 as uninterruptible and 1 as interruptible[9]. Cronbach's Alpha score of inter-rater reliability was 0.97 between the co-authors. The models were trained on one of these two annotations.

In addition to human annotations of interruptibility, we obtained ground truth interruptibility labels of participants from the tablets provided to them in the study (Section 3.9), where the ground truth label for a participant was 0 if they were provided a build assignment on their tablet, and 1 otherwise. The Cronbach's Alpha score was 0.95 for each of the annotators with the ground truth labels[10]. We trained and tested our models on the human annotated labels because the ground truth labels did not always correlate to the social cues of interruptibility projected by the participant.

[8]As mentioned in Section 3.2, a binary scale of interruptibility is most intuitive, with the extended interruptibility classification scale being of use in situations involving multiple, potentially occluded, people. In our study, such conditions do not arise, allowing us to use the more intuitive binary scale.

[9]The annotators operated under the instructions provided to the wizards in the WOZ condition: they observed a video feed from the robot's camera and provided a moment-by-moment label of whether the participant was interruptible.

[10]The ground truth rating sometimes differed from the human annotation if the participant chose to ignore the tablet build or if they appeared busy in another task despite the tablet marking them as available.



(a) (b)

Figure 3.11: Model Performance

Method:

Similar to the process described in Section 3.6, we tested all hyperparameter configurations of the models with five-fold cross-validation, with special care undertaken to ensure that none of the data from any of the study trials was shared between the train and test sets. We evaluated the models on two metrics: the first, an *MCC* score to gauge classification accuracy, and the second, a measure of the fluctuation rate in model prediction, *FR*, similar to the one used by Foster et al. [91]. Our measure of fluctuation is calculated as follows:

$$fluctuation\_rate(FR) = \frac{num\_prediction\_changes}{total\_num\_predictions}$$

Values for *FR* vary from 0–1, with ideal values as close to the *FR* of human labels as possible, which in turn is almost always 0.

Result:

Fig. Figure 3.11 presents the performance of the three models across the data in the 15 trials on the metrics of the *MCC* score and fluctuation rate, *FR*. A Kruskal-Wallis test indicates that there is no significant difference between the models on the metric of *MCC* score ($H(2, N = 45) = 0.63, p = .73$) or on the metric of *FR* ($H(2, N = 45) = 4.0, p = .13$).

Due to the lack of significant signal from the evaluation metrics to aid us in choosing a

model, we conducted additional tests on the robot using all three models. Empirically, we found that despite favourable MCC scores, the MLP and RF classifiers performed poorly in practice, often oscillating between interruptibility classes with miniscule changes to the scene. The RF classifier was particularly prone to this problem, often varying its classification output within a seemingly static scene. Our experiences corroborate those of Foster et al. [91], where the authors deemed their temporal CRF classifier more appropriate for use on their robot due to greater stability in classification output, despite inferior classification accuracy to other non-temporal models. We note that the discrepancy is ill-studied and scope for further research.

In conclusion, based on our evaluations, we used the LDCRF as our classifier of choice for the study trials in the MDL of our user study. The following section introduces the full design of the study.

3.9 User Study: Design

We conducted a between-subjects user study to evaluate the research questions outlined in Section 3.7. The study involved 48 trial participants[11]. Six trials were excluded from the study analysis: two due to hardware malfunction, and four due to participants deviating from the study protocol. The resulting 42 participants (20 women, 22 men) were aged between 21 and 29 ($Mdn = 24$). The study took approximately 50 min, and participants were paid $10 USD.

3.9.1 Study Procedure

We devised a skill-based experimental task in which human participants took part in a mock manufacturing assembly activity. Participants were instructed to construct structures (*builds*) out of wooden pieces (Figure 3.12b), and told that their build process would be

[11] Six additional participants took part in pilot trials used to tune build complexity, robot behavior, train the classification models, and to familiarize the wizards with their interface.

video recorded to be used later as training data for the robot. Additionally, participants were told that the robot was performing and studying its own builds, and that it would occasionally enter the space to request assistance.

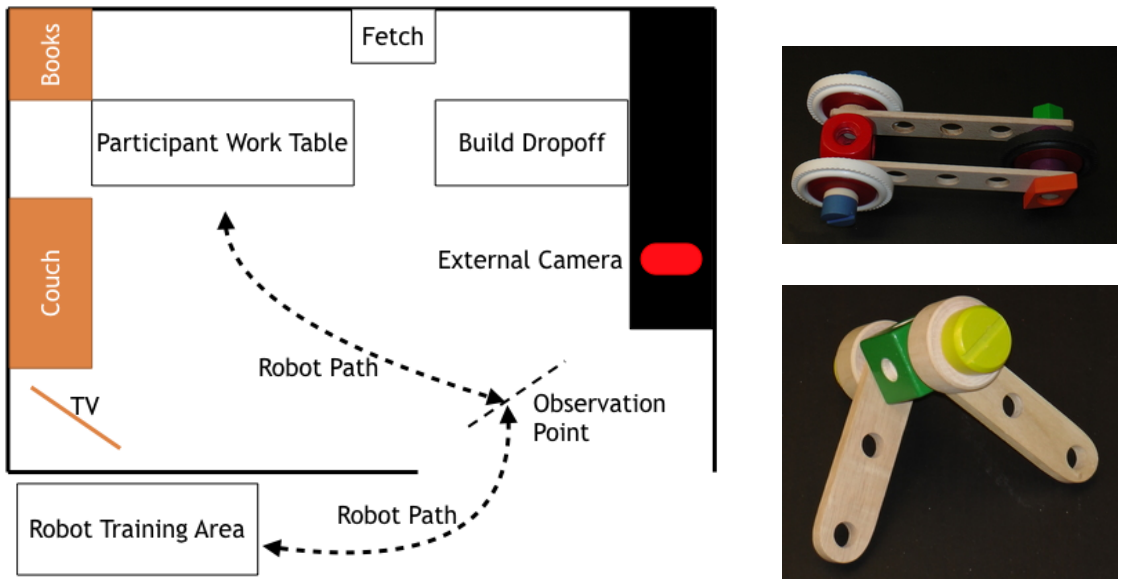
Pre-Study: Upon arrival, participants were briefed on the study, completed consent forms, and filled in a pre-study questionnaire. Nearby, to support the narrative of the robot learning to construct builds, an experimenter could be seen "training"[12] the robot by responding to the robot's questions (e.g., "Is this a correct build?").

Study Space: After the study briefing, participants entered the building area (Figure 3.12a), consisting of an enclosed space with fetch area for retrieving build components, a work area for construction, and a dropoff area for completed builds. A key element of the study design is that the study schedule was split into periods of work and leisure to ensure that participants had periods of low and high interruptibility. To induce participants to showcase a diverse range of natural leisure behaviours (to fully evaluate the performance of the classifier and generalizability of our system), the room included a TV playing muted videos[13], a stack of books, and a couch. Participants were also allowed to keep their cell phones. Overall, during breaks 64% sat on the couch, 50% used their cell phones, 40% drank a refreshment, and 14% read a book.

For the remainder of the study period, participants alternated between constructing builds (*build*) and break times (*idle*), while being occasionally interrupted by the robot. Figure 3.12c presents an example timeline.

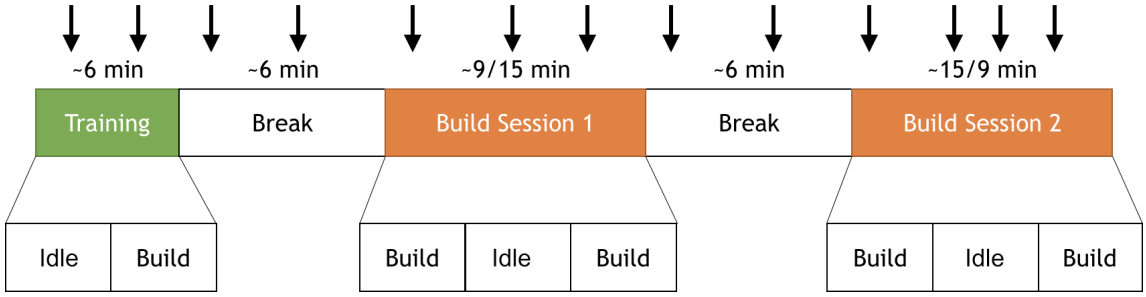
Builds: Each participant trial consisted of 3 build sessions. The first build session was a training session during which participants were allowed to ask questions and acclimate themselves to the task and the robot. We do not report data from this session. Sessions 2 and 3 each consisted of two builds, with a short break in between. Instructions for each

[12]No actual training of the robot occurred during the study trials.
[13]http://bit.ly/2xR65aG



(a) Map of the study area

(b) Builds



(c) Sample trial timeline, arrows indicate interruptions

Figure 3.12

build were provided on a tablet located on the work table; the tablet remained blank until the designated build time, and presented a NASA-TLX [127] workload questionnaire each time the participant selected that they had completed a build. The build sessions were either 15 min or 9 min in length, and were presented to all participants in a counterbalanced manner. The different length build sessions were configured to provide differing degrees of time pressure on the participant. In addition, pilot studies showed that some participants improved in build performance due to learning; the counterbalanced sessions were used to amortize the effects of learning on high time pressure and low time pressure sessions. All builds in a build session had a time limit, and participants were shown a countdown timer

30 sec before the end of this time limit; participants were not allowed to work past the end of the limit.

Breaks: Each trial included two break times approximately 6 min in length (differences in duration occurring due to robot interruptions), during which the tablet was taken away and the participants were invited to rest on the couch. The purpose of the break was to expose the robot to interruptible human behavior. In both cases, the experimenters presented fictitious excuses to the participant for pausing the study, in one case claiming a non-existent tracking device required adjustment, and in the other case simulating a tablet malfunction. For both breaks, experimenters explained the pause in the experiment, invited participants to wait on the couch, and then returned at the end of the break to "continue" the study. Participants were told that the robot interruptions would continue since the robot remained unaffected by the glitch.

Robot Interruptions: The robot continually entered the building area looking for assistance from the start to the end of a trial. The schedule of these entrances was not predefined and the robot was sent back in as soon as it returned from an interruption. The first three robot entrances coincided with the training build session and part of the first break; we allowed participants to ask questions during these interruptions and do not report data from them. The robot was equipped with a small box containing the blocks for its builds and a tablet, which provided instructions to the robot builds.

During an entrance, the robot followed the path shown in Figure 3.12a. It waited at the observation point upon entering and after waiting—a random duration in RND, until 2.5 sec of consecutive[14] interruptible classifications in MDL, or until the wizard sent an interruptible signal in WOZ—chose to move toward the participant. Upon arrival, the robot verbally requested assistance and waited for 2 min. Participants were aware of the wait duration and could accept the interruption within the time limit by grabbing the tablet, at

[14]Empirically, 2.5 sec of consecutive classifications at 2 Hz worked well.

which point the robot waited indefinitely until the build was completed. If the participants did not respond in 2 min, the robot left the participant build area. Upon returning to the training area, the robot audibly requested verification of the build (e.g., "Is this a correct build?") from an experimenter. The experimenter provided a Yes/No response on whether the interruption was built, prepared the next robot build in the box, and sent the robot back in.

Post-Study: After the last build session, participants were asked to complete a post-study questionnaire, were debriefed on the true purpose of the study, and the deceptions that we employed.

3.9.2 Hypotheses

Our central premise is that the robot in MDL & WOZ will interrupt at appropriate moments, and that such interruptions will improve robot task performance and the social perceptions of the robot compared to those metrics in RND. Based on results from HFE research (Section 3.1.2), we also predict that human task performance will not be greatly affected. Specifically, we formulate:

H1 (RQ3) With an interruptibility classifier (MDL), the robot will interrupt fewer builds than it would without the classifier (RND), waiting longer to interrupt when participants are building and interrupting more quickly when they are idle. In addition, the robot with the classifier will interrupt as many builds as a robot directed by a human (WOZ).

H2 (RQ4) When the robot has interruptibility-aware behavior (MDL & WOZ), participant task performance will not significantly differ from participant task performance when a robot interrupts at random (RND).

H3 (RQ5) When the robot has interruptibility-aware behavior (MDL & WOZ), fewer of its tasks will be ignored and it will not need to spend as much time awaiting human assistance as it will when it interrupts at random (RND).

H4 (RQ6) Participants will perceive an interruptibility-aware robot (MDL & WOZ) as more socially aware and considerate than one that interrupts at random (RND).

3.9.3 Measurements

Prior work in HCI and HFE quantifies task performance using metrics such as time on task [74, 84, 66, 99], the number of tasks completed [74], and task switching time [75, 85, 99]. We use similar quantitative measures of human and robot performance, and 5-point Likert scale responses to questions of participant opinions and participant background:

M1 (RQ3) Percentage of builds interrupted by robot; robot wait (to interrupt) time when participant is on build; robot wait (to interrupt) time when participant is idle.

M2 (RQ4) Participant's time idle; total number of tasks done.

M3 (RQ4/RQ5) Number of interruptions of the participant; number of interruptions ignored; interruption lag, measured as the time between when the robot requests assistance and the participant begins constructing the robot build; interruption duration, measured as the total time the robot waits after it has requested assistance.

M4 (RQ6) Perceived appropriateness of timing[15]; perception of robot's considerateness (workload-awareness)[16].

M5 (Control) Experience with building blocks; proficiency at multitasking; familiarity with robots; motivation and anxiety during trial; difficulty of trial; predictability of robot interruptions. These measures instrumented factors that had the possibility to confound results based on results in prior literature and our experience from the pilot studies.

Most quantitative measures were automatically logged from timestamps on the tablet and the robot, but some discrepancies caused by unexpected participant behavior[17] were corrected using video from the external camera. For all trials, timestamps from the tablets

[15] Q1: When the robot interrupted you, was it a good time to interrupt?

[16] Q2: Did the robot take your workload into consideration when asking for help?

[17] For example, ignoring a build on the main tablet, or picking up the robot tablet and then replacing it without completing the robot build

are treated as ground truth of participant interruptibility. In addition to the above metrics, we also allowed participants to verbally elaborate on their choices and reasoning during post-study debriefing.

3.10 User Study: Results

In this section we examine the results from our user study to draw conclusions on the effects of interruptibility classification.

3.10.1 Analysis of Model-Driven Robot Behavior

We first evaluate the performance of the robot's interruptibility model in the study setting and explore metrics pertaining to the question, "Can we use measures of the robot's behavior to show that our models accurately estimate interruptibility online on a robot platform?" (**RQ3**). Our analyses in this section are conducted using a one-way analysis of variance (ANOVA) with the study condition as an independent variable. The ANOVA is followed by post-hoc comparisons using Tukey's HSD. Results for this section can be seen in Figure 3.13.

Results:

We first examine the amount of time the robot waited at the observation point when participants were busy or idle as an indication of moment-to-moment interruptibility classifier accuracy. Concretely, we expect an accurate classifier to make the robot wait longer when a participant is busy, and not as long when the participant is idle. Over the course of each of the 42 trials, the robot entered the manufacturing environment between 2–6 times when the participant was busy, and between 2–9 times when the participant was free. Of the robot entrances when the participant was busy, the data shows a significant difference between the conditions ($F(2, 39) = 113, p = 6.0e^{-17}$), with the robot waiting longer, on average, (Tukey HSD, $p = .014$) in MDL ($M = 49.4, SD = 31.0$)

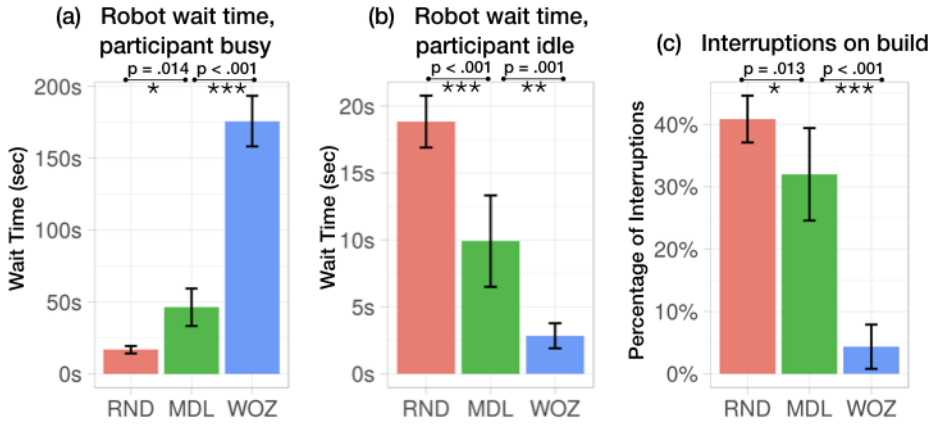


Figure 3.13: Data and analysis for results in Section 3.10.1. In Figure 3.13, Figure 3.14, Figure 3.15 & Figure 3.16, asterisks indicate level of statistical significance after post-hoc tests: * $p < .05$, ** $p < .01$, ***$p < .001$. Error bars in the bar charts indicate the 95% confidence interval.

than in RND ($M = 16.5, SD = 4.27$), and longer (Tukey HSD, $p < .001$) in WOZ ($M = 175, SD = 40.2$) than in MDL (Figure 3.13a). Of the robot entrances when the participant was idle, there was again a significant difference between the conditions ($F(2, 39) = 39.7, p = 4.0e^{-10}$), with the robot waiting less time, on average, (Tukey HSD, $p < .001$) in MDL ($M = 10.2, SD = 6.5$) than in RND ($M = 19.3, SD = 4.62$), and less time (Tukey HSD, $p = .0011$) in WOZ ($M = 3.06, SD = 2.60$) than in MDL (Figure 3.13b).

We next examine the percentage of interruptions per trial that occurred during a build. We expect that a more accurate interruptibility classifier will have a lower percentage of interruptions in the middle of a build. As shown in Figure 3.13c, the data from the 14 trials indicates a significant difference between the conditions ($F(2, 39) = 74.8, p = 4.5e^{-14}$), where the percentage is lower (Tukey HSD, $p < .001$) in WOZ ($M = .043, SD = .061$) than in MDL ($M = .32, SD = .13$), and lower (Tukey HSD, $p = .013$) in MDL than in RND ($M = .41, SD = .065$).

We observe significant differences between our two wizards in the above metrics. The conservative wizard (wizard **C**) never interrupted a participant in the middle of a build ($M = 0, SD = 0$), while the aggressive wizard (wizard **A**) preferred to interrupt as a

participant completed their task, sometimes catching them at the end of a build ($M = .087, SD = .061$). As a result, the robot's wait time at the observation point differs between the wizards. However, both wizards' metrics are closer to each other than to MDL or RND.

Summary:

Our results support our hypothesis that an interruptibility model showcases behavior indicating a tendency to interrupt participants at appropriate moments (**H1**). In this study, we defined an appropriate interruption as one that occurs when the participant is idle, and not engaged on a tablet build. The above analyses show that the classification model results in appropriately timed interruptions, with a model-equipped robot approaching participants quickly when they are free while waiting to approach when they are busy. Examining these metrics, it is clear that a robot equipped with an interruptibility model is socially aware through its ability to use the model to autonomously select appropriate times to engage with people. However, the robot controlled by a wizard is the most interruptibility-aware, indicating that we still have room to improve the model in order to achieve human-level accuracy.

3.10.2 Analysis of Human Task Performance

The results above validate that an interruptibility-aware robot has an increased likelihood of making appropriately timed interruptions. In this section, we explore the effects that this change in robot behavior has on human task performance. Specifically, we examine the metrics relevant to, "How does interruptibility-aware robot behavior affect human task performance when a robot regularly needs assistance?" (**RQ4**).

Results:

We find that the self-reported rating of experience with building blocks (*build experience*) was a significant confounding factor in participant build proficiency. Correlating self-

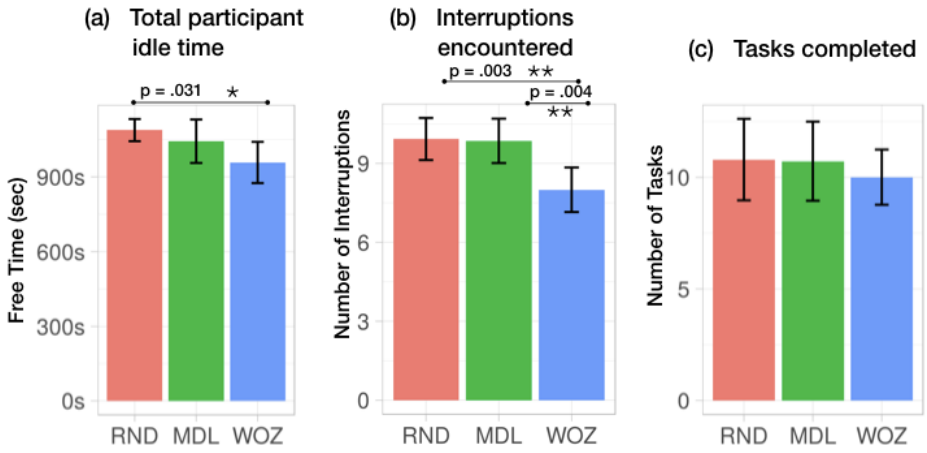


Figure 3.14: Data and analysis for results in Section 3.10.2.

reported experience to observed task performance, we observe most difference between those who self-reported experience as 1 or 2 (*low* experience), and those who reported experience of 3 or higher (*high* experience). Participants with *high* and *low* experience were similarly distributed between conditions, with 10 *high*, 4 *low* experience participants in RND and MDL, and 9 *high*, 5 *low* experience participants in WOZ. The following analyses control for build experience.

Idle Time: We assume that moments when the participant is idle are moments of lost productivity; even while the main builds are unavailable, the robot has tasks that can be completed. We therefore wish to minimize participant idle time. For the 14 trials in each condition, a two-way ANOVA with the study condition and build experience as independent variables shows a significant effect of study condition ($F(2, 36) = 3.36, p = .046$) and no significant effect of build experience ($F(1, 36) = 1.95, p = .17$). A post-hoc Tukey's HSD reveals lower ($p = .031$) idle time in WOZ ($M = 957, SD = 143$) than in RND ($M = 1087, SD = 76.7$), but no significant difference ($p = .64$) between MDL ($M = 1043, SD = 151$) and RND, or between MDL and WOZ ($p = .20$) (Figure 3.14a).

Interruptions Encountered: In our study, the robot continually re-entered the building area to interrupt, which resulted in participants that attended to interruptions quickly receiv-

ing more interruptions. Therefore, the number of interruptions presented to a participant is an indication of the number of tasks that they encountered[18]: another indicator of task performance. For the 14 trials in each condition, a two-way ANOVA with study condition and build experience shows a significant effect of study condition ($F(2, 36) = 7.63, p = .0017$) and no significant effect of build experience ($F(1, 36) = 2.04, p = .16$). A post-hoc Tukey's HSD reveals a lower ($p = .0031$) number of interruptions encountered in WOZ ($M = 8, SD = 1.47$) than in RND ($M = 9.93, SD = 1.38$) and a lower ($p = .0044$) number in WOZ than in MDL ($M = 9.86, SD = 1.46$), with no significant difference ($p = .99$) between MDL and RND (Figure 3.14b).

Interruptions Ignored: Participants were given the freedom to ignore robot interruptions during the study. We expected such ignores to occur when participants were overwhelmed, and therefore consider the number of interruptions ignored as a negative indicator of human task performance. For the 14 trials in each condition, a Kruskal-Wallis test shows a significance of study condition ($H(2, N = 42) = 7.15, p = .028$) and marginal effect of build experience ($H(1, N = 42) = 2.95, p = .086$). A post-hoc pairwise Wilcoxon rank sum test with Benjamini & Hochberg [128] correction reveals a lower number of interruptions ignored in WOZ ($Mdn = 0$) than in RND ($Mdn = 1.5$, Post-hoc Wilcoxon, $p = .033$) or MDL ($Mdn = 1$, Post-hoc Wilcoxon, $p = .033$), with no significant difference between MDL and RND ($p = .94$) (Figure 3.15a).

Tasks Completed: The final measure is the total number of tasks (builds + robot interruptions) that were completed by participants during a trial. For the 14 trials in each condition, a two-way ANOVA with study condition and build experience as independent variables finds a significant effect of build experience ($F(1, 36) = 14.9, p = .0004$) and no significance with study condition ($F(2, 36) = 0.22, p = .8$) (Figure 3.14c).

We again comment on the differences between our wizards. Although there is no differ-

[18] All participants received 4 tasks from the main tablet.

ence between the wizards in the amount of idle time, interruptions encountered, and tasks completed, the wizard **C**'s interruptions were ignored less often ($Mdn = 0$) than wizard **A**'s ($Mdn = 1$).

Summary:

The results above lead us to mixed conclusions regarding the effects of interruptibility-aware robot behavior on human task performance (**H2**). Firstly, we find that idle time is minimized by the awareness of interruptibility, with participants exposed to the perfect interruptibility-aware robot in WOZ enjoying significantly less idle time than participants in RND. An obvious cause of the reduced idle time is the robot's behaviour in waiting to interrupt participants until they are free (Section 3.10.1), which in turn causes participants in WOZ to encounter fewer tasks than participants in either MDL or RND. However, we find that waiting until participants are free leads to fewer interruption builds that are ignored, thereby offsetting the potential cost to throughput incurred by presenting fewer tasks to people. Ultimately, we find the factors relating to task throughput balance each other such that the total number of tasks completed by humans is not significantly different due to interruptibility-aware behaviour.

The tradeoff in the factors affecting task completion explain the similar throughput between RND and WOZ, but they fail to explain the similarity in the task metrics between RND and MDL, despite the results in Section 3.10.1 showing that the robot tended to wait longer and interrupted fewer builds in MDL. This discrepancy is explained in part by the results from HFE research (Section 3.1.2), which suggest the embodiment of the robot interruptions and the skill-based main task contributed to unaffected task performance: it is likely that participants were able to optimize their build process such that their performance remained unaffected on the metrics of task throughput that we instrumented. With better instrumentation, future research has the potential to examine additional metrics of task performance, such as interruption resumption lag [75, 85, 99], which should differ between

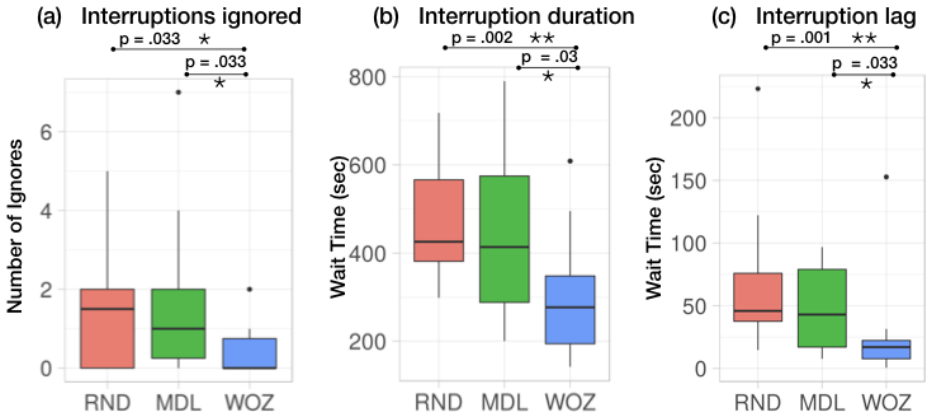


Figure 3.15: Data and analysis for results in Section 3.10.3.

RND and MDL according to the predictions of the Goal-Activation model [105].

In conclusion, we find that all three of our conditions achieved similar task throughput, suggesting our participants maximized their potential throughput in our manufacturing environment. However, the maximization came at the cost of robot tasks being ignored in the interruptibility-unaware condition of RND. In fact, we find that the addition of interruptibility-aware behaviour (WOZ in particular) greatly improved the efficiency of the robot, particularly with a reduction in the number of its tasks that were ignored. This is explored further in the next section.

3.10.3 Analysis of Robot Task Performance

In this section, we examine metrics relevant to answering the question, "How does interruptibility-aware robot behavior affect robot task performance when relying on humans for assistance?" (**RQ5**). In answering the question, we make a distinction between the time spent by the robot waiting at the observe point, and the time spent by the robot waiting in front of the participant's work table. We do not consider the observe time to be wasted time, as we assume that the robot might find an alternative interruption candidate during this time in a different environment.

Results:

Our conclusions are drawn from the number of interruptions that the robot presented (Figure 3.14b), the number of those that were ignored (Figure 3.15a), and the delays incurred by the robot by waiting on the human after it requested assistance. For the last metrics, we only present analyses on interruptions initiated during a build[19]. Our analyses use a Kruskal-Wallis test on study condition followed by post-hoc pairwise Wilcoxon rank-sum tests with Benjamini & Hochberg correction.

The interruption duration is unproductive robot time spent waiting on the human's assistance, and is therefore a measure of low productivity. We hypothesize that poorly timed interruptions result in a longer interruption duration, and therefore more time wasted by a robot that needs assistance. For the 14 trials in each condition, the data reveals a significant difference between the study conditions ($H(2, N = 42) = 10.8, p = .0046$), with a shorter total interruption duration in WOZ ($Mdn = 277$) than in MDL ($Mdn = 414$, Post-hoc Wilcoxon, $p = .032$) or in RND ($Mdn = 426$, Post-hoc Wilcoxon, $p = .0024$) (Figure 3.15b).

The interruption lag is another metric of how long the robot had to wait on participants, and is a better indicator of the effect of appropriate timing to the robot's task delay because it is not affected by a participant's capability to build, or by whether the interruption was ignored. As with interruption duration, higher interruption lag means more time wasted by a robot and lower efficiency. For the 42 trials, the data reveals a significant effect of study conditions ($H(2, N = 42) = 11.4, p = .0034$), with lower average lag in WOZ ($Mdn = 17.0$) than in MDL ($Mdn = 43.0$, Post-hoc Wilcoxon, $p = .033$) or in RND ($Mdn = 45.8$, Post-hoc Wilcoxon, $p = .0016$) (Figure 3.15c).

We observe a significant difference between our wizards in interruption lag, with participants showing lower lag with wizard **C** ($Mdn = 10.1$) than with wizard **A** ($Mdn = 22.7$).

[19]The difference between the study conditions is most apparent in such interruptions. Kruskal-Wallis tests on robot delay data from the interruptions when participants were observed idle show no significant effect of the study condition and show instead a significant effect of participant build experience.

Meanwhile, there is no significant difference between the wizards on the metric of interruption duration.

Summary:

Our results support our hypothesis that interruptibility awareness has a positive impact on robot task performance (**H3**). Not only is the robot able to accomplish the same amount of work with fewer requests for assistance, but well-timed interruptions also reduce the amount of time the robot has to wait on the participant to respond to its request, even when the interruptibility-awareness might not be perfect (as in MDL). In summary, well-timed interruptions allow a robot to operate more efficiently, completing tasks with fewer requests and in less time. In the next section, we evaluate participants' perception of such well-timed interruptions.

3.10.4 Analysis of Robot Impressions

Our results thus far show that the robot in the MDL and WOZ conditions succeeded in interrupting participants more appropriately, that this behavior did not have significant impact on participant task performance, but that it did improve robot task performance. Here, we evaluate our hypothesis that participants have a higher opinion of interruptibility-aware robots (**H4**) using participants' Likert scale responses to questions of interruption appropriateness and of robot timing (measures **M4**, Cronbach's $\alpha = 0.7$). For our analyses, we use a Kruskal-Wallis test on study condition followed by post-hoc pairwise Wilcoxon rank-sum tests with Benjamini & Hochberg correction. We also drop one of the 14 responses in the MDL condition because the participant spent less than 10 sec on the post-study questionnaire.

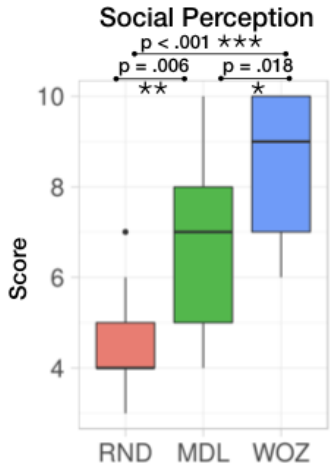


Figure 3.16: Data and analysis for results in Section 3.10.4.

Results:

For the 14 trials in each condition, the data reveals a significant difference ($H(2, N = 42) = 21.1, p = 2.6e^{-5}$) in the scores of social perception between all three conditions, with participants rating WOZ ($Mdn = 9$) the highest ($p = .018$), followed by MDL ($Mdn = 7$, $p = .0062$), followed by RND ($Mdn = 4$). We did not observe any difference between our two wizards on the score (Figure 3.16).

Summary:

Our results support our hypothesis; participants had a higher opinion of the robot in WOZ than in MDL, and a higher opinion of the robot in MDL than in RND. Post-study conversations with participants revealed interesting directions for future research on the social perceptions of interruptibility-aware robots.

We found that participants were not always objective regarding the appropriateness of the interruption timing (Q1); perhaps as as result of the relatively short time participants had with the robot and the overall novelty of the robot interaction. A large portion of participants factored in considerations of whether they thought they could finish a main build when the robot interrupted, whether they needed a break from the main build, or whether interacting with the robot was just more fun than working. Participants were also prone

to misremembering their experience, with notable examples where one participant did not remember experiencing any interruptions in the middle of a build and another participant recalled a mistimed interruption in WOZ with wizard **C**, despite contrary evidence in the video.

Additionally, we found that perception of the robot's workload awareness and considerateness (Q2) resulted in part from a different overall assessment of the robot's nonverbal behavior. Several participants in the INT condition noted (1) the robot's proclivity to wait when they were building, (2) its ability to approach immediately when they were free, (3) the robot's willingness to wait silently in front of the table if they were busy, and (4) the robot's head motion, all as evidence of the robot's intelligence. Note that only (1) and (2) differ across our study conditions, while the wait behavior (3) and the head motion (4) are identical in all study conditions. However, none of the RND or WOZ participants attributed any importance to (3) or (4). Instead, participants in WOZ and RND reflected on the difficulty of the builds that were interrupted, and the appropriate or inappropriate (respectively) time of the robot's approach. These responses echo prior robotics research [76] and highlight the potential of the interruption behaviors in ameliorating mistakes in interruptibility classification; thereby presenting avenues for further interaction research for interruption management with embodied robots that interrupt humans.

3.10.5 Conclusion

In conclusion, our results supporting **H1** show that the interruptibility-aware system we developed is effective at predicting interruptibility at high accuracy, and that using it our robot interrupts at more appropriate times than a robot without interruptibility awareness. The results further validate that developing interruptibility-aware robotic systems is important to future deployments of interactive autonomous systems. We find that human performance of skill-based tasks is not affected by interruptions (**H2**), primarily because participants effectively regulate their workload by ignoring the robot when too many tasks are given.

Critically, however, interruptibility-aware behavior improves metrics associated with robot task performance (**H3**) by reducing the robot's time wasted on inappropriate interruptions. Finally, interruptibility-aware behavior improves humans' perceptions of the robot's social aptitude (**H4**).

3.11 Insights

In this chapter, we have described the first fielded mobile robotic system that classified human interruptibility online based on social and contextual cues and without reliance on external sensors. In developing the system, we found that the social signals of a person's interruptibility can be usefully augmented with the contextual cues to their interruptibility such as the objects they're using. We also found that temporal models, such as our LDCRF, proved to be more appropriate than non-temporal models, such as our MLP and RF, for online classification on a robot. We then evaluated our system in a user study and we found that developing interruptibility-aware systems to modulate when to interrupt collocated bystanders might not affect human performance, but they can reduce the duration of a robot failure by reducing the lag between the failure and the human's resolution of it.

However, the total *duration* of an intervention is dependent on more than just the interruption lag from the interruption: it is dependent on additional factors such as the human's understanding of *what went wrong* and their knowledge of *how to fix it* [25]. In the subsequent chapters of this dissertation, we examine methods of improving a human operator's understanding of a robot failure with the goal of reducing the total duration of the intervention after an interruption might have occurred. We focus our investigation on remote robot operators, who are assumed to be always interruptible.

CHAPTER 4
ON THE EFFECTS OF PROVIDING DECISION SUPPORT TO REMOTE OPERATORS

As autonomy improves, robots are increasingly operating without close expert supervision. Robots making hospital deliveries, taking inventory at grocery stores, or organizing a warehouse operate largely independently but occasionally encounter an error. In such cases, it is unlikely that a local robotics expert will be available, and the robot will instead rely on remote *call center of operators* for assistance [27, 38]. Such a request, even if expected, challenges operators with too much information, the complexity of the situation, or the urgency of the intervention: as such it constitutes a difficult information processing problem [129]. Therefore, operators can benefit from decision support to help them in their decision making during an intervention. In this chapter, we provide remote teleoperators with action and diagnosis suggestions from pretrained decision support models and investigate the effects of the interaction on the operators and on the models.

From the prior work, it is unclear (1) what effect providing suggestions to operators during an intervention might have on their performance, and (2) what effect operator behaviour might have on a model that provides them with suggestions. Concretely, while prior work has found increased operator accuracy and compliance when provided decision support [130, 131, 132], other work has found an equivocal effect of feedback on human performance, since the performance is often mediated by expertise, personality, motivation, etc. [133]. Therefore, although troubleshooting aids have been developed in the past [134, 135, 136], it is unclear from the prior work whether these actually help robot operators during an intervention. Similarly, human operators are often asked to help recover robot autonomy because they are assumed to have additional information unavailable to the robot [26], which can also aid a decision support model in its suggestions. However, humans are liable

to get confused and act incorrectly [137], even if they're experts [36], and even if they have access to decision support [138]. As a result, there exists a possible tradeoff between the additional information available to a decision support model from the human, and a threat to the model's accuracy as a result of human errors.

We therefore present the results of a user study that we conducted to investigate the above effects. In particular, we focus on an operators' MTCI, which measures robot reliability [31], and on the accuracy of the suggestions of the decision support models. We recruited experienced roboticists as operators in the study and the participants were required to assist a robot during errors in pick-and-place tasks. We also trained diagnosis and action suggestions models on a dataset of interventions handled by two experts, one of them, the experimenter. The suggestions from the models were shown to the participants based on the study condition they were in. During the experiment, participants were asked to think-aloud their decision-making and the data from the think-aloud, combined with automatic annotations from the operator User Interfaces (UIs), were used to report on operator workflows during an intervention.

We gain the following insights from the results of our user study: (1) operator performance is likely improved by decision support, but the benefits might be mediated by the type of decision support and its inaccuracies, (2) decision support models can suffer from greatly degraded accuracy as a result of operator actions, and (3) operators do not distinguish between diagnosis and recovery, indicating that both types of decision support in our study must be provided at all times. The insights greatly inform our investigations in the subsequent chapters of this dissertation.

4.1 Related Works

A robot intervention is an information processing task in which operators might be provided with too much information that overwhelms them, or they might lack enough information to gather sufficient situational awareness [129]. Prior work has found that when faced with

Figure 4.1: During an intervention, an operator's behaviour is influenced by the decision support from the robot, and the decision support is affected by the nuances of operator behaviour. We investigate the interactions in this work.

a robot failure, operators often ask, "What is wrong?" and "How do I fix it?" [25]. Decision support systems have proven useful in such troubleshooting settings where operators have to sift through a lot of information, the task is complex, or the decision needs to be made quickly [135, 139, 136]. In this work, we evaluate the effect of providing operators with the decision support to better answer the two questions they frequently raise.

Answering "what is wrong" requires a problem diagnosis, which has been the focus of much prior work in robotics [28]. Diagnoses can be provided using pre-defined models and expert-knowledge of the system [55, 24], or using the correlations between symptoms and diagnoses observed in a dataset [140, 141, 142]. While model-based diagnosis systems are capable of providing detailed diagnoses, they can be particularly brittle if the model is misspecified, which is a concern in a complex system such as a robot [28]. We therefore eschew model specification in favour of data-driven approaches to diagnosis in this work. Additionally, prior works in robot fault diagnosis often assume that failure recovery is a single-step update, or at most a trivial sequence of steps, once the diagnosis is determined [55, 24, 141, 142]. However, recovering from failure requires *troubleshooting*, in which diagnosis hypotheses can be updated with new evidence over time, and the recovery options can change as a result of the updates [143, 135, 144, 145, 136]. We therefore investigate methods to provide diagnoses throughout an intervention, and create a model to update diagnoses based on the actions taken by the operator.

Action recommendations can answer the second question of "how do I fix it". Prior work in mixed-initiative planning [11], learning from demonstration [146, 147, 148], and scheduling [149, 150] are designed to provide such answers. In fact, research in troubleshooting has produced systems that are capable of providing *both* diagnosis and action recommendations [143, 135, 144, 145, 136]. However, some of the action recommendation systems and all of the troubleshooting systems require the specification of domain, environment, or failure models, which increases developer burden and can lead to inaccurate suggestions if the model is misspecified. Therefore, we use a simple data-driven approach, Behaviour Cloning [146], to learn action recommendations from a dataset of interventions.

Unfortunately, the process followed by operators during an intervention is not formalized, and as such, determining the moments at which to provide decision support—every time an operator acts, only when certain actions occur, etc.—or what to provide at each moment—diagnosis and action suggestions at all times, diagnosis suggestions first followed by actions, etc.—is unclear. Some works avoid the problem by providing decision support only when requested by an operator [150], which can be appropriate in some settings, but which shifts the burden of requesting support to the operator and can therefore be undesirable for its reactive nature [137]. Other works, particularly in the realm of fault diagnosis [141, 142], consider a simple two-step approach to an intervention: diagnosis followed by repair (Figure 4.2a). The approach has been extended by other works depicting flowcharts to show cyclic processes of diagnosis and repair until completion [151, 152] (Figure 4.2b). While the simpler process does not capture the troubleshooting observed in automated systems [144, 145], the cyclic processes have never been validated on any system (robot or otherwise) [151], or incorporate assumptions, such as perfect observability, which are not acceptable in robotics [152]. In this work, we show that a human conducts diagnosis and repair concurrently, with one influencing the other (Figure 4.2c). The finding affects when and what decision support should be provided to the operator.

Finally, few works study the challenges related to human-robot interaction during an

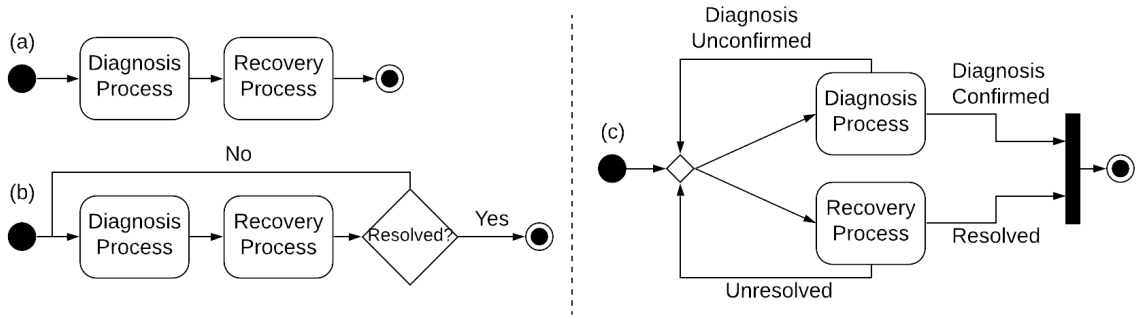


Figure 4.2: (a) The most common approach to viewing human actions during an intervention—the human confirms a diagnosis of problem(s) and then takes actions to resolve them. (b) The process can be extended such that diagnoses are confirmed and actions are taken to resolve the problem(s) until all problems are resolved. (c) In our work, we find that diagnosing problems and resolving them are parallel processes that can influence each other (a detailed diagram is in Figure 4.6).

intervention [129]. In particular, humans are prone to making mistakes [137], and as such decision support models must remain robust to their errors. However, a human operator is used to help recover robot autonomy *because* humans are assumed to possess additional information that is useful to the recovery process [26]: a decision support can therefore leverage the information for increased accuracy. In this work, we begin to examine the effects of operator behaviour on the performance of decision support models.

4.2 Research Questions

In this section, we introduce our investigation into the results of providing decision support to human operators in an effort to reduce the time they spend on the intervention, their Mean Time Completing Interventions (MTCI). In addition, we also investigate the effect of operator behaviour on the accuracy of decision support models. Concretely, we investigate the following research questions:

RQ1 *What is the effect of decision support on operator performance during interventions?* We are particularly interested in the effect of the type of decision support provided—diagnosis vs. diagnosis and action suggestions—and its effect on an operator's MTCI.

RQ2 *What is the effect of operator behaviour on the quality of the decision support?* We are particularly interested in investigating if an operator's behaviour, particularly their actions, can improve or reduce the accuracy of a decision support model.

We also examine the workflow of operators when they are required to intervene on behalf of the robot. Prior works often assume the existence of two primary processes followed by operators during an intervention (see Section 4.1). We redefine the processes as:

1. **Diagnosis Process**: the process of finding the root cause error that caused the robot intervention to occur. The root cause error is that error, which when removed, would allow the robot to resume its autonomy [153, 154]. The diagnosis process ends when the existence of the root cause is confirmed; the error need not be rectified.

2. **Recovery Process**: the process of rectifying the root cause error that caused the intervention. The rectification of the error can occur even if the error itself remains unknown or unconfirmed.

In this work, we investigate how an operator conducts the diagnosis and recovery processes, and how they transition from one process to the other or vice-versa.

4.3 User Study

We conducted an IRB-approved three condition between-subjects study to investigate our research questions above. The study conditions were:

NONE Participants were provided with no suggestions during a robot failure. They had to determine the cause of the failure and rectify it using their knowledge of robots and information available to them through a provided user interface (UI). This was a baseline condition.

DX An ensemble of Hierarchical Dirichlet Process-Hidden Markov Models (HDP-HMMs) from prior work [142] was used to provide *diagnosis suggestions* to participants after every action they commanded.

DXAX An RNN-based filter was added to the output of the diagnosis model to explicitly capture dependence between an operator's actions and the decision support's outputs. The filter provided both *diagnosis and action suggestions* based on the HDP-HMM's diagnoses and the operator's chosen action.

From *NONE* to *DX* to *DXAX*, the amount of decision support provided to operators increased, and the reliance of the decision support output on the human's actions increased. In the following paragraphs, we provide details on the study procedure. In Section 4.4, we provide details on the decision support models.

Environment

A Fetch robot was placed in the FetchIt! challenge arena[1], which was a 10ft x 10ft area with five tables containing upto five different objects. Participants were shown the arena upon arrival but they were required to be present at a nearby workstation, which was not in the arena, during the study. Participants could not view the robot or the arena from their workstation and industrial background noise[2] was played in the environment to drown out the actuator noise from robot movements. An experimenter was also present near both the workstation and the robot arena to (1) inject failures according to the experiment scenario, (2) annotate details about interventions or the participant behaviour, (3) assist the participant with choosing how to end an intervention, (4) assist the participant with technical problems or logistical questions, and (5) to E-stop the robot in case of safety concerns.

[1] Described in Chapter 2

[2] https://youtu.be/ylD4EmcxB6o

Robot System and Tasks

The experiment used a Fetch mobile manipulator [63], which has a differential drive base with a 2D laser scanner for navigation, and an RGB-D camera alongside a 7-DOF arm with a parallel-jaw gripper for manipulation. The robot's sensors and actuators were orchestrated through the system described in Chapter 2. In the system, faults were detected when a primitive action failed, either through unmet preconditions or by the detection of unintended effects. While the system in Chapter 2 was designed to diagnose and automatically recover from detected faults, in this work, we disabled all automatic diagnosis and recovery routines. Instead, when a fault was encountered, the system presented a UI that participants could use to help the robot recover autonomy.

All of the robot's tasks in the FetchIt! arena required it to navigate to one of the five locations, pick up an object, and in most cases, put it in a kit on its base. In the case of an object called the 'large gear', the robot had to insert the gear into a faux machine in the arena, wait 2 minutes, and then extract it before putting the gear in its kit[3]. Participants were made aware of the special requirement for the large gear.

Participant Workstation

The participant's workstation had two monitors, a microphone, a keyboard, and a mouse. By default, the workstation displayed a study home webpage[4] with links to:

1. a Web UI that participants could use during a robot failure,
2. a Tutorial presentation introducing the participant to the robot, the environment, the robot's tasks, and the Web UI,
3. a Manual webpage that the participant could reference during the study for more detailed information related to the robot, the environment, the robot's tasks, and the

[3]Examples of the robot performing tasks are at https://youtu.be/j2w1FffgD3s and https://youtu.be/osNtFxqC1UM

[4]http://bit.ly/3pINzsx

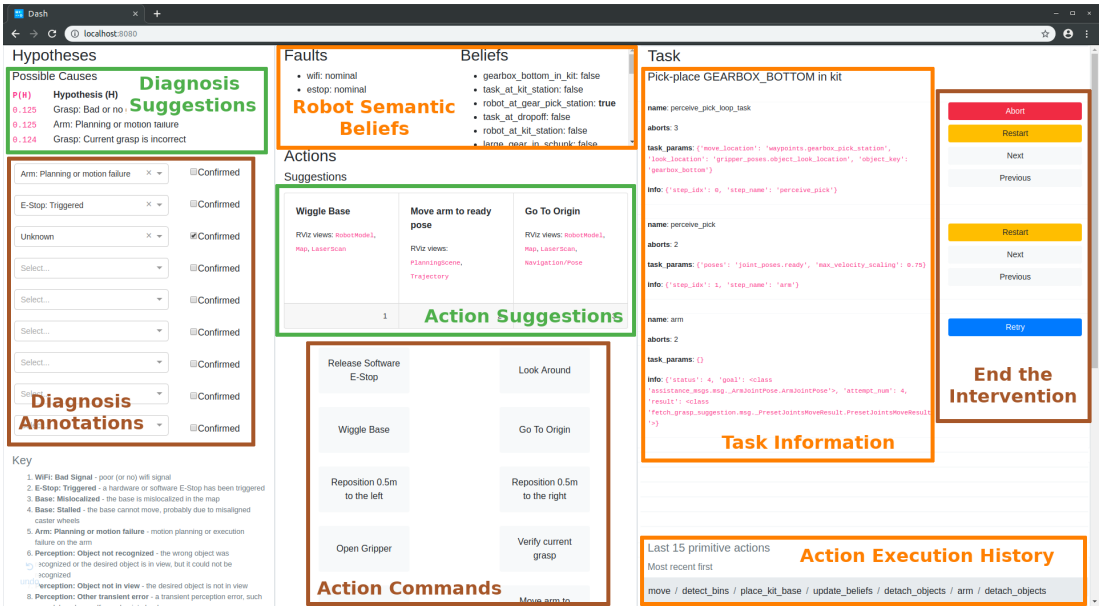


Figure 4.3: The Web UI provided to operators. The suggestions sections were shown or hidden based on the study condition.

Web UI,

4. three questionnaires to be completed over the course of the study,

5. three web-based idle games [155] that the participant was required to engage in when not occupied by a failure intervention.

Participants were not allowed to navigate to web pages not linked to the home webpage.

During a robot failure intervention, the workstation revealed an RViz interface [56], otherwise inaccessible, with displays from the robots sensors, which the participant could use in conjunction with the Web UI to assist the robot. Participants were encouraged to think-aloud [156] their decision-making while using the either RViz or the Web UI and the think-aloud was recorded using the workstation's microphone.

Web User Interface

The Web UI (Figure 4.3) complemented the RViz UI in helping operators assist the robot. In particular, during an intervention, the Web UI provided:

1. the semantic beliefs of the robot, such as 'the robot is near table 3', 'the E-stop is pressed', etc.
2. the goals of the task and the state of the task at the moment of the intervention
3. a history of the actions attempted by the robot during the task
4. buttons for the operator to command one of 15 high-level actions for the robot to perform[5]
5. dropdown menus for the operator to annotate one or more of 14 plausible problem diagnoses (including the diagnosis of 'unknown') that they were attempting to address
6. buttons for the operator to select a method of resuming the task, or abort it, once they deemed the intervention complete
7. three diagnosis and three action suggestions for the operator, which were shown or hidden depending on the study condition

Information in the UI was only populated upon an intervention start and the UI was disabled when a participant selected a button to end the intervention.

The dropdown menus to annotate problem diagnoses were added to the UI in order to obtain a better understanding of operator workflow. The annotations captured participants' goal(s) when taking actions during the intervention and as such provided a rudimentary cognitive walkthrough [157] of their reasoning process when paired with the think-aloud audio. In order to facilitate the annotation process, participants had to 'suspect' at least one problem hypothesis for the intervention before they could take any actions, and they had to 'confirm' at least one problem hypothesis before they could end the intervention. Participants could provide or rescind the hypotheses and confirmations at any moment during the intervention.

During an intervention, a participant *interaction* was defined as an event where the participant updated problem hypotheses or commanded the robot to take an action. As such, the stream of interactions defined an intervention workflow and any given interaction

[5]We chose to provide high-level actions based on recommendations of prior work [36].

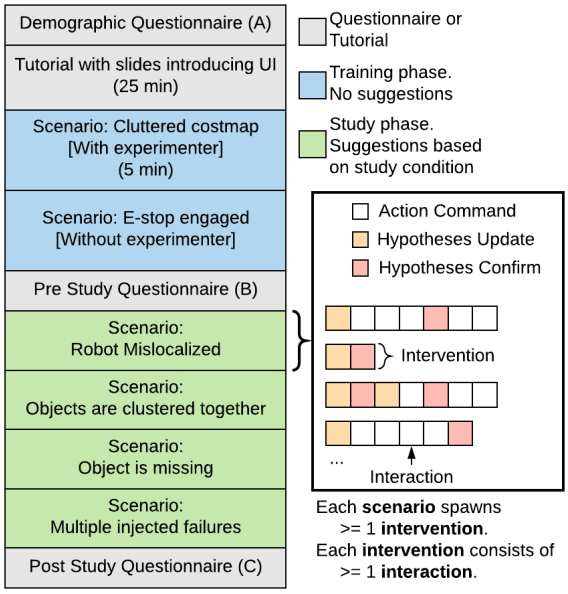


Figure 4.4: Study Timeline. Each scenario is designed to spawn at least one intervention corresponding to an injected error. Additional interventions could manifest due to operator behaviour or other external factors.

was a candidate for updating the decision support provided to the operator. Since updates to the problem hypotheses were simply annotations that did not alter the robot or environment state, diagnosis and action suggestions were *both* updated after every action interaction. The decision to update both types of decision support after actions was validated by an analysis of operator workflow, as discussed in Section 4.5.

Timeline

The study was segmented into a *training phase* of about 30–35 minutes, which familiarized participants with the study, and a *study phase*, the data from which was used to investigate our research questions (Figure 4.4). Each phase consisted of multiple *scenarios*: during each scenario, the robot was given a single task requiring the pick-and-place of one or more objects in the environment, while the experimenter injected one or more predetermined faults in order to induce at least one intervention. Other faults could arise due to the uncontrolled environmental factors or participant behaviour, thereby leading to additional interventions. A scenario was deemed complete if the robot completed the task after all

interventions or if the participant chose to abort the task through the Web UI. The training phase consisted of two scenarios, each designed to induce at least one intervention, and the study phase consisted of four scenarios, with the first three designed to induce at least one intervention each, and the last scenario designed to induce at least three interventions.

Participants were asked to sign a consent form and complete a demographic questionnaire upon entering the study environment. They were then shown the robot's environment and seated at their workstation. The experimenter introduced participants to the study using a 15 minute tutorial presentation linked to from the study home page. Participants were then given 10 minutes to further familiarize themselves with elements of the study. Once ready, the training phase began with the first training scenario, in which the experimenter provided a walk-through of the tools available to the participant during the intervention. The experimenter walk-through was absent in the second scenario of the training phase. During both training phase scenarios, no suggestions were provided in the Web UI. The training phase concluded with the participant answering a pre-study questionnaire on their experience.

During each scenario in the study phase, participants were shown diagnosis and action suggestions in the Web UI depending on the study condition they were in. Although participants could see the particulars of the pick and place task that the robot was performing in the study phase, they were not told when one scenario was completed and another started. Due to the length of the study, the experiment was paused, i.e. robot execution halted, between interventions if the participant required a break. When all scenarios were complete, participants were asked to answer a post-study questionnaire on their experience.

Robot Errors

The scenarios in the study simulated a broad range of robot failures arising from internal and external failures to different components in the robot's task system. The two training phase and four study phase scenarios were scheduled such that the difficulty or complexity

of the predetermined errors increased from one to the next:

1. *Cluttered costmap (training)*: the robot's manipulation costmap was injected with clutter in order to induce a motion-planning failure from the arm. The scenario was used to familiarize participants with RViz.
2. *E-stop engaged (training)*: a software E-stop was engaged, thereby preventing the robot from moving its actuators. The status of the E-stop is not checked as a pre-condition in the robot's task but it is monitored by background system monitors and therefore displayed on the Web UI. The scenario was designed to familiarize participants with the Web UI.
3. *Robot Mislocalized*: the robot's localization module was poorly initialized, which led the robot to navigate to incorrect locations in the environment. An intervention was triggered when the robot failed to find its object-to-pick, because it was at an incorrect table, and participants had to debug the original cause of error from this symptom.
4. *Objects are clustered*: the object-to-pick was clustered together at the center of the table, leading to a failure in perception if the cluster of objects was not recognized (common) or a failure in manipulation if an individual object from the cluster could not be picked up (uncommon). Participants had to debug the cause of the perception or manipulation failure.
5. *Object is missing*: the object-to-pick was removed from the environment entirely. The scenario was designed to simulate realistic situations where the tools available to an operator during an intervention prove to be insufficient. Hence participants were required to abort the robot's task.
6. *Multiple failures*: three failures were injected at pre-determined moments in the robot's task: (1) the large gear was placed on the table so that when picked, its pose in the robot's gripper would not allow it to be inserted into the faux machine, (2) another object, the 'small gear', was placed out of the field-of-view of the robot's camera,

and (3) the large gear was induced to 'fall' out of the machine after the first time it was inserted by the robot. Participants had to address each problem when the related interventions surfaced.

In addition to the predetermined errors, additional uncontrolled errors were also encountered: (1) participant mistakes could produce a snowball-effect of errors, and (2) idiosyncracies of the environment and robot could also surface errors. Examples of idiosyncratic errors included a 'stalled robot base' caused by the wheels of the robot sticking to the carpeted floor of the experiment area, 'intermittent phantom obstacles' cluttering the robot's manipulation costmap, etc.

Intervention Protocol

During the study, participants were required to engage with at least one of the three idle games provided to them when they were not in an intervention. The participant was notified of an intervention when the RViz UI appeared on their workstations; at that moment, they were required to devote their full attention to the RViz and Web UI.

As participants interacted with the UI to recover the robot's autonomy, they were allowed to ask the experimenter for clarifications under the constraints that (1) the experimenter could only respond to Yes/No questions unless the question pertained to technical problems with the study apparatus, and (2) no questions that revealed the problem diagnosis or indicated a manner of recovery would be answered. When participants were done assisting the robot, they were asked to describe to the experimenter how they wanted the robot to resume its task, e.g. "Redo the entire task", "Try to find the gear again", etc. The experimenter would then suggest buttons for them to click in order to end the intervention. The final suggestion from the experimenter was necessary because dictating the manner of resuming robot autonomy required detailed knowledge of the task's implementation within the robot system: participants were not expected to learn such intricacies for the study.

The experimenter was also required to annotate details of the intervention or partic-

ipant behaviour. Given the uncontrolled causes for some of the robot interventions, the experimenter was provided with an administrative UI, not visible to the participant, which provided more details about the robot and system, and which could be used in order to annotate (1) the cause(s) of an intervention, (2) whether the participant identified the cause(s) based on their think-aloud, (3) notes on the reasons behind actions selected by participants based on their think-aloud, and (4) any other pertinent information that could be used to better understand participant behaviour, the system, or decision support outputs after the study. All data from the robot and participant interactions on the Web UI were saved to the robot's hard disk for such analysis.

Metrics & Hypotheses

We use the timeline of participant interactions in the Web UI to investigate how participants transition from the diagnosis process to the recovery process (defined in Section 4.2) and vice-versa. According to the definitions, the diagnosis process is deemed complete when participants confirm diagnoses on the Web UI and the recovery process is deemed complete after the last action interaction by participants in the Web UI. We hypothesized that there is no universal order to when participants transition from one process to the other.

In addition to operator workflows, we analyze the time participants spent completing interventions, since it is an indicator of system reliability [31] and a strong indicator of the benefits of decision support. We hypothesized that providing decision support to operators would reduce their Mean Time Completing Interventions (MTCI).

We also evaluate the accuracy of the model's diagnoses[6], using it as a quantitative metric for evaluating the effect of operator behaviour on decision support models. We hypothesized that incorporating a participant's actions for providing suggestions, as with the RNN filter in the DXAX condition, would improve the accuracy of the diagnosis suggestions as

[6]The uncontrolled causes for some robot interventions made it impossible to annotate the accuracy of participant actions or model action suggestions. Instead, we only had access to the experimenter's in-the-moment notes on the accuracy of a select few participant actions or model action suggestions.

more information is provided to the model.

Finally, our post-training phase and post-study phase questionnaires consisted of a 10 question survey that evaluated the following four factors using 5-point Likert scales[7]:

- Ease of making diagnoses: 4 questions, $\alpha = 0.76$
- Ease of recovering autonomy: 4 questions, $\alpha = 0.76$
- Helpfulness of the system: 3 questions, $\alpha = 0.71$
- Perception of time in interventions: 2 questions, $\alpha = 0.75$

We hypothesized that providing decision support would improve participant's scores on all of the above factors.

4.4 Decision Support Models

In this section, we formalize the problem of providing diagnosis and action decision support and then provide details on the models we implemented to provide the suggestions. The resulting pipeline is shown in Figure 4.5.

4.4.1 Problem Definition

We define two time-scales, denoted by t and τ, in our domain. t indexes participant action interactions (defined in Section 4.3) and as such denotes successive timesteps when action or diagnosis suggestions must be generated. τ indexes the robot's execution, and is therefore updated more frequently, since the robot may execute multiple actions as a result of a single participant action. We then define the following variables that are time-indexed in one the two time scales:

D_t**:** A random variable such that $P(D_t = d_i)$ captures the likelihood of a diagnosis $d_i \in \mathcal{D}$ at time t. $\mathcal{D} = \{d_1, \ldots, d_N\}$ is the set of N possible problem diagnoses. In our study, $N = 14$, including the diagnosis 'unknown'.

[7]The item reliability metric for each factor, Cronbach's α, was calculated using participant responses from the study. A value ≥ 0.7 is an acceptable level of reliability.

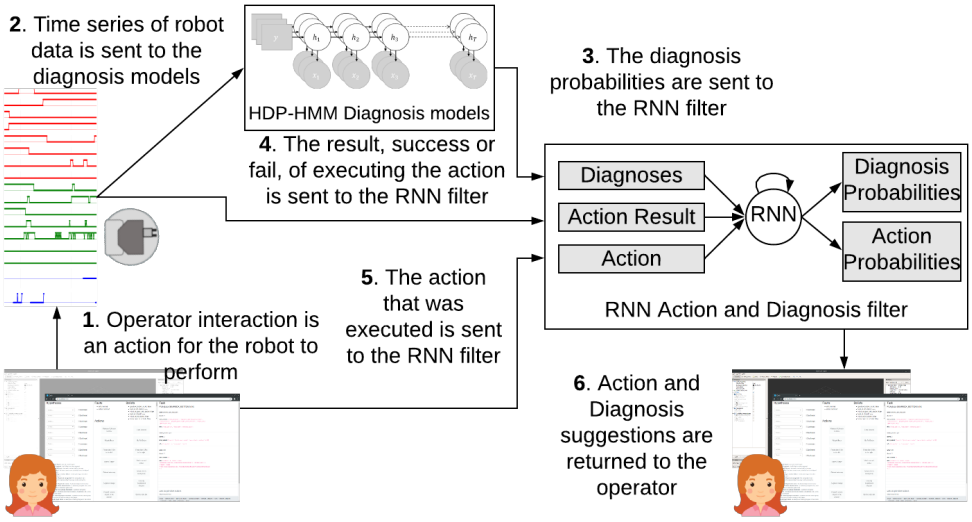


Figure 4.5: The pipeline for generating action and diagnosis suggestions after every action interaction in the *DXAX* study condition. Steps 3–5 do not occur in the *DX* study condition and as a result, participants receive diagnosis suggestions only.

A_t**:** A random variable such that $P(A_t = a_i)$ captures the likelihood that action $a_i \in \mathcal{A}$ is the most appropriate action, to either diagnose or resolve the failure, at time t. $\mathcal{A} = \{a_1, \ldots, a_M\}$ is the set of M possible actions operators can take, including that of ending the intervention. In our study, $M = 16$.

X_τ**:** Data from the robot at time τ. In our system, X_τ is a 86-dimensional vector with 61 dimensions indicating the state of the robot's task or attempted actions, and 25 dimensions capturing the semantic beliefs of the robot or the output from background fault monitors (see [158] for details). As mentioned above and with some abuse of notation, at time t in the intervention, we use a sequence of X_τ of length ω, i.e. $X_{t-\omega:t}$, to provide diagnosis or action suggestions.

Using the definitions, we formalize the problem of providing diagnosis suggestions as that of calculating $P(D_t|X_{t-\omega:t}, A_{t-1}, D_{t-1}, \ldots)$ and the problem of providing action suggestions as that of calculating $P(A_t|X_{t-\omega:t}, D_t, A_{t-1}, \ldots)$.

Unfortunately, the desired conditional distributions are hard to model, especially without a large dataset of operator interventions. Additionally, we are unable to label ground-

truth actions at any moment in an intervention because of the uncontrolled sources of error in our domain. We therefore make the following assumptions in order to make learning and inference tractable:

1. We assume a static fault diagnosis [159], common in the prior work, for the duration of an intervention[8]. Therefore, under a perfect model providing diagnosis suggestions, $P(D_t|\cdot) = P(D_{t-1}|\cdot) = \ldots$, i.e. the diagnosis suggestions for successive operator interactions are independent. With the assumption, we simplify the goal of providing diagnosis suggestions to that of calculating $P(D_t|X_{t-\omega:t}, A_{t-1})$.
2. We assume that given a diagnosis, operators need only follow a *recipe* of actions[9]; a practice assumed in some prior works [144, 141, 142]. Concretely, operator actions are conditionally independent from robot data given the diagnosis, i.e. $A_t \perp\!\!\!\perp X_{t-\omega:t}|D_t$ and therefore the goal of providing action suggestions is that of calculating $P(A_t|D_t, A_{t-1}, A_{t-2}, \ldots)$. Note that the history of actions helps identify the presumed action recipe.

Finally, we compare against prior work in robot fault diagnosis, which often does not use operator actions to update diagnoses during an intervention [142, 28]. In effect, the works assume that data from the robot is sufficient to determine a diagnosis, i.e., diagnosis suggestions are given by $P(D_t|X_{t-\omega:t})$. We therefore introduce the random variable D_t^P to model the diagnosis suggestions provided by a model from the literature and factor the task of providing diagnosis suggestions to the following:

$$P(D_t|X_{t-\omega:t}, A_{t-1}) = \sum_{D_t^P} P(D_t|D_t^P, A_{t-1})\, P(D_t^P|X_{t-\omega:t})$$

Note that we also assume $D_t \perp\!\!\!\perp X_{t-\omega:t}|D_t^P$ for simplicity, but the assumption can be further investigated in future work.

[8] An alternate phrasing: the underlying causes of an intervention do not change during the intervention.
[9] In some cases, the recipe can be a single action.

In the following sections, we provide details on the models used to learn each of the above probability distributions from prior intervention data, and then infer the same probability distributions online during interventions in the user study.

4.4.2 Baseline Diagnosis Models

We use an ensemble of Hierarchical Dirichlet Process-Hidden Markov Models (HDP-HMMs) from prior work [142] as our baseline to model $P(D_t^P|X_{t-\omega:t})$ because the models achieved very high accuracy in robot fault diagnosis from time series data. Specifically, for each diagnosis $d_i \in \mathcal{D}$, we train the parameters, $\theta^{(d_i)}$, of an HDP-HMM model from hierarchical Dirichlet priors using memoized variational inference in order to maximize the likelihood of $P(X_{\cdot}^{(d_i)}; \theta^{(d_i)})$, where $X_{\cdot}^{(d_i)}$ is a time series of robot data with the label d_i in a training dataset (see [142, 160] for details). At test time, the output diagnosis from the ensemble is then the diagnosis associated with the HDP-HMM model with the highest likelihood of observed robot data, i.e. $D_t^P = \arg\max_d P(X_{t-\omega:t}; \theta^{(d)})$, and the probability distribution of D_t^P is simply the distribution of likelihood values from the ensemble.

4.4.3 Diagnosis and Action Filter

We use a Recurrent Neural Network (RNN) with two outputs to filter the output of the baseline diagnosis models and calculate the two remaining probability distributions, $P(D_t|D_t^P, A_{t-1})$ and $P(A_t|D_t, A_{t-1}, \ldots)$. The recurrent operation, carried out by a Gated Recurrent Unit (GRU) [161] with a hidden dimension of 11, conditions each of the diagnosis and action suggestion outputs on the history of inputs, primarily action interactions, to the filter. There are two outputs generated from the RNN, one for diagnosis and another for action probability distributions. The network was trained to optimize Cross-Entropy classification loss using Adam [116] with a learning rate of 0.001.

4.4.4 Training Datasets

The experimenter and one of the primary developers of the system in [40], who was also part of pilot studies for this work, provided a dataset of 118 interventions that were used to train the RNN filter described above. The actions of these experts were assumed to constitute the action recipes that were the target action suggestion outputs of the filter. The hyperparameters of the filter—hidden dimension, number of layers, structure, etc.—were determined using a Hit@3 accuracy metric for diagnosis and action suggestion[10] under grouped three-fold cross-validation using action-labels as the grouping variable.

The experimenter annotated diagnoses from 81 additional interventions in the *NONE* condition of the user study were added to the previous dataset to train the baseline HDP-HMM diagnosis models. The latter data could not be used to train the filter because the sequences of actions taken by participants in the *NONE* condition were not desirable action recipes. The hyperparameters of the baseline models—Dirichlet priors, initial covariance, etc.—were determined using a Hit@3 accuracy metric for diagnosis suggestion [10] under grouped five-fold cross-validation using diagnosis-labels as the grouping variable.

4.5 Results

In this section, we present the results of our user study.

Participants

Due to the technical nature of the domain, the operator tools, and the robot errors, the study was limited to students in a university Robotics program with at least one year of ROS experience. There were 17 participants across all conditions, 12 male and 5 female, aged 20–30, *Mdn* 24. The study session for one participant in the *NONE* had to be discarded due to an irrecoverable failure in the study's logging tools.

[10] We use the Hit@3 metric because three suggestions were provided to operators.

In order to illuminate potential effects of expertise with a system, participants were divided into two cohorts—(1) *Experts* were familiar with the system developed in [40] and helped develop significant portions of it, (2) *Novices* did not have such expertise. There were four male experts, two in the *NONE* condition and two in the *DXAX* condition. The 12 remaining novice participants were equally divided between the *NONE*, *DX*, and *DXAX* conditions.

Study sessions for all participants ranged from 2 hours (120 min) to 3.5 hours (210 min), *Mdn* 2.5 hours (150 min); participants were paid $5/half-hour. Over the course of the sessions, the robot had to be recovered from 261 interventions in the *study phase*. The following paragraphs present an analysis of the data from such interventions. Note that due to the small sample size of 16 participants, we do not conduct any statistical significance tests.

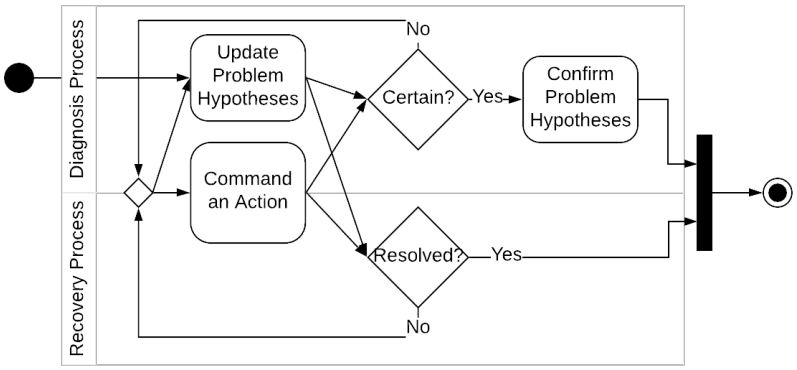


Figure 4.6: UML diagram of operator workflows in the study

Participant workflow

An analysis of the sequence of participant interactions in the Web UI across all conditions finds that in 230 (88%) of the 261 interventions, participants *confirmed* a problem diagnosis as their last interaction before ending the intervention; action interactions are the last interaction in only 31 (12%) of interventions. We converted the the sequences of interactions into transition diagrams, which were then combined into the Unified Modeling Language

(UML) flowchart shown in Figure 4.6. The result indicates that during an intervention, operators do not naturally separate the diagnosis process from the recovery process; instead both processes heavily inform and influence each other.

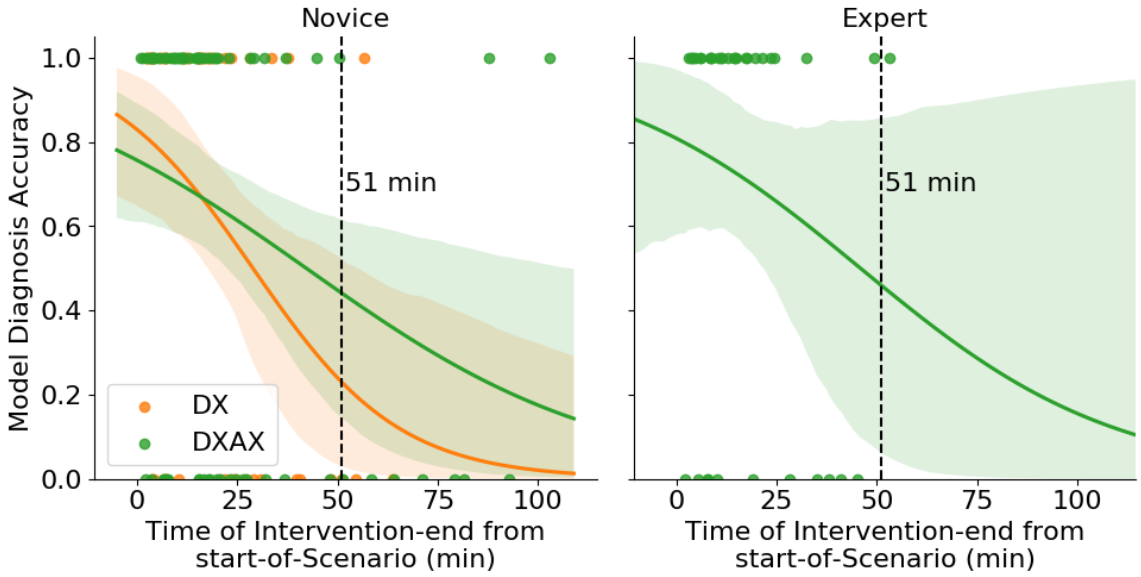


Figure 4.7: Model Diagnosis Accuracy vs. Time of Intervention-end from the start of a scenario. The line is a logistic regression fit to the data; shaded regions are bootstrapped 95% confidence interval of the fit. 51 min is the 95th percentile of end times across all conditions.

Model diagnosis accuracy degrades over time

As shown in Figure 4.7, the diagnosis accuracies of the baseline diagnosis models in the *DX* condition and the RNN filter in the *DXAX* condition are similar, and the accuracy of both degrade the later in a study scenario that an intervention occurs, i.e. the longer a study scenario, the less accurate the diagnosis from the decision support models [6]. There were two primary causes for scenarios taking a long time: (1) an unexpected number of idiosyncratic errors, such as the appearance of phantom obstacles in the manipulation costmap, occurred, and (2) a snowball-effect of incorrect participant actions as a result of confusion, misunderstanding, or other factors, led to additional errors. Regardless of the cause, the likely drift in the domain of the data provided to the models over the course of the scenario

resulted in a significant loss of accuracy, and hence usefulness of the decision support.

In the following paragraphs, we do not analyze 5% (14) of the interventions that manifested after 51 min from the start of a scenario because most of them were a result of uncontrolled errors[11].

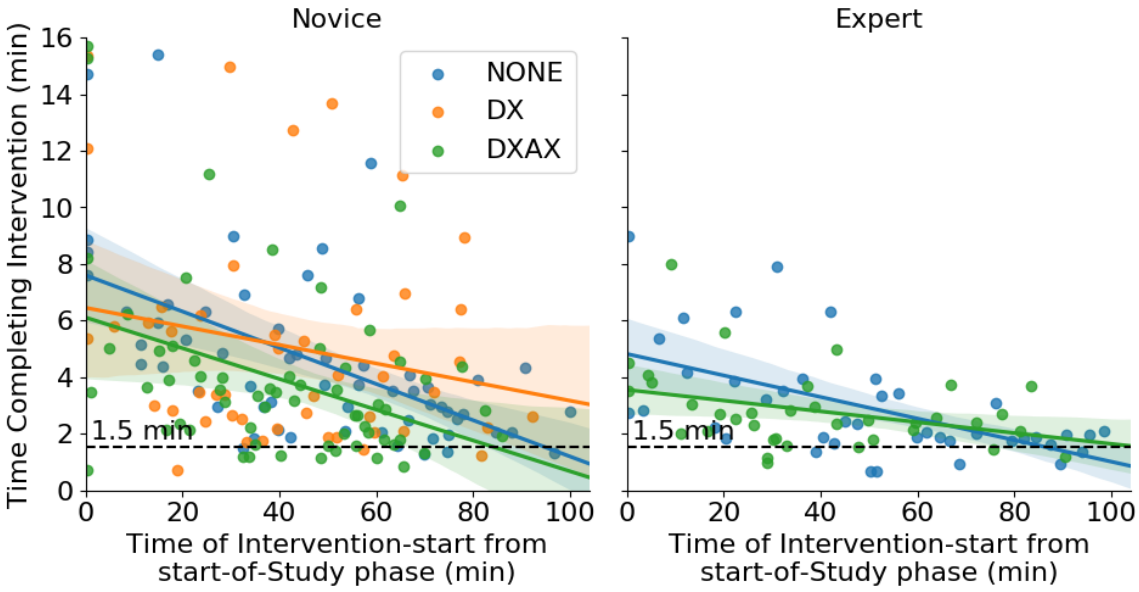


Figure 4.8: Time completing interventions vs. Time in the study-phase for interventions contained within 51 min from the start of a scenario. The lines are linear regression fit to the data; shaded regions are bootstrapped 95% confidence interval of the fit. 1.5 min is the MTCI of the experts used in the RNN training dataset and therefore it is a soft lower-bound on a participants' time completing interventions.

Learning effect

As shown in Figure 4.8, the mean time taken by participants to complete interventions (MTCI) decreases across all conditions the longer the participant is in the study. The result indicates a strong learning effect to operators' ability to help the robot recover as they get more familiar with the system. Additionally, there is a lower bound: once an operator is fully trained, as was the case for the operators used to generate action recipes in the RNN training dataset (Section 4.4.4), there is no further improvement to MTCI over time.

[11] 9 of the 14 discarded interventions are from a particularly idiosyncratic-error prone study session in the *DXAX* condition.

The effect of decision support

The lower Y-axis intercepts for novices in *DX* (6.45 min) and *DXAX* (6.11 min) compared to *NONE* (7.60 min) of the linear regression fit for MTCI in Figure 4.8 show that there is a noticeable reduction in MTCI with decision support when operators are not fully trained. The effect is apparent experts too with a lower Y-axis intercept for operators in *DXAX* (3.55 min) compared to those in *NONE* (4.82 min).

However, the slope of the regression lines indicate that the decision support might adversely affect learnability[12]. In particular, the lower magnitude of the slope for novices in *DX* (-0.032) compared to those in *NONE* (-0.064) or *DXAX* (-0.054) indicates a possible harm to learnability of diagnosis suggestions, which might be ameliorated by the additional inclusion of action suggestions in the *DXAX* condition. Since the slope of the regression line can also be a regression artefact of the lower bound to MTCI (discussed above), the phenomenon requires further investigation.

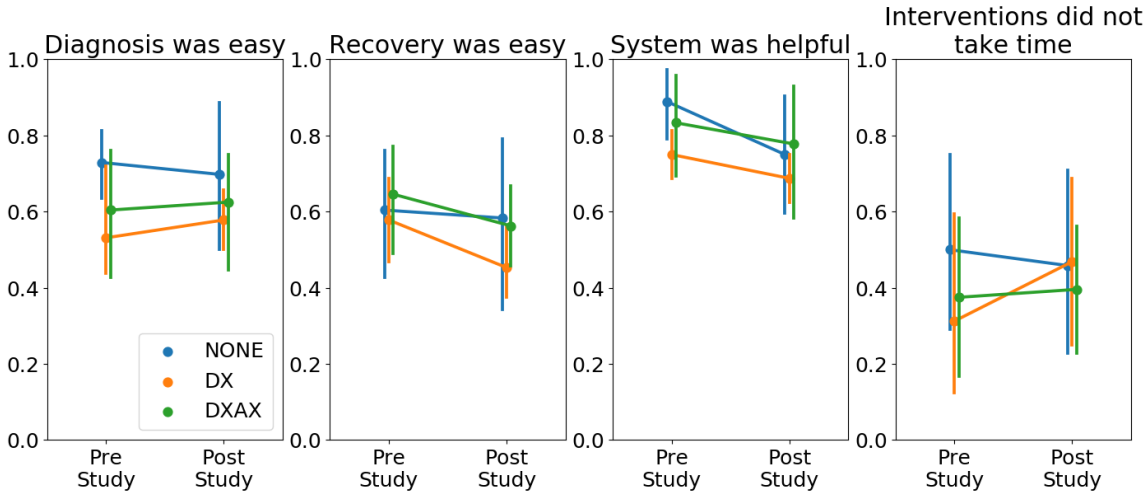


Figure 4.9: Participant responses to Survey Questions.

[12]The greater the magnitude of the slope, the greater the learnability.

Survey Results

There are no discernable trends in the effect of adding decision support to participants' survey results as shown in Figure 4.9. The (lack of an) effect can be partly explained by the frustrations of the uncontrolled interventions, which had a non-uniform effect on participant experiences across the study conditions. It can also be explained by issues of usability in the Web and RViz UI: most participants mentioned the 'lack of fine-grained control of the robot', the 'complexity and volume of information in the UI', and the 'lack of actual situation awareness' as hindrances in a free-form text response in the post-study questionnaire[13]. The same usability factors, more than the presence or absence of decision support, also determined what participants found helpful: some of the 'high-level operator actions', the 'variety of visualization options', and the 'awareness available from a camera feed or point cloud' were frequently cited as helpful in another free-form text response[14].

4.6 Discussion & Conclusions

From the results in the previous section, we find that:

Operator workflows do not distinguish between diagnosis and recovery Unlike the operator workflows that are assumed by some prior works [151, 152], operator workflows in our study did not distinguish between diagnosis and recovery processes. In fact, more often than not, operators seemed to *confirm* their diagnoses, i.e. complete the diagnosis process, upon completing the recovery process and finding no further fault. The finding implies that (1) the decision support provided to operators during an intervention must facilitate *both* processes, and (2) investigating the separate effects of decision support on the diagnosis and recovery processes requires harder constraints on operator workflow.

[13] Q: List three (or more) things that made it difficult for you to diagnose errors and recover from them.
[14] Q: List three (or more) things that made it easy for you to diagnose errors and recover from them.

Operator actions can harm decision support accuracy Contrary to our initial hypotheses, our results show that the snowball-effect of errors from incorrect actions taken by operators or the idiosyncratic errors that might have arisen from the vagaries of the study domain, led to less accurate diagnosis decision support. The result suggests that decision support models need to be made *more robust* to the whims of human inputs rather than mining them for additional help in classification. However, the finding needs to be replicated in a more controlled domain before a conclusive recommendation can be made.

Operator MTCI is most likely reduced with decision support, but the effects of each type of support are unclear Our results show that adding decision support at the beginning of the study phase, before operators are fully trained to the domain, reduces their MTCI. However, the effect of the decision support on the learnability of operators remains unclear: the noticeable difference in learnability for operators in the *DX* condition vs. those in *NONE* or *DXAX* is of particular interest. While the difference could indicate a learnability benefit of adding action suggestions to diagnosis suggestions, we do not have the data to make such a conclusion. Additionally, given the reduction in accuracy for diagnosis suggestions as discussed above, it is unclear if the supposed benefit to learnability is (1) a result of the type of the suggestions, (2) a result of possible higher accuracy in action suggestions[15], or (3) simply a coincidence. Further investigation is necessary.

Conclusions In this work, we discovered the existence of non-trivial interactions between the effects of decision support on human operators and the effects of human operator behaviour on decision support accuracy. Unfortunately, due to the uncontrolled nature of our domain, our inability to annotate the accuracy of action suggestions, the lack of participants to test an action suggestions-only condition, and issues of usability with the operator UI, we are unable to make definite statements regarding the nature of the interaction. Additional confounds not discussed above, further complicate the task. For instance, our assumption

[15]We could not unfortunately calculate the accuracy of action suggestions.

of a static diagnosis proved incorrect because incorrect operator actions during an intervention sometimes led to the introduction of additional problems that had to be diagnosed and recovered from in the same intervention. Participant's personalities also led to differing outcomes: some participants were skeptical of the suggestions and preferred to always double-check them, thereby increasing their MTCI when given suggestions, while other participants were overtrusting of the incorrect suggestions, resulting in lower MTCI but a greater overall number of interventions and a longer time taken in the scenarios.

Therefore, in the following chapters, we introduce a controlled domain in which ground-truth diagnoses *and* actions exist at all moments, and in which participants are presented with a simpler UI. Within this domain, we investigate the effect of the type and accuracy of decision support on the accuracy of remote operators in a large-scale user study in Chapter 5. We shift our focus from MTCI (time) to accuracy to account for situations when a lower time completing an intervention does not actually resolve the intervention (discussed in the previous paragraph). Then, in Chapter 6, we use data from the study to investigate the effect of varied operator accuracy on the accuracy of pre-trained decision support models.

CHAPTER 5
ACTION AND DIAGNOSIS RECOMMENDATIONS FOR RESPONDING TO ROBOT FAILURE

This work focuses on one of the interactions uncovered in the previous chapter: the effects of decision support provided to remote robot operators in their failure resolution interface. Prevalent guidelines for designing the user experience (UX) of a failure resolution interface suggest that operators should be provided with *feedback* information—to be made better aware of the failure state—and with *feedforward* information—to better enable decision-making [137]. However, it is unclear from the prior literature on UX [162, 163] which of feedback or feedforward information could be more useful to robot failure resolution.

The importance of knowing the relative benefits of feedforward and feedback is highlighted in Chapter 4 and by recent work, which has recommended that interfaces designed for error recovery not overwhelm the information processing capabilities of operators, lest operators themselves make mistakes [129]. Despite their usefulness, both feedback, such as through automated diagnoses, and feedforward, such as through action recommendations, can potentially add too much information to an interface. It is unclear from prior work if such is the case.

In addition to receiving too much information, an operator's information processing capabilities can be taxed in dealing with inaccuracies with decision aids: for instance, feedback provided through automated fault diagnosis systems or feedforward provided through action recommendation models can be imperfect, resulting in robots deployed with inaccurate decision aids (Chapter 4). Prior work has shown that in the face of incorrect suggestions, humans are prone to both follow the recommendations [164] and to ignore them [130]. It is therefore important to determine how inaccurate decision aids might affect the resolution of robot failures.

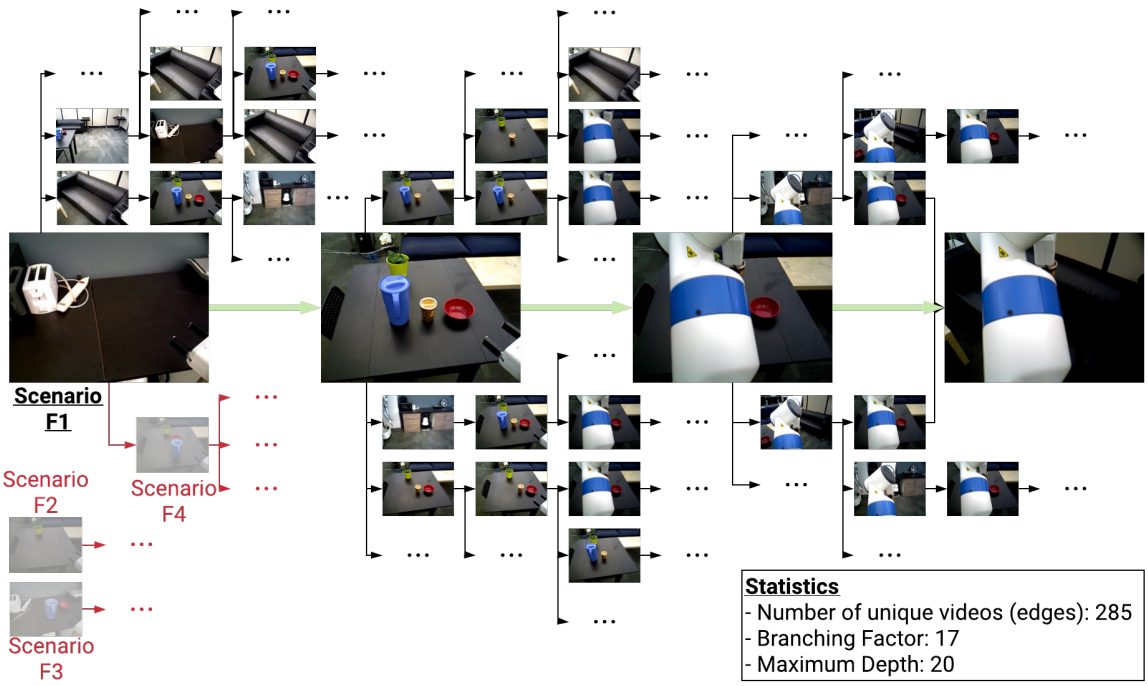


Figure 5.1: Storyboard for interactive failure recovery. Participants start in one of four failure scenarios and attempt to resolve the error by selecting one of 17 actions, which the robot then executes in an accompanying video. We evaluate the action sequence taken by participants under different interface conditions, and how it compares to the shortest possible error recovery (green arrows)[1].

In this work, we contribute a 10-condition study evaluating the effects of providing noisy and noise-free diagnosis suggestions (feedback) and/or action recommendations (feedforward) as decision aids to humans. We perform our analysis within an interactive user experience, in which users select robot recovery actions in response to observed error states. The interactive experience occurs in the context of a dynamically generated story graph in which nodes are faulty or fault-free robot states. The story nodes are connected by edges corresponding to one of 17 actions, captured as 285 videos of a physical Fetch robot, and which are selected by the participants (Figure 5.1). The resulting highly realistic evaluation framework enables us to examine how elements of UX design impact a user's ability to effectively recover from robot errors. Our findings show that although action recommendations (feedforward) have a greater effect on successful error resolution than diagnosis in-

[1] A video showing the study design is at https://youtu.be/drCHgwkpaqA

formation (feedback), the feedback likely helps ameliorate the deleterious effects of noise. Therefore, we find that error recovery interfaces should display both diagnosis and action recommendations for maximum effectiveness.

5.1 Related Work

Robust robot execution is difficult to achieve. To address this challenge, prior research has proposed techniques for adjustable autonomy in order to enable humans to assist during difficult tasks [24]. A previous ethnographic study of a robot in various work environments found that even when collocated, humans tasked with intervening on behalf of the robot wanted assistance addressing two fundamental questions, "What's wrong?" and "How do I fix it?" [25]. The finding is consistent with prevalent design guidelines that in the face of failure, operators should be provided with *feedback* information—to be made better aware of the failure state—and with *feedforward* information—to better enable decision-making [137].

Prior work has introduced decision aids to help operators capture high level problem diagnosis [24, 134], assist with action selection or planning [11, 147, 149], or both [165, 135]. Furthermore, failure recovery (i.e., troubleshooting) is often an iterative process during which failure hypotheses and/or recovery actions are pruned through execution of diagnostic test actions [145, 144]. Automated troubleshooting aids are usually equipped with the capability to suggest potential problems and to recommend actions to fix them [166, 165, 135, 143]. However, automated diagnosis or action recommender systems are not perfect [23, 36]. As a result, robots must often be deployed with imperfect decision aids.

Prior work has shown that in the face of errors in robot decision support systems, humans are prone to both *overtrust* its recommendations—following them to their detriment [164]—as well as mistrust its recommendations—ignoring them to their detriment [130]. It is therefore unclear what the effects of imperfect decision support systems might be in the case of robot failure recovery, a scenario not evaluated in the aforementioned studies.

We bridge that gap in this work.

Additionally, a recent survey calls into question a naïve recommendation to include both the feedback and feedforward decision aids because the authors find that failure recovery can be a cognitively demanding task for an operator and argue that the interfaces used must not overwhelm the information processing capabilities of a human [129]. They cite the prevailing design knowledge that humans are liable to themselves make errors if they are overwhelmed with too-much-information [163, 137]. Additional research in UX design has found that *feedforward* suggestions in widgets are especially suitable in applications that users might be unfamiliar with [162], but it is unclear whether the finding generalizes to robot error recovery, as well as when noise is present in feedforward output. We aim to identify the tradeoffs, if any, that might exist from providing either types of decision support to remote operators during robot failures.

Finally, while prior work has evaluated the interaction consequences of failure presentation, none evaluate the consequences on the robot's performance. Lee et al. [167] used online surveys to gauge participant evaluations of a robot's service based on the manner of robot communication during a failure; participants were not required to aid the robot based on the information presented to them. Similarly, Brooks et al. [168] introduced two types of decision support—human-support and task-support, which correspond to feedback and feedforward respectively—and found that people's reactions towards the robot were improved as a result of both types. However, their evaluations were conducted using survey responses to hypothetical scenarios and participants were not actually required to supervise a robot based on the information they received. In contrast, we allow people to supervise the execution of a robot during a failure, and we evaluate the robot's recovery outcome as a result of varying types of decision support.

5.2 Research Questions

In this work, we evaluate the relative benefits of both feedback and feedforward information by evaluating the following decision support for robot failures:

- **Diagnosis-based Suggestion**—the diagnosis of one or more faults (feedback). For example, "The [object] is not visible", (if a perception action fails) or "The robot has collided" (if the base of the robot is unable to move).
- **Action-based Recommendation**—the recommendation of actions to take to resolve the problem (feedforward). For example, "Navigate to [location]" (to perhaps check for a missing object), or "Move the robot back" (assuming the collision is at the front).

Despite several independent systems for diagnosis and recommendation having been developed (Section 5.1), their relative benefit when used either independently or together remains unexplored. Furthermore, automated diagnosis or recommendation systems have their own limitations, leading to imperfect performance [23, 36]. As a result, it is important that we understand how the relative benefits of both techniques are affected by their accuracy.

In this work, we study how various types of decision support aids, under varying levels of performance noise, affect the user's ability to effectively recover from errors. Specifically, we formulate the following research questions:

RQ1 *How is a human operator's assistance of a robot affected by Action Recommendations (AX), Diagnosis Suggestions (DX), or both (DXAX)?* We formulate this first question to investigate the types of decision support that might be necessary in a failure resolution UX.

RQ2 *What is the effect of inaccuracies in the decision support provided to human operators?* The aim of this second question is to investigate trends in human operator performance as the reliability of decision support varies.

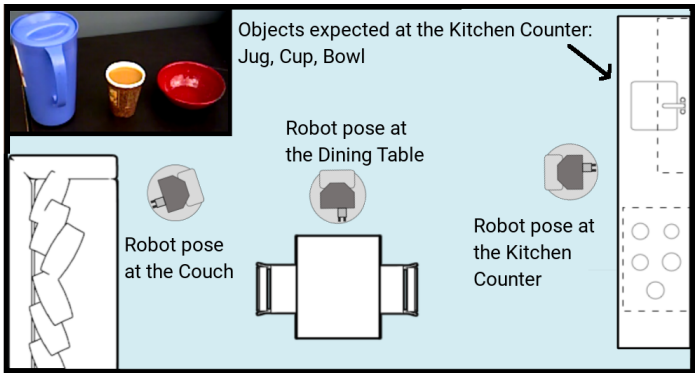


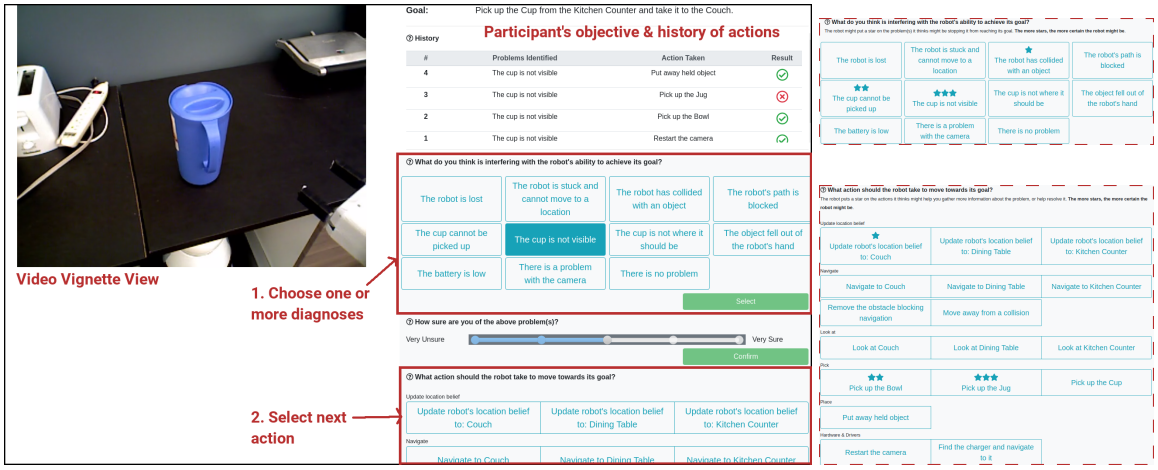
Figure 5.2: The robot is in a mock apartment with three locations. It can work with the Jug, Cup, and Bowl (left-to-right in inset).

To answer **RQ1**-**RQ2**, we designed a large scale user study using Amazon Mechanical Turk, in which we varied two factors determining the manner of generating decision aids. The first factor determined the type of suggestions that participants received—no suggestions (BASELINE), Action Recommendations (AX), Diagnosis Suggestions (DX), or both (DXAX). The second factor determined the accuracy of the suggestions at three levels often achieved by fault diagnosis models in prior work [142]—100% accurate, 90% accurate, and 80% accurate. The resulting ten study conditions are enumerated in Table 5.1.

5.3 Domain

We situated our investigation in a mock apartment environment, with a Fetch mobile manipulator performing an object retrieval task (Figure 5.2). We chose the apartment environment as one that would be familiar to study participants, and the retrieval task due to its intuitive nature and the diversity of potential errors [40], many of which are also common in other applications (e.g. occlusion of objects) [169].

We conducted our experiment online by simulating the experience of a participant remotely controlling the robot. Participants were told that they would be using a web interface (Figure 5.3) to guide the robot through error recovery. In reality, in response to their actions, the interface would display videos that showed the robot executing the target behavior. All videos were pre-recorded for consistency and scalability of the experiment. As



(a) (b)

Figure 5.3: (a) The web UI for participants in the BASELINE condition. The red annotations are for illustration purposes only. (b) Examples of starred suggestions for diagnoses (top) and for actions (bottom).

shown in Figure 5.1, we recorded 285 videos representing a rich storyboard of potential recovery behaviors.

In addition to the video of the robot's action execution, the interface displayed a summary of the task objective and showed a history of participant actions alongside the results of those actions (i.e., success/failure). Using this information, at each step of the experiment participants could (1) indicate their diagnoses of problems with the robot from a set of 11 possible problems, D, and (2) select the next action to take to resolve the problem from a list of 17 possible actions, A.

5.3.1 Task Scenarios

The robot's environment consisted of three possible locations the robot could navigate to: *couch*, *dining table*, and *kitchen counter*. Three manipulable objects were present in the environment: *bowl*, *jug*, and *cup*. The robot was able to navigate to locations depending on localization, as well as recognize the three objects or pick up and place them, resulting in 13 possible object-action combinations. To construct an experiment storyboard, we modeled the state as the state of the robot, the objects, and the presence of any of the failures

described below, and then applied deterministic action transitions based on predetermined action preconditions (e.g., executing *pickup(obj)* would cause the robot to pick up the *obj* if it was unoccluded in front of the robot).

In each experiment, the robot started in one of four errors, F1–4, described below. The common task objective (i.e., terminal condition of the experiment) was for the robot to be located near the *couch* while holding the *cup*. In the absence of errors, this objective could be achieved by the robot navigating to the *kitchen counter*, picking up the *cup* (Figure 5.2), and taking it to the *couch*.

To facilitate our goal of evaluating decision support suggestions for robot failures, we injected an error into each participant trial in order to study the participant's recovery behavior. Each failure scenario, and corresponding start state, represents a type of error commonly encountered in robot task execution [169]:

F1: *Mismatch between design and the environment* where the objects are actually at a location different from the one specified in the nominal task objective. Participants started with a view of an empty *kitchen counter* and needed to find the objects, which were on the *dining table*. Min. recovery steps: 3.

F2: *Non-fatal cause of a failure* where the robot was mislocalized so that navigation commands were remapped, which then triggers a failure while trying to find the objects. E.g. the command "Navigate to [locationA]" sent the robot to [locationB] instead. As a symptom of the remapping, participants started with a view of an empty *dining table*. Min. recovery steps: 4.

F3: *Environment occlusion*, with the *jug* occluding the *cup* on the *kitchen counter*. The scenario also showcases aliased faults, because visually this failure is similar to F1. Min. recovery steps: 4.

F4: *Multiple concurrent faults*. The task was misspecified (F1), the *jug* occluded the *cup* (F3), and the *bowl* was placed on top of the *cup* requiring it to also be moved out of

the way (an additional fault). Min. recovery steps: 7.

5.3.2 Suggestions

Depending on the study condition, participants were shown three diagnosis suggestions and/or three action recommendations. Stars were used to recommend diagnosis/actions to participants in the user interface (UI) shown in Figure 5.3b. Instructional text in the UI notified participants that the number of stars was a proxy for model certainty. We provided a ranked list of suggestions without numerical values as a result of recommendations from prior work [130].

Diagnoses: There were 11 total diagnoses, such that $D = D_{none} \bigcup D_{fault} \bigcup D_{distractors}$, where D_{none} indicated no fault in the current robot execution, $|D_{fault}| = 4$ corresponded to four failures from the scenarios above (unknown to participants ahead of time), and $|D_{distractors}| = 6$ represented distractor diagnoses that never occurred in the experiments (e.g., *There is a problem with the camera*). Based on the pilot studies, all diagnoses were assigned easy-to-understand labels, e.g., the term "gripper" was substituted with "hand" because the latter is more accessible to the general public. Each robot state was associated with one or more diagnoses in the set $D_{none} \bigcup D_{fault}$ and we used a lookup table to suggest diagnoses to participants (suggestions were padded to three by random sampling of D).

Actions: There were 17 total actions, such that $A = A_{domain} \bigcup A_{distractor}$, where $|A_{domain}| = 13$ corresponded to the 13 object-action combinations defined with the domain (above), and $|A_{distractor}| = 4$ represented distractor actions that did not cause any visible changes in the participant videos (e.g., *Restart the camera*). Participants could select any of the 17 actions at all times. When presenting action recommendations, we included the optimal action (i.e., the action on the trajectory with the least number of actions to the goal) as the highest priority, and then randomly chose from the remaining executable actions at the state (i.e., actions that would succeed) in order to present a total of three recommendations.

Modulating Accuracy: We provided the participant with three incorrect suggestions if

Suggestion Type	**Acc: 80%**	**Acc: 90%**	**Acc: 100%**
No suggestions	-	-	BASELINE*
AX	AX_{80}	AX_{90}	AX_{100}
DX	DX_{80}	DX_{90}	DX_{100}
DX & AX	$DXAX_{80}$	$DXAX_{90}$	$DXAX_{100}$

**Baseline does not provide feedback, so has no associated accuracy*

Table 5.1: Study Conditions.

the study condition required it. For diagnosis suggestions, we used three random choices among all the diagnoses that were not applicable in the robot state. For action recommendations, we made three random choices among all actions that could successfully execute in the state, taking care to not select the optimal action. In the DXAX conditions, action recommendations were corrupted when diagnosis suggestions were corrupted.

5.4 Experiment Procedure

We conducted a 4x3 between-subjects fractional factorial experiment, with a total of 10 conditions (Table 5.1).

5.4.1 Protocol

We recruited 200 participants through Amazon Mechanical Turk, with 20 participants per condition. The study was designed to take 20 minutes and participants were compensated $4 for their time. After providing basic demographic information, participants were introduced to the robot system through an instructional web page containing an accompanying video (available at https://youtu.be/0jYuxLTKlyM) to familiarize them with the domain. They were then asked five yes/no knowledge review questions to test their understanding of the task. Participant data was discarded if participants failed the review questions more than five times or refreshed the browser during the experiment.

Participants who passed the knowledge review were presented with the UI introduced in Section 5.3 and allowed up to 20 actions to assist the robot. Within the 20 actions,

Metrics (data type)	**Assumed Model**	**Parameter Priors**	**ROPE**
FRR (binary)	$metric_i \sim Bernoulli(p_i)$ $logit(p_i) = \beta_0 + \mathbf{X_{control,i}}\beta_{control} + \mathbf{X_{condition,i}}\beta_{condition}$	$\beta_\cdot \sim \mathcal{N}(0, 10)$	[-0.055, 0.055]
RAX, RDX, CAX, CDX (binary)	$metric_{ij} \sim Bernoulli(p_{ij})$ $logit(p_{ij}) = \beta_0 + \beta_{0,i} + \mathbf{X_{control,i}}\beta_{control} + \mathbf{X_{condition,i}}\beta_{condition} + X_{state_{ij}}\beta_{state}$	$\beta_\cdot \sim \mathcal{N}(0, 10)$ $\beta_{0,i} \sim \mathcal{N}(0, \sigma_i)$ $\sigma_i \sim HS(3, 0, 10)$	[-0.055, 0.055]
SUS ([0, 100])	$metric_i \sim SkewNormal(\mu_i, \sigma, \alpha)$ $\mu_i = \beta_0 + \mathbf{X_{control,i}}\beta_{control} + \mathbf{X_{condition,i}}\beta_{condition}$	$\beta_\cdot \sim \mathcal{N}(0, 10)$ $\sigma \sim HS(3, 0, 22)$ $\alpha \sim \mathcal{N}(0, 4)$	[-2.3, 2.3] $(0.1 * SD[metric])$

HS: Half-Student distribution

Table 5.2: The assumed Generalized Linear Mixed Models for each of the metrics in the analyses. In the models, i indexes a participant, and j is the j[th] action taken by participant i. ROPE is set based on recommendations by Kruschke [170].

participants in the 90% accuracy conditions received inaccurate AX and/or DX suggestions on the 2[nd] and 12[th] actions (if they took at least 12 actions), and participants in the 80% accuracy conditions received inaccurate suggestions on the 2[nd], 5[th], 12[th], and 15[th] actions. Once a participant resolved the error or exhausted their budget of 20 actions, they were directed to a post-study usability questionnaire.

5.4.2 Metrics & Hypotheses

We evaluate the following performance metrics:

1. *Failure resolution rate* (**FRR**): The failure scenario is considered resolved if the participant accomplishes the robot's goal within the budget of 20 actions. FRR captures the likelihood of a participant resolving a failure scenario.

2. *Rate of optimal action selection* (**RAX**): Each state has an optimal action that leads to the goal in the shortest number of actions (Section 5.3). RAX examines the propensity of participants to select the optimal action and is a measure of operator reliance on decision support [132].

3. *Rate of correct diagnosis selection* (**RDX**): Each state corresponds to a set of correct fault diagnoses (Section 5.3.2). RDX examines the propensity of participants to select at least one of those diagnoses and is a measure of operator reliance on decision support [132].

4. *Compliance with AX suggestions* (**CAX**): When provided with action recommendations (in the AX or DXAX conditions), CAX captures participants' likelihood of following those suggestions and is a measure of operator compliance with decision support [132].

5. *Compliance with DX suggestions* (**CDX**): When provided with diagnosis suggestions (in the DX or DXAX conditions), CDX captures participants' likelihood of following those suggestions and is a measure of operator compliance with decision support [132].

6. *System Usability Scale* (**SUS**): The SUS is a 10-item Likert scale used the measure the usability of the UX [171] and was administered to participants in the post-study questionnaire. In our study, the reliability rating, Cronbach's α, for items in this instrument was 0.94.

The hypotheses associated with each of the above metrics are enumerated in Table 5.3. Each metric is associated with two hypotheses pertaining to each of the two research questions that motivate this work.

5.4.3 Bayesian Data Analysis

We draw our conclusions from a Bayesian analysis performed on Generalized Linear Mixed Models over the data. The Bayesian analysis allows us to quantify both the likelihood for the *existence* of an effect as well as a practical estimate of the *significance* of that effect. To perform the analysis, we assume that all our metrics are generated under a structural model with the following explanatory variables:

- $\mathbf{X_{condition,i}} = X_{Type,i} + X_{Acc,i}$: A suggestion type factor ($Type \in \{$BASELINE, AX, DX, DXAX$\}$) and an accuracy factor ($Acc \in \{80\%, 90\%, 100\%\}$). We encode the Type factor as Helmert contrasts and report the effects of AX, DX, and DXAX levels vs. the BASELINE level. We encode the Acc factor as orthogonal polynomials in order to investigate linear or quadratic trends in the effects of the factor. Our model does not include interaction effects between Type and Acc in order to preserve model identifiability.
- $\mathbf{X_{control,i}}$: The demographics of a participant and the failure scenario (F1–4) they were assigned.
- $X_{state,ij}, \beta_{0,i}$ (for RAX, RDX, CAX, & CDX): The state of the robot and a random intercept effect of the participant.

Depending on the nature of a metric's data (i.e. binary, count, etc.), we fit recommended probability distributions [172] using Maximum Likelihood Estimation (MLE). The chosen distribution is the one that had the lowest AIC of fit. The probability distribution and the model we used in the analysis of each metric are listed in Table 5.2.

In order to perform Bayesian analysis, we formulate a null hypothesis of minimal effect based on the nature of the data: this is called the Region of Practical Equivalence (ROPE). For instance, we can set the ROPE to be [-0.055, 0.055] for binary data, which then implies that effects resulting in a likely change of less than 0.055 (within the ROPE interval) are considered insignificant (we cannot reject the null hypothesis). The ROPE for each analysis are in Table 5.2.

We begin the analysis by initializing model parameters with weak priors (e.g. $\mathcal{N}(0, 10)$). We then sample parameters for models that might explain the observed data using Hamiltonian Monte-Carlo sampling [172, 173]. We use 4 chains with a burn-in of 1000 iterations, before sampling for 1000 iterations to get the the posterior distribution of the parameters. We verify the diagnostics of the convergence of the samples using established methods involving Leave-One Out Cross-Validation [174]. Note that the posterior distributions of the parameters imply a posterior distribution on the metrics' values. Our inferences on the

Metric	Hypotheses	Metric better with AX	Metric better with DX	Metric better with DXAX	Metric trend with Acc. is Linear	Metric trend with Acc. is Quadratic
FRR	**H1$_{\mathbf{FRR}}$**: FRR increases with suggestions than without. **H2$_{\mathbf{FRR}}$**: FRR increases with suggestion accuracy.	97.2% [pd] ***		96.2% [pd] ***		U-shape 97.0% [pd] *
RAX	**H1$_{\mathbf{RAX}}$**: RAX increases with suggestions than without. **H2$_{\mathbf{RAX}}$**: RAX increases with suggestion accuracy.	97.0% [pd] **		99.0% [pd] **		
RDX	**H1$_{\mathbf{RDX}}$**: RDX increases with suggestions than without. **H2$_{\mathbf{RDX}}$**: RDX increases with suggestion accuracy.		97.8% [pd] *	98.7% [pd] **	Positive slope 99.3% [pd] *	
CAX	**H1$_{\mathbf{CAX}}$**: CAX improves with DX suggestions. **H2$_{\mathbf{CAX}}$**: CAX increases with suggestion accuracy.	N/A‡				
CDX	**H1$_{\mathbf{CDX}}$**: CDX improves with AX suggestions. **H2$_{\mathbf{CDX}}$**: CDX increases with suggestion accuracy.		N/A‡		Positive slope 100% [pd] *	
SUS	**H1$_{\mathbf{SUS}}$**: SUS increases with suggestions than without. **H2$_{\mathbf{SUS}}$**: SUS increases with suggestion accuracy.	98.9% [pd] n.s.	95.2% [pd] n.s.			

‡ *CAX (CDX) does not apply when AX (DX) is not present.*

Table 5.3: Metrics, hypotheses, and the main effects results from the data analysis (Section 5.4.3). In the results columns, we report in the table if [pd] >95%. We show an effect size if the overlap in ROPE is <2.5%. Effect sizes are indicated by the asterisks: *** for a large effect (Std.Median >.8), ** for a medium effect (Std.Median >.5). and * for a small effect (Std.Median >.2) [175].

effects of interest are performed using the posterior distributions.

Using the guidelines presented in [176], we report:

1. A Probability of Direction [pd], which quantifies the likelihood of the *existence* of an effect.
2. The Median and the 89% High Density Credible Intervals (CI) of effect sizes. Effect sizes are classified from Median estimates using the thresholds in prior work [175].
3. The degree of overlap of the full posterior distribution of the metric with the ROPE. The value is used to reject (or not) the null hypothesis on the *significance* of the effect.

We report only a subset of the data analyzed in this chapter. Interested readers can find the complete analysis, including model diagnostics and the effects of non-condition factors at https://bit.ly/2UjPPtE.

5.5 Results

Table 5.3 summarizes our research hypotheses and key results of the study, which we discuss in detail in this section.

Demographics: Our experiment consisted of 200 participants (age group mode 26–30 years, 36.5% / 63% / 0.5% female/male/unspecified gender). The majority of participants (166/200) interacted with a robot at most three times a year.

Failure Resolution Rate (FRR): Figure 5.4a shows the proportion of participants that resolved the fault for a given condition. Across conditions, the FRR ranged from 0.6 (DX_{90}) to 1.0 (AX_{100}).

On evaluating $\mathbf{H1_{FRR}}$, we find that the resolution rate with action suggestions (AX) compared to BASELINE has a 97.2% [pd] of being positive (Median = 1.89, 89% CI [0.29, 3.37]) and can be considered large (Std.Median = 1.04) and significant (0.73% in ROPE) [ROPE (full)]. We also find that the resolution rate with both suggestions (DXAX) compared to BASELINE has a 96.2% [pd] of being positive (Median = 1.57, 89% CI [0.16,

2.97]) and can be considered large (Std.Median = 0.87) and significant (1.15% in ROPE) [ROPE (full)]. As seen in Figure 5.5a, the results indicate that adding action suggestions (AX and DXAX) greatly increases the probability of the participants resolving the robot's faults.

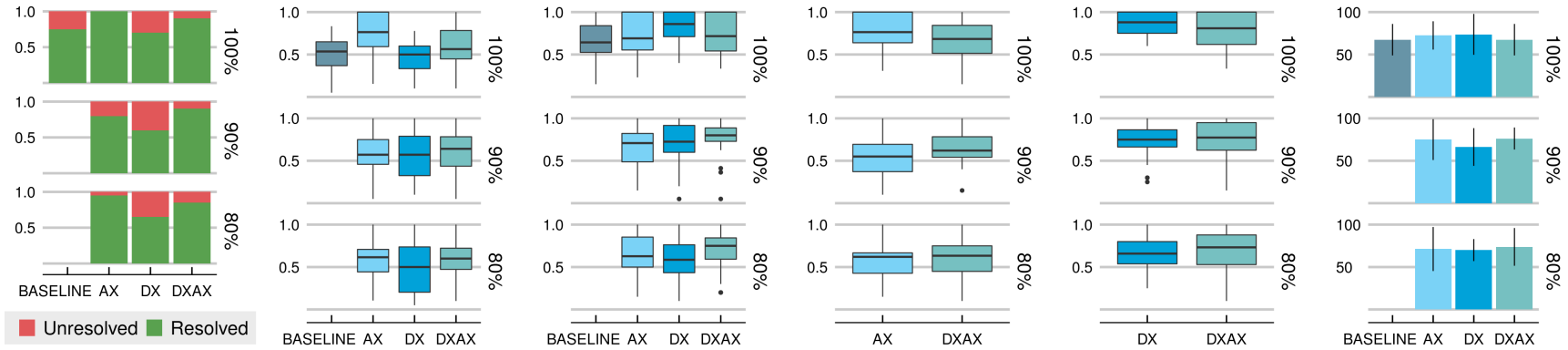
On evaluating $\mathbf{H2_{FRR}}$, we find that the noise level has a quadratic relationship to the probability of fault resolution with a 97.0% [pd] positive effect size (convex-shape) (Median = 0.77, 89% CI [0.12, 1.40]), which can be considered small (Std.Median = 0.42) and significant (1.75% in ROPE) [ROPE (full)]. Therefore, as seen in Figure 5.5b, the data suggests that as the accuracy of suggestions increases, there is a U-shaped relationship to participant performance.

Rate of optimal action selection (RAX): Figure 5.4b shows the proportion of optimal actions taken by participants in each study condition. Across conditions, the Median RAX ranged from 0.50 (DX_{100}, DX_{80}) to 0.76 (AX_{100}).

On evaluating $\mathbf{H1_{RAX}}$, we find that the optimal action rate with action suggestions (AX) compared to BASELINE has a 97.0% [pd] of being positive (Median = 1.03, 89% CI [0.16, 1.95]) and can be considered medium (Std.Median = 0.57) and significant (1.98% in ROPE) [ROPE (full)]. We also find that the optimal action rate with both suggestions (DXAX) compared to BASELINE has a 99.0% [pd] of being positive (Median = 1.20, 89% CI [0.39, 2.09]) and can be considered medium (Std.Median = 0.66) and significant (0.50% in ROPE) [ROPE (full)]. As seen in Figure 5.5c, the results indicate that adding action suggestions (AX and DXAX) greatly increases the likelihood that participants take the desired actions to resolve a failure.

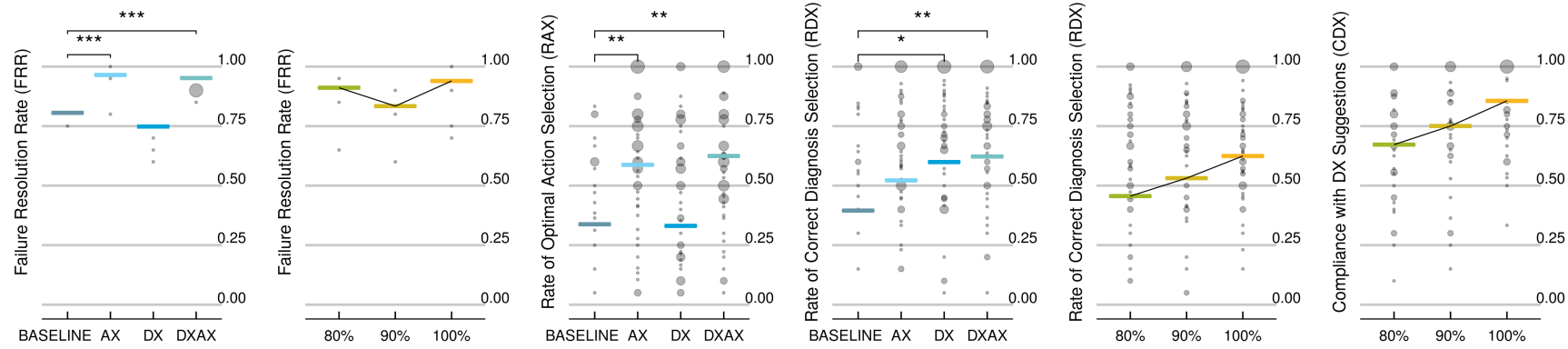
On evaluating $\mathbf{H2_{RAX}}$, we find no significant effect of the optimal action rate with the accuracy of the suggestions.

Rate of correct diagnosis selection (RDX): Figure 5.4c shows the proportion of of correct diagnoses made by participants in each study condition. Across conditions, the Median RDX ranged from 0.59 (DX_{80}) to 0.86 (DX_{100}).



(a) FRR (b) RAX (c) RDX (d) CAX (e) CDX (f) SUS

Figure 5.4: Study data for each of the metrics defined in Table 5.3.



(a) FRR v. Type (b) FRR v. Acc. (c) RAX v. Type (d) RDX v. Type (e) RDX v. Acc. (f) CDX v. Acc.

Figure 5.5: Predicted Median of the posterior of significant effects after Bayesian analysis. Asterisks indicate effect sizes (see Table 5.3). Points in the figure represent data from the study; larger points indicate more data instances with the same value.

On evaluating $\mathbf{H1_{RDX}}$, we find that the correct diagnosis rate with diagnosis suggestions (DX) compared to BASELINE has a 97.8% [pd] of being positive (Median = 0.82, 89% CI [0.15, 1.51]) and can be considered small (Std.Median = 0.45) and significant (1.60% in ROPE) [ROPE (full)]. We also find that the correct diagnosis rate with both suggestions (DXAX) compared to BASELINE has a 98.7% [pd] of being positive (Median = 0.92, 89% CI [0.22, 1.60]) and can be considered medium (Std.Median = 0.51) and significant (0.98% in ROPE) [ROPE (full)]. As seen in Figure 5.5d, the results indicate that adding diagnosis suggestions (DX and DXAX) results in more correct diagnoses.

On evaluating $\mathbf{H2_{RDX}}$, we find a 99.3% [pd] positive effect (positive slope) of accuracy in suggestions to correct diagnosis rate (Median = 0.48, 89% CI [0.16, 0.79]), which can be considered small (Std.Median = 0.26) and significant (1.25% in ROPE) [ROPE (full)]. As shown in Figure 5.5e, there is a linear improvement in the rate of correct diagnoses from participants as the accuracy of suggestions improves.

Compliance with AX suggestions (CAX): Figure 5.4d shows the proportion of participants that complied with action suggestions in the conditions that received AX suggestions (AX & DXAX). Across conditions, the Median CAX ranged from 0.55 (AX_{90}) to 0.76 (AX_{100}). On evaluating $\mathbf{H1_{CAX}}$ and $\mathbf{H2_{CAX}}$, we find no significant effects of either diagnosis suggestions (DXAX vs. AX) or the accuracy of the suggestions.

Compliance with DX suggestions (CDX): Figure 5.4e shows the proportion of participants that complied with diagnosis suggestions in the conditions that received DX suggestions (DX & DXAX). Across conditions, the Median CDX ranged from 0.66 (DX_{80}) to 0.88 (DX_{100}).

On evaluating $\mathbf{H1_{CDX}}$, we find no significant effects of action suggestions (DXAX vs. AX) on the compliance rate. On evaluating $\mathbf{H2_{CDX}}$, we find that there is a 100% [pd] positive effect (positive slope) of accuracy in suggestions to the rate of compliance with suggestions (Median = 0.77, 89% CI [0.39, 1.11]), which can be considered small (Std.Median = 0.42) and significant (0.08% in ROPE) [ROPE (full)]. As shown in Figure 5.5f, the

compliance of participants with diagnosis suggestions improves as the accuracy improves.

System Usability Scale (SUS): Figure 5.4f shows the responses of participants to the SUS questionnaire across the different conditions. Across conditions, the median SUS scores range from 66 (DX_{90}) to 76 ($DXAX_{90}$).

On evaluating $\mathbf{H1_{SUS}}$, we find that the score with action suggestions (AX) compared to BASELINE has a 98.9% [pd] of being positive (Median = 11.68, 89% CI [3.82, 19.0]), which can be considered medium (Std.Median = 0.56) but not significant (3.02% in ROPE) [ROPE (full)]. We also find that the SUS score with diagnosis suggestions (DX) compared to BASELINE has a 95.2% [pd] of being positive (Median = 8.16, 89% CI [1.17, 16.04]), which can be considered small (Std.Median = 0.38) but not significant (9.48% in ROPE) [ROPE (full)]. The results indicate that adding suggestions, AX or DX, but not both (DXAX), might result in greater usability.

On evaluating $\mathbf{H2_{SUS}}$, we find that the accuracy of suggestions does not affect the usability score.

5.6 Discussion and Conclusions

In this section, we discuss the implications of the statistical results presented above. We frame the discussion in relation to our research questions, with potential guidelines for future UX development highlighted in bold.

RQ1—Type of Decision Support: **Error recovery systems should display both feedback and feedforward information for maximum effectiveness.** From the results of evaluating $H1_{RAX}$ and $H1_{RDX}$, we find that participants are more likely to select the correct failure resolution actions if feedforward action recommendations (AX) are provided, and more likely to select the correct problem diagnoses if feedback as diagnosis suggestions (DX) are provided. Operators perform both functions in most common error recovery scenarios, with diagnosis typically informing identification of subsequent actions [165, 143, 135]. As a result, most systems should display both feedforward and feedback information.

Feedforward information has a greater effect on successful error resolution than feedback information. Analysis of the failure resolution rate metric in the context of $H1_{FRR}$ highlights that participant ability to successfully recover from errors was greatest in the presence of feedforward action recommendations (AX & DXAX conditions) than when presented with feedback diagnosis information alone (DX). The result, consistent with UX research [162], demonstrates that although diagnosis suggestions aid in greater understanding of the overall state of the system (as shown by RDX results), feedforward information, suggesting "what-to-do" is more effective in leading to the correct solution.

RQ2—Decision Support Accuracy: **If the feedforward information is noisy (as in most systems), supplementing feedforward with feedback information (even if also noisy) leads to effective recovery strategies.** Analysis of the failure resolution rate metric in the context of $H2_{FRR}$ highlights that participant performance was significantly affected by accuracy levels. Specifically, we observe a U-shaped response in which participant performance is likely to drop significantly in the 90% accuracy conditions. The drop is likely evidence of overtrust in the system [164]. However, Figure 5.4a and Figure 5.4b provide an indication that the performance drop might not be present in the DXAX conditions, indicating that diagnosis information, and the situational awareness that users might gain from it, can help ameliorate overtrust in the faulty system.

Additional Findings: Evaluating $H2_{CDX}$, we find that the compliance of participants with the suggestions linearly improves with the accuracy of the suggestions. The finding is consistent with prior work, which has found that operator compliance is dependent on the reliability of the decision support [132, 130]. Additionally, we find that there is a linear improvement in participants choosing the correct diagnoses (RDX) as the accuracy of suggestions improves, showing that reliance on decision support can also be dependent on the reliability of the decision support [132].

Evaluating $H1_{SUS}$, we find that both feedforward (AX) suggestions and feedback (DX) suggestions might independently improve the usability of an error recovery UX over a

baseline without decision support, but the same might not hold true when both are present (DXAX). While the result might indicate a potential problem of too-much-information, further investigation is needed because the independent effects of DX and AX were not considered to be of practical significance (based on overlap within the ROPE).

Final Conclusions: In summary, we find that users are most effective in guiding error recovery when both feedback-focused diagnosis information and feedforward-focused action suggestions are presented. Note that further studies are required to better understand the above effects. For instance, we evaluated non-experts for whom diagnosis suggestions might not have been as useful as for experts. Additionally, when inaccurate, our suggestions for diagnoses and actions were inaccurate at the same time, but an accurate diagnosis might have ameliorated inaccurate action recommendations, or vice-versa. Finally, we assumed that (a) the space of diagnosis and action suggestions corresponded to the space of diagnoses and actions picked by operators and (b) suggestions were provided frequently and at each time step: changing the granularity or frequency of suggestions could lead to different effects. Future work can study the additional factors.

In the next chapter, we turn our attention to the effects of operator accuracy on the accuracy of decision support models: the second interaction that we uncovered in Chapter 4. We use the data from our experiments in this chapter to evaluate the robustness of the decision support models in the next.

CHAPTER 6
ON THE ACCURACY OF DECISION SUPPORT MODELS DURING ROBOT FAILURE INTERVENTIONS

Our work so far has confirmed the results of prior research, finding that providing operators with decision support improves their accuracy in a task [130, 131, 177, 132] (see Chapter 5). In fact, the effects can be part of a virtuous positive feedback cycle: more accurate decision support leads to increased compliance [130, 132], which can then further improve operator accuracy. Therefore, creating accurate decision support models to assist operators in their decision-making during an intervention is of paramount importance.

However, developing accurate decision support models presents many challenges. While prior work has generated troubleshooting aids to help humans resolve failures [165, 136, 139], the methods require accurate domain models, which can be cumbersome to specify and are often brittle in the face of increasing system complexity [28]. Other work has used data-driven approaches to generating diagnoses [142, 141] and actions [150, 149] on a robot, which can be used as decision support. Unfortunately, we have found that even with highly accurate decision support, operator compliance can be low and operator inaccuracy high [130, 177], possibly as a result of unfamiliarity, lack of information, lack of expertise, overtrust, or other factors [164, 133, 129, 178] (again, see Chapter 5). Therefore, as seen from the examples in Figure 6.1, data-driven decision support models need to be made robust to operator errors in an intervention.

In this work, we highlight the issues that can arise from the tightly coupled interactions of a machine learning model and a human during robot failure interventions and consider methods for making the model more accurate and robust. We begin by formalizing the intervention process as a Partially Observable Markov Decision Process (POMDP) and specify the aim of developing decision support models in terms of that formalization. We

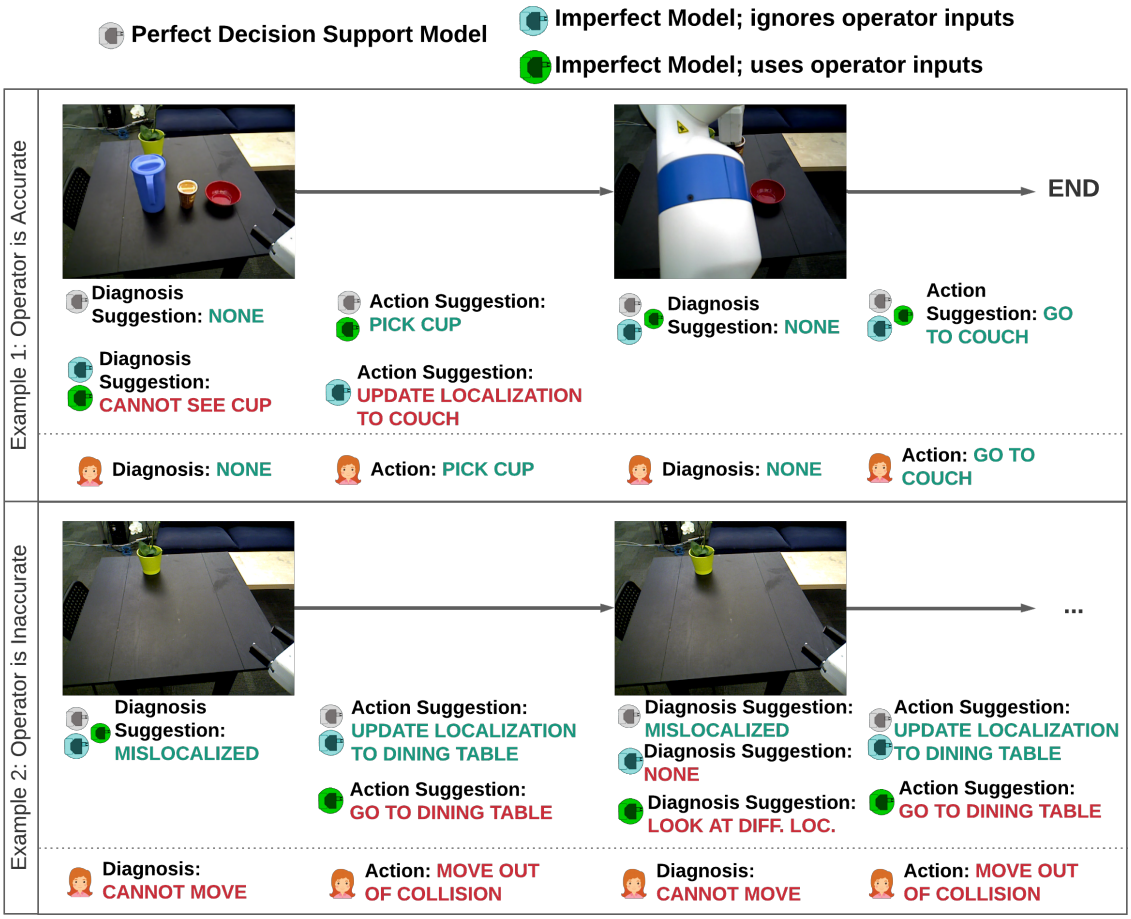


Figure 6.1: Decision support outputs from models in this work to examples of operator behaviour witnessed in Chapter 5. Text in green indicates accurate diagnoses/actions while text in red indicates inaccurate diagnoses/actions. The accuracy of an imperfect decision support models can be improved if the operator is themselves accurate (top). However, the same model can suffer degraded accuracy if the operator is inaccurate (bottom).

then conduct six experiments (Section 6.4) in which we vary the inputs to the decision support models and the techniques used to learn the parameters for them using noisy datasets of interventions.

We gain the following insights from the results of our experiments: (1) the robustness and accuracy of decision support models is greatly improved if the features that are used as inputs are informative of the failures, (2) diversity in the training data is particularly helpful for decision support robustness, and (3) current techniques in unstructured machine learning are unable to meet the challenge of adequately learning to assist operators in an

intervention domain. The insights can inform future work that might wish to develop decision support for robot operators.

6.1 Related Work

Prior research has long studied the proclivity of human operators to make mistakes [129, 36], even when provided with decision support [130, 131] (see also Chapter 5 and [43]). However, the research is frequently conducted in the context of simple one-off tasks [130, 131], and any inaccuracies in the decision support are pre-specified to maintain control of experimental conditions [43, 130, 131]. In contrast to the prior research, we do not control the predictions from our decision support models and instead evaluate the *accuracy of the models* as a result of interactions with mistake-prone human operators.

There is a large body of work in robotics focused on fault diagnosis [28, 142, 141, 179] or in assisting with action selection and planning [150, 149, 147, 11], which can be leveraged to develop decision support models. The techniques used are broadly categorized [28] as either model and knowledge-based, leveraging models of the system or environment to produce diagnosis and action recommendations [55, 145], or statistical and data-driven, leveraging the correlations between observed features and diagnoses or actions in a dataset [142, 141]. The former set of approaches, while expressive and capable of providing detailed outputs, are often cumbersome to specify and can be brittle in the face of increased system complexity [28]. Therefore, we eschew the effort of modeling the domain to see if diagnosis suggestions and action recommendations can be learned directly from the data.

Automated troubleshooting aids to provide humans with diagnoses as well as diagnostic and repair action suggestions have been developed in the past, especially in fields beyond robotics [165, 180, 136, 181, 139, 135, 143]. Some of the works account for the cost of actions in order to facilitate recoveries while minimizing the total cost incurred [180, 136, 181, 139, 143]. Yet others attempt to remain robust to noisy observations and actions by

formulating the problem as Markov Decision Processes (MDPs) [182, 136, 181]. However, all of the works require well-specified domain models of the consequences of actions and the relationships between observations and diagnoses—in the parlance of Section 6.2.1, $\mathcal{T}$, $P_{\mathcal{D}}$, $P_{\mathcal{O}}$, or some combination of the three, must be specified. The requirement introduces unwanted modeling effort and can hinder troubleshooting if the models are misspecified. Additionally, many of the works have not been evaluated for online troubleshooting during a robot intervention [182, 180, 136, 181, 143]. In this work, we utilize the MDP formalization of troubleshooting, which is similar to the formalisms used in human-robot shared autonomy [183, 184], and apply it to modeling human-robot interaction during interventions. We also adopt a data-driven approach to our decision support models and make no assumptions on the availability of domain information that can be used improve model inference or can be incorporated as structured priors to facilitate model learning.

Learning unstructured models from inaccurate data is a large field of study [185], with much interest arising from the fields of adversarial learning [186, 187], semi-supervised learning [188], robust statistics [189], among others. The techniques either focus on noise in the features or noise in the labels; seldom both. Introducing noise in the features, sometimes called domain randomization, can in fact make a model robust by preventing overfitting [190, 191]. Noise in the labels is a harder problem [192, 193] but some machine learning models, such as neural networks, are able to learn accurate models in spite of it [194, 195]. Nevertheless, works have proposed using heuristics such as early stopping [196], learning noise models using trusted subsets of data [197, 198], clustering [199], reweighting data points [200, 201], and even using secondary learning objectives [188], to improve accuracy when there is label noise. The techniques have had great success in different domains, but they are often evaluated in one-off classification tasks, which are not characteristic of robot interventions. In particular, during an intervention, label noise in the form of operator selected diagnoses or actions in one timestep, can become feature noise in the next, by changing the distribution of input data provided to the suggestions model. There-

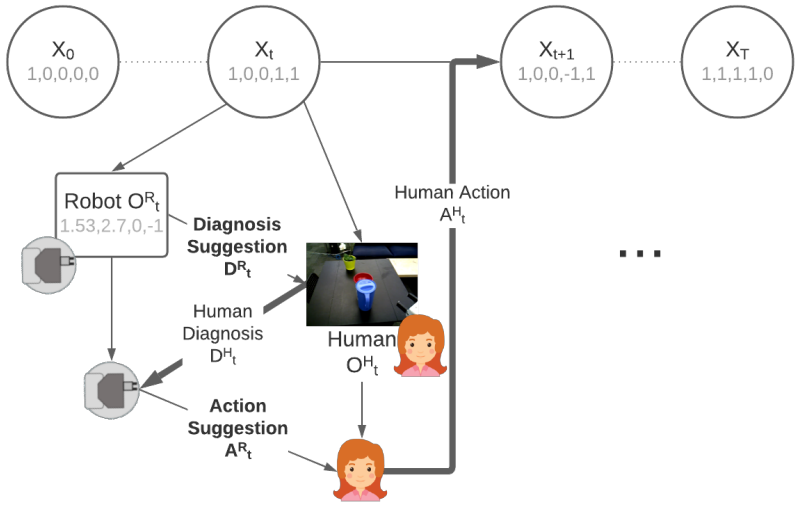


Figure 6.2: An intervention process.

fore, it is unclear from the prior work if the techniques used to learn from noisy data in the literature are capable of creating robust models that can be used in robot interventions.

6.2 Definitions

In this section we formalize an intervention process, introduce the challenges unique to creating decision support systems within the domain, and then introduce our research questions in the context of those challenges.

6.2.1 The Intervention Process

Recall that a situation where a human is tasked with assisting a robot recover its autonomy after a failure is termed an *intervention* [31]. The goal during an intervention for both the human and robot is to reach a *resolution* of the intervention: a state from which the robot can resume autonomous operation. We assume that the human enacts a supervisory role during the intervention by directing the robot's actions, while the robot executes the humans directives and can, in addition, provide additional information to the human in order to better guide the human towards a resolution (Figure 6.2).

Concretely, we assume that the state of the world during an intervention can be perfectly

modeled by an underlying state at time t, $X_t \in \mathcal{X}$, which is unobservable to the actors, the robot and the human. The goal of both the robot and the human, the resolution of the intervention, is then defined by a subset of states, $\mathcal{G} \subset \mathcal{X}$, from which the robot can resume autonomous operation. Therefore, upon starting at a non-resolution state, $X_0 \notin \mathcal{G}$, the robot and human must collaborate to reach $\mathcal{G}$ in the fewest number of steps.

We assume that each actor in the intervention has access to a partial observation of the situation, which is dependent on the state X_t. Specifically, we assume that the robot has access to an observation, $O_t^R \in \mathcal{O}^R$ where $\mathcal{O}^R$ need not overlap with $\mathcal{X}$, and the human has access to $O_t^H \in \mathcal{O}^H$ where $\mathcal{O}^H$ need not overlap with $\mathcal{O}^R$ or $\mathcal{X}$. We can therefore define the existence of observation models, $P_{\mathcal{O}^R} : \mathcal{X} \to \mathcal{O}^R$ and $P_{\mathcal{O}^H} : \mathcal{X} \to \mathcal{O}^H$, which dictate the relationship of the observations of the robot and operator to the underlying state. There is considerable research, especially in the realms of Human Factors and User Interface Design, that characterizes the desirable properties of $\mathcal{O}^H$ in order to provide the "best" information to the human [202, 137]; we do not address $\mathcal{O}^H$ or $P_{\mathcal{O}^H}$ in this work.

Given incomplete observations, we assume that a diagnosis, $D_t \in \mathcal{D}$, provides supplementary information by helping both agents localize their beliefs on the underlying state, X_t; e.g., a diagnosis of *robot mislocalized* can help distinguish between states where object recognition fails due to a navigation error from states where object recognition fails due to failures in perception. As such, a diagnosis is dependent on the state, i.e. we can assume the existence of a diagnosis model $P_{\mathcal{D}} : \mathcal{X} \to \mathcal{D}$, and is a means of sharing state information between the human and robot. The robot's goal is to suggest diagnoses, $D_t^R \in \mathcal{D}$, to help the human better understand the problem or situation. The human's goal is to provide the robot with diagnoses, $D_t^H \in \mathcal{D}$, to enable the robot to provide better decision support in the future.

Actions, $A_t \in \mathcal{A}$, evolve the state according to an unknown (to the human and robot) transition model, $\mathcal{T} : \mathcal{X} \times \mathcal{A} \to \mathcal{X}$, and are therefore the means by which the problem(s) in an intervention may be rectified. We assume that the robot executes actions selected

by the human[1] and has the ability to influence the human's action selection by providing recommendations. The robots goal is to suggest actions, $A_t^R \in \mathcal{A}$, such that the human resolves the intervention in the fewest steps; the human's goal is to select actions, $A_t^H \in \mathcal{A}$ to also resolve the intervention as quickly as possible.

Finally, we assume that an intervention continues for T timesteps until a resolution is reached, i.e., $X_T \in \mathcal{G}$; or until a timeout is encountered, i.e. $X_T \notin \mathcal{G}$, in which case the intervention remains unresolved. We assign an outcome variable, $R_T = 1$ if the intervention is resolved and $R_T = -1$ if the intervention remains unresolved.

We have thus defined a Partially-Observable Markov Decision Process (POMDP), which has also been used in prior work in troubleshooting [181, 136] and in shared human-robot autonomy [183, 184]. Our POMDP is described by the 7-tuple, $(\mathcal{X}, \mathcal{A}, \mathcal{T}, R, \{\mathcal{O}^R, \mathcal{D}\}, \{P_{\mathcal{O}^R}, P_{\mathcal{D}}\}, \gamma)$, where most of the terms are defined above and $\gamma \in [0, 1]$ is a discount factor that can be used to forecast the likely value for R_T before the end of an intervention is actually encountered.

6.2.2 Decision Support Models

In order to provide suggestions to the human, the robot must have access to a model that it can use to generate diagnosis and action suggestions in a given situation. As mentioned in Section 6.1, we adopt a data-driven approach to the problem. Concretely, we assume access to an intervention dataset with N recorded interactions of the human during failure interventions, $\Xi = \{\xi_{T_1}, \ldots, \xi_{T_N}\}$, where each $\xi_{T_i} = ([(O_0^R, D_0^H, A_0^H), \ldots, (O_{T_i-1}^R, D_{T_i-1}^H, A_{T_i-1}^H)], R_{T_i})$ is tuple containing a sequence of the intervention data of length T_i, and R_{T_i}, the intervention's outcome. Each term in the sequence consists of O_t^R, the robot's observation, D_t^H, the diagnosis selected by the human, and A_t^H, the action selected by the human. Given Ξ, we train two decision support models—$f_{\mathcal{D}}$, to suggest diagnoses, and $f_{\mathcal{A}}$, to suggest actions at every timestep.

[1] In non-safety critical domains where the robot has sufficient confidence in its action selection, it may execute the action autonomously while supervised by the human.

6.2.3 Research Questions

Prior work in Human Factors [130] and Human-Robot Interaction [43] has found that even with accurate decision support, humans can showcase high levels of error through non-compliance and inaccuracy. The observation presents challenges in learning and inference for $f_{\mathcal{D}}$ and $f_{\mathcal{A}}$.

The primary challenges in learning the models are two-fold: (1) human inaccuracy can result in multiple labels for the same input, resulting in noisy and sometimes adversarial training data; and (2) even when the training data is perfect, the imperfect nature of the robot's observations mean that multiple labels can *still* be aliased given an observation. There is a large body of work in machine learning dedicated to robust learning from noisy features and labels (Section 6.1), and we draw inspiration from such works to investigate model performance under realistic learning challenges.

Our work also focuses on investigating and addressing challenges that arise during inference, where the human's errors can have adverse effects on model accuracy at test time. In particular, human error can (1) introduce noise into the features of the model by providing unexpected diagnoses or actions as inputs, and (2) through inaccurate actions, induce unexpected domain-shifts in model features at test time. To the best of our knowledge, scant research in robotics has characterized these problems; we address that.

Concretely, our research questions are as follows:

RQ1 *What inputs provided to a decision support model can make it more accurate and also more robust given human inaccuracies?* We investigate this question to characterize the features that might be available to a model during an intervention to help it improve accuracy while remaining robust to operator errors.

RQ2 *How can techniques in learning from noisy data for unstructured machine learning help maintain the robustness of models while also increasing decision support accuracy?* With the second question we aim to identify techniques used in prior work that

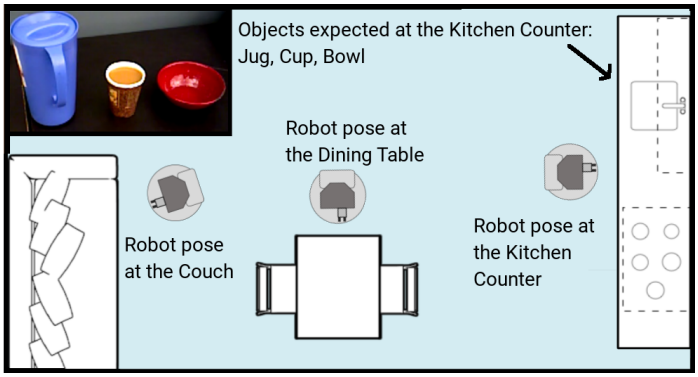


Figure 6.3: An overhead schematic of the evaluation domain.

learn from noisy data so that we can learn accurate models from a noisy dataset of interventions while remaining robust to noise at inference time.

In our effort to answer both questions, we survey a variety of features and techniques. Additionally, we focus particularly on including the human operator's inputs, $D_{\cdot}^{H}$ & $A_{\cdot}^{H}$, directly as features to answer RQ1 because the operator's inputs can be particularly helpful to the model if correct (see Figure 6.1). Similarly, we focus on the accuracy of labels in the training dataset to answer RQ2 because one of the most prevalent findings in prior works has found that some noise (or variety or randomness) in the training dataset is necessary for the increased robustness of unstructured models [190, 195, 191, 197, 188].

6.3 Evaluation Setup

In this section we introduce the domain, the decision support models, and the datasets that we used to investigate our research questions.

6.3.1 Domain

We extend a pick-and-place domain in a living-room environment, introduced in Chapter 5 to evaluate our research questions[2] (Figure 6.3). The domain simulates a closed-world environment with deterministic transitions as a result of robot actions. In particular, the

[2]Available at https://github.com/GT-RAIL/isolation_cyoa

domain is implemented as a story graph, with data from real robot executions on a Fetch robot accompanying all possible transitions in the graph. Such a setup provides us with the ground-truth knowledge necessary to evaluate the answers to our research questions while at the same time preserving the realism and the partial observability of real intervention scenarios.

The domain has three locations—a Kitchen Counter, a Dining Table, and a Couch—as well as three objects—a Cup, a Jug, and a Bowl. All the objects are expected to be at the Kitchen Counter, but based on the domain initialization, they may be placed on the Dining Table instead. The objects are expected to be visible and unobstructed, but based on the domain initialization, they might be occluded or in collision. The nominal task plan of the robot is to pick up the Cup from the Kitchen Counter and bring it to the Couch. The domain is designed to simulate errors that might occur in the nominal task.

The underlying state of the domain, $X_t \in \mathcal{X}$, is represented by a 7-tuple characterizing the robot's location in the environment, the location of the objects in the room and their placement relative to each other, the localization state of the robot (localized or not), and the object, if any, in the robot's gripper. There are $|\mathcal{X}| = 270$ states in total, with $|\mathcal{G}| = 8$ states that satisfy the task objective. An intervention simulated in the domain starts at one of 262 non-goal states and is resolved when one of the eight goal states is reached.

As part of the domain definition, there are a total of $|\mathcal{D}| = 11$ plausible problem diagnoses that are defined, including a catch-all diagnosis of *none*, for when no problems exist. However, six of the diagnoses are distractors with the available states actually simulating only five diagnoses: each state is associated with one of the five diagnoses. Examples of diagnoses include *cup is not visible* (used), *robot is mislocalized* (used), and *robot has low battery* (distractor).

The domain also defines $|\mathcal{A}| = 17$ actions that can be attempted at any of the states in order to transition to a new state. However, actions only succeed if their preconditions, specified in the domain of $\mathcal{X}$, are satisfied; as such, the preconditions of the actions are

unknown to the robot or human operator. As with the diagnoses, each state is annotated with one out of 10 actions that lie on the shortest path from the state to one of the goal states, and seven of the 17 actions are distractor actions that are plausible in the domain, but that never lie on the shortest path. Examples of actions include *go to the couch* (used), *pick up the jug* (used), *dock with the charger* (distractor), and *look at the kitchen counter* (distractor)[3].

The Fetch robot within the domain uses a pick-and-place architecture introduced in prior work [40] (see Chapter 2). The architecture provides signals for easy fault detection and diagnosis in a pick-and-place task, and as such, these signals, enumerated below, constitute the domain of robot observations, $\mathcal{O}^R$:

`num_objects` Number of object point clouds. Available after an attempt at a *pick* action.

`duration` The duration of the last action.

`grasped` Indicator for an object in the robot's gripper.

`[loc]_distance` Distance of the robot to each of the three locations, according to its localization.

`[loc]_heading` Heading of the robot to each of the three locations, according to its localization.

`arm_moved` Indicator for arm movement in the last action.

`action_success` Indicator for success of the last action.

Note that $\mathcal{O}^R$ does not contain features directly identifying the robot's localization state, the location of the objects in the environment, or to the relative poses of the objects with respect to each other, i.e. it does not contain aspects of the underlying state. Therefore, as defined in Section 6.2.1, $\mathcal{O}^R$ has only a 'partial' view of the true state of the domain.

The work in the previous chapter, which introduced the domain, used 285 robot executions to generate robot data for all possible state-action transitions in the domain. We

[3]Some distractor actions can help a human gain situational awareness but they don't advance the state of the system.

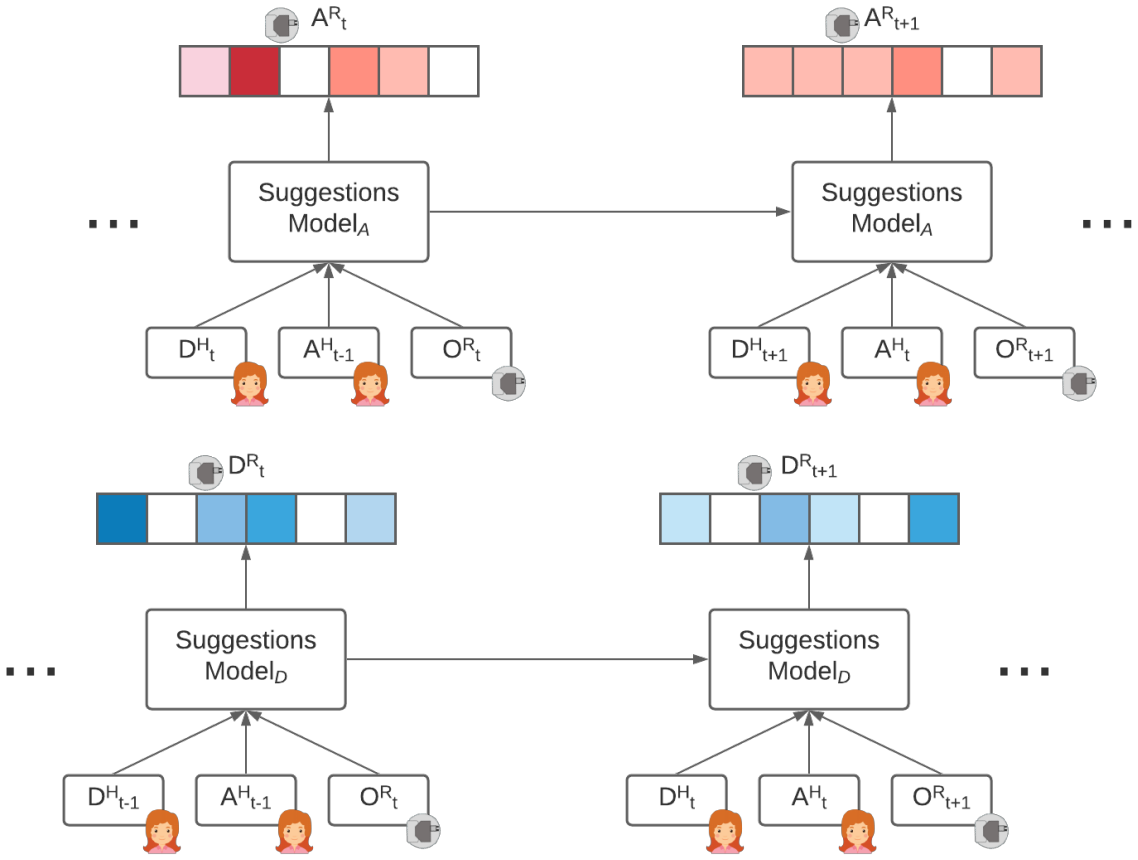


Figure 6.4: The neural network suggestions models.

processed data from the executions to create $|\mathcal{O}^R| = 1152$ robot observations that could be assigned to all state-action transitions. As a result of the processing, all 262 non-goal states emit at least two different observations, and approximately 7% of the observations are shared between 2–18 states, further highlighting the complexity of the domain. Concretely, even if a decision support model memorizes all robot observations, it will be unable to provide accurate diagnosis or action suggestions without additional information, such as from the diagnosis and action inputs of a human or from a history of interactions with the domain.

6.3.2 Neural Network Suggestions Models

We model our two decision support models as neural networks[4] due to their expressivity and high capacity. Prior works have used neural networks to great effect in learning diagnoses [141, 140] or in learning to select actions [203, 204]; we build upon their results. We create two identical neural networks, one to provide diagnosis suggestions, D_t^R, and another for action recommendations, A_t^R (Figure 6.4).

Each of the networks is a recurrent network (RNN), in particular a Gated Recurrent Unit (GRU) [161], which conditions its output in the current timestep to the inputs in the current timestep as well as a hidden state that encodes the history of the inputs provided to the network at previous timesteps. A recurrent model is appropriate given the temporal nature of the decision support problem and the possibility of a history of inputs helping the models in providing suggestions. However, we investigate this claim in our experiments detailed in Section 6.4. Our GRU units have a hidden state dimension of 32.

At each timestep t, both the models are given the robot's observations, O_t^R, and the human's inputs—the human's previous diagnosis, D_{t-1}^H, and previous action, A_{t-1}^H, if suggesting diagnoses; the human's current diagnosis, D_t^H, and previous action, A_{t-1}^H, if suggesting actions. In all of our experiments, we evaluate the usefulness of the diagnosis and action input from the human (Section 6.4). The human inputs are encoded using shared embedding layers, and all inputs are normalized using Layer Normalization [205] before they are provided to the models.

The models provide suggestions by treating it as a classification problem: they provide a softmax probability over diagnoses or actions the human should take at each timestep. They are therefore trained using Cross-Entropy loss, and we add Dropout of 0.5 [206] and an L2 parameter weight penalty for regularization. The parameters of the models are updated with Adam [116] using a learning rate of 0.001 and with a batch size of 64. The models are trained for a maximum of 600 epochs, with early stopping based on loss on

[4]Code is available at [url].

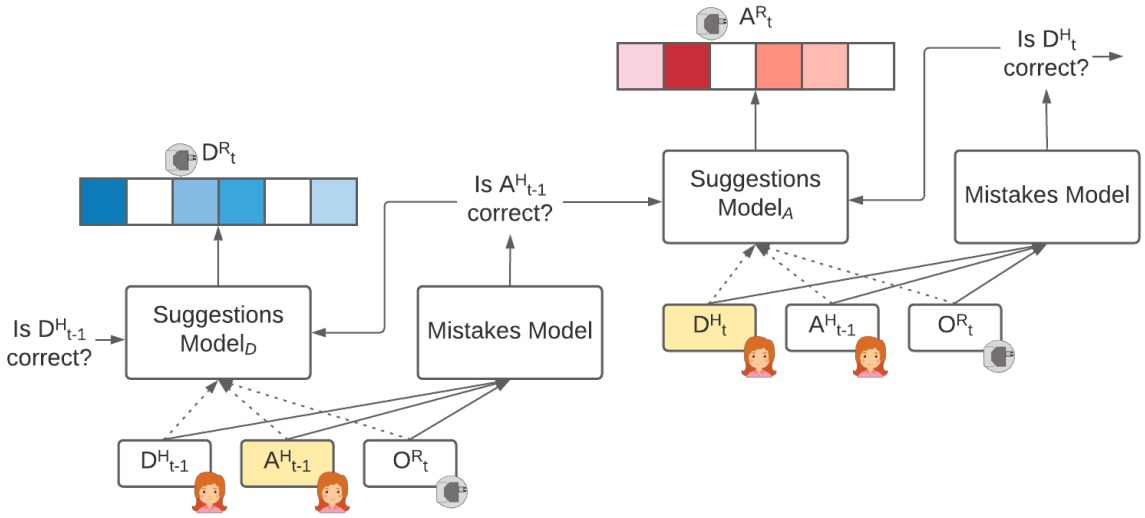


Figure 6.5: An overview of how suggestions models are augmented with a mistakes model.

a validation set—prior work has shown that early-stopping leads to more robust models when given noisy data [196] and we verified the claim in our experiments but do not show the results in this work.

6.3.3 Modeling Operator Mistakes

A central assumption of our work is that human operators make mistakes in their diagnoses, $D^H_\cdot$, and actions, $A^H_\cdot$. Therefore, in addition to learning the suggestions models, $f_\mathcal{D}$ and $f_\mathcal{A}$, we also learn a 'mistakes' model, g, which can predict if the human's last input, action or diagnosis, was a mistake, with the assumption that such an input would improve the robustness of the suggestions models (Figure 6.5). In our experiments detailed in Section 6.4, we evaluate the accuracy of the mistakes model and its usefulness in improving the robustness of the suggestions models.

Since generating labels for the mistakes model can be a time consuming burden in general, we leverage the POMDP structure defined in Section 6.2.1 to train the model. Specifically, given the features to the suggestions models, the mistakes model learns parameters to forecast the *expected value* of R_T at the presumed end of the intervention[5]. It then uses

[5]For the reader familiar with learning in the context of MDPs, we performed value-iteration; as explained in the next paragraph

the forecast of the expected value over two successive timesteps to determine if the human might have made a mistake—if the expected value does not increase, the model assumes the operator made a mistake.

Concretely, given the dataset of interventions $\Xi = \{\xi_{T_1} \dots \xi_{T_N}\}$, we use the POMDP discount factor, γ, to assign a 'value' to each timestep in ξ_i s.t. $V_{t-1} = \gamma V_t \; \forall t \in 2...T_i \; \forall i \in 1...N$, and $V_{T_i} = R_{T_i}$. In our domain, $\gamma = 0.9$ and $V_{\cdot} \in [0, 1]$, so we train the mistakes model such that $g : \mathcal{O}^R \times \mathcal{D} \times \mathcal{A} \rightarrow [0, 1]$. Given the inputs to the suggestions models at time t, the mistakes model predicts $\hat{V}_t$ and then outputs that a mistake was made if $\hat{V}_t - \hat{V}_{t-1} \leq 0$.

We used a Multi-Layer Perceptron (MLP) as our mistakes model[6] with two hidden layers, each of dimension 16, and with Gaussian Error Linear Unit (GELU) [207] non-linearities. The mistakes network was trained concurrently with the suggestions networks using methods mentioned in the previous section. In our experiments in Section 6.4, we examine the accuracy of the mistakes model (Table 6.1), and then we investigate the efficacy of the mistakes model in improving the robustness of the suggestions models (Figure 6.10).

6.3.4 Training Datasets

We created oracles to act as simulated human supervisors during an intervention in order to train our decision support models to specific levels of accuracy of the human in an intervention dataset. The oracles had access to X_t but at each timestep, an oracle could be randomly inaccurate in its diagnoses or actions, if configured to do so. Four oracles were created, with accuracies ranging from 70% to 100% accurate, in increments of 10%. Each of the oracles simulated 262 interventions, with maximum intervention length of 15 timesteps, with each simulation starting at each of the 262 non-goal states in the domain. The inaccurate oracles, 70%–90%, also repeated simulations nine additional times[7] from each non-goal state, for a total of 2620 simulated interventions from each inaccurate oracle.

[6] Recurrent models, such as the GRU, had lower mistake prediction accuracy and are therefore not discussed in this work.

[7] Repeated simulations of the 100% accurate oracle would not have generated different data.

The extra simulations were only used as training data in Experiment 2 and in test splits of cross-validation for choosing model hyperparameters. The 8122 simulated interventions were split into train, val, and test splits based on the start state of the simulation. We generated five such random splits, so that model hyperparameters and structure could be chosen using five-fold cross-validation on the test splits.

6.4 Experiments

We conduct six experiments, described in the following paragraphs, to answer our research questions (Section 6.2.3) regarding what inputs provided to models, or which techniques in unstructured machine learning, can help suggestions models improve the accuracy of suggestions while remaining robust to human error. In each experiment, we evaluate the effect of a particular input or a particular technique on model accuracy and robustness. We also evaluate the effect of using the operator's inputs directly and the effect of training dataset accuracy across all the experiments since we hypothesize that these factors are particularly important to model accuracy and robustness.

1. Recurrent vs. Non-recurrent

In this experiment, we investigate if a history of observations or operator inputs, available in recurrent models, helps improve accuracy and robustness of models over not having access to such a history. Prior work has found that recurrent models can have a smoothing effect on estimates [208, 209] and could therefore prove to be useful when learning from noisy training data. Our models, which use GRUs, are inherently recurrent. To evaluate a comparable non-recurrent model, such as an MLP, we mask the recurrent state information in the GRU[8].

[8]In additional experiments not shown here, we also trained MLP models and found the results to be identical to masking within the GRU.

2. Size of the training dataset

In this experiment, we examine the effect of the size of the training dataset. We use the additional simulated interventions from the 70%–90% oracles as training data to expand the size of the training dataset. We assume that the greater the size of the training dataset, the better the model performance.

3. Informative model inputs

In this experiment, we evaluate the necessity of choosing the correct features in $\mathcal{O}^R$, particularly features that can be more informative of failures, by introducing two conditions with new input features. Prior work often recommends the use of fault forecasting methods such as Fault Tree Analysis (FTA) [210, 141], Failure Modes and Effects Analysis (FMEA) [24, 211], etc. to determine such features. In the first condition, we add two features from $\mathcal{X}$ that simulate improved perception capabilities to $\mathcal{O}^R$: we include the state of the Jug ($X^{(J)}$)—visible with the Cup, visible without the Cup, or not visible—and the state of the Bowl ($X^{(B)}$)—visible, placed above the Cup, or not visible. In the second condition, we use $\mathcal{X}$ instead of $\mathcal{O}^R$ as features in order to investigate an upper bound to performance when perfect features are used.

4. Trusted fraction of data

In this experiment we investigate the effect of using a small subset of trusted (100% accurate) data in the training dataset in order to better guide learning and improve model robustness. Prior works often use the trusted subset of data to learn a noise model on the untrusted data, which can then be used to update suggestions [197, 198]. We use one such state-of-the-art method, Gold Loss Correction (GLC) [197], and evaluate if the better quality data or the augmented method of training improves model accuracy or robustness.

5. Using mistake estimates

In this experiment, we evaluate the accuracy of the suggestions models when the output from the mistakes model is included. First, we report the accuracy of the mistakes model, including when the input features to the model are perfect, i.e. $\mathcal{X}$, to establish an upper-bound on mistakes accuracy. Then, we report the performance of the suggestions models when the mistake estimates are included. To examine upper bounds, we also evaluate suggestions models when the mistakes models are provided with perfect features, $\mathcal{X}$, and when the mistakes models are substituted for ground-truth labels of operator mistakes.

6. Checking operator compliance with suggestions

In this experiment, we examine the effect of including features to the model that indicate if the operator followed the model's suggestions or not, i.e. operator compliance features. As discussed in the results below, we find in Table 6.2 that model accuracy is greatly influenced by the accuracy of the operator ("did they make a mistake or not?") and by their compliance with suggestions ("did they follow the suggestions or not?"). Therefore, we include two compliance features, "$A^H_{t-1} == A^R_{t-1}$" and "$D^H_{t-1} == D^R_{t-1}$", to investigate if adding them improves the performance of the suggestions models[9].

6.5 Results

We use the data gathered in Chapter 5 from 200 Amazon Mechanical Turk workers in four of the non-goal states within the domain to report our results in this section. The accuracies of the models are reported as micro-averages when tested across the five folds of cross-validation.

Additionally, our plots in Figure 6.6–Figure 6.11 highlight a model's robustness to the

[9]Models provided with the compliance features are not trained with the 100% accurate dataset because the features are meaningless in that data.

incorrect actions of operators[10]. In particular, the plots compare a model's average suggestion accuracy when an operator's action is inaccurate (X-axis) to the model's average accuracy when the operator's action is accurate (Y-axis). Models are generally more accurate when an operator is accurate (Y-axis value $>$ X-axis value), but the more robust the model, the closer its accuracy on the X-axis is to its accuracy on the Y-axis: a perfectly robust model would be plotted on the $Y = X$ line. Our plots also show dotted isoclines indicating overall accuracy, such that models on the same isocline have the same overall accuracy[11]. Additional details on the plots are available in Appendix A.

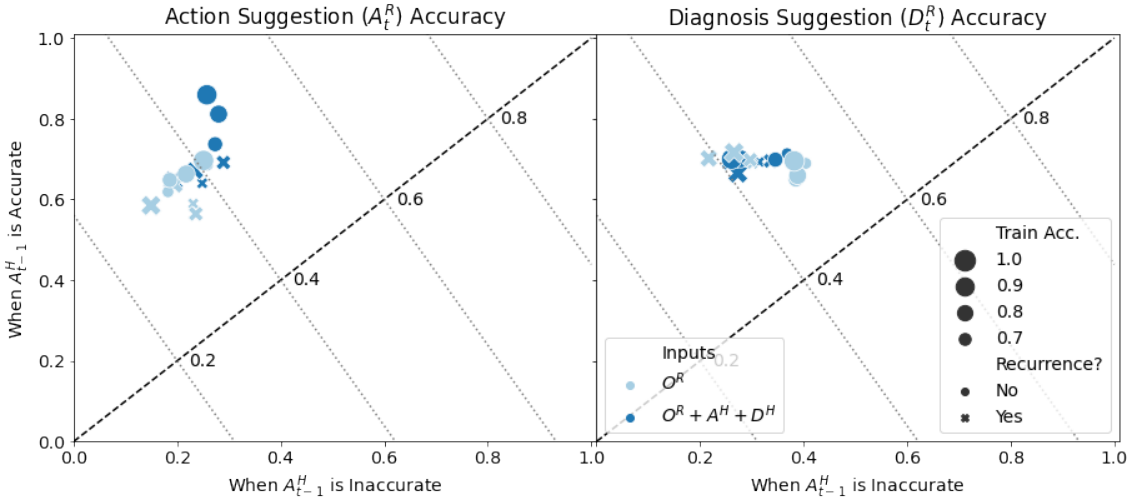


Figure 6.6: The effect of model inputs, training dataset accuracy, and the presence or absence of recurrent updates on suggestions model accuracy when A_{t-1}^H is inaccurate (X-axis) vs. when A_{t-1}^H is accurate (Y-axis).

6.5.1 Recurrent vs. Non-recurrent

Our results in Figure 6.6 show that the action suggestions model achieves the best overall accuracy of 0.47 when it is provided the human's inputs A_{t-1}^H and D_t^H, it is non-recurrent, and it is trained with high, 100% or 90%, accurate training data. However, regardless of

[10]We do not show plots of model robustness to incorrect diagnoses because errors in diagnosis had a smaller effect on model robustness than errors in actions.

[11]The overall accuracy is calculated as $acc_{overall} = N_Y * acc_Y + N_X * acc_X$, where $acc_{(\cdot)}$ is the model's accuracy overall or on the X or Y axes, N_Y is the number of times human operators were accurate in the test dataset, and N_X is the number of times they were inaccurate. As a result, the slope of the isoclines is $-\frac{N_X}{N_Y}$ instead of -1. In our evaluation data, $N_X = 5470$ and $N_Y = 3030$.

the training accuracy, *including* operator inputs leads to higher overall accuracy, 0.35–0.47 (*Mdn* 0.42), than not including them, 0.30–0.41 (*Mdn* 0.35). By contrast, the diagnosis suggestions model achieves marginally higher overall accuracy when operator inputs A^H_{t-1} and D^H_{t-1} are *excluded*, 0.39–0.51 (*Mdn* 0.46), than when they are included, 0.41–0.49 (*Mdn* 0.44). As shown in Figure 6.6, the improvement in overall accuracy is mostly a result of improved robustness to inaccurate human actions (X-axis). In fact, the diagnosis suggestions model achieves a best overall accuracy of 0.51 when it is not provided with the human's inputs, it is non-recurrent, and it is trained on 70% accurate training data. Additionally, regardless of the training accuracy, *non-recurrent* diagnosis suggestions models that do not use operator inputs achieve similar accuracy values to the best model, 0.48–0.51 (*Mdn* 0.49).

The previous results highlight the fact that recurrent models are less robust and less accurate overall than their non-recurrent counterparts for both diagnosis and action suggestions. In particular, the overall accuracies of recurrent action suggestions models and recurrent diagnosis suggestions models are in the ranges 0.30–0.43 (*Mdn* 0.36) and 0.39–0.46 (*Mdn* 0.43) respectively. By contrast, the overall accuracies of non-recurrent action suggestions models range from 0.34–0.47 (*Mdn* 0.41) and that of diagnosis suggestions models range from 0.42–0.51 (*Mdn* 0.48). The result indicates that **the mistakes made by humans in a history of actions overshadows any possible benefit that the history can have in improving suggestion accuracy.**

In the following experiments, we do not compare the accuracies of recurrent models as they are consistently lesser than the accuracies of their non-recurrent counterparts.

6.5.2 Size of the training dataset

Our results in Figure 6.7 show that with 10x more data, the overall accuracy of action suggestions is 0.47–0.52 (*Mdn* 0.50) and of diagnosis suggestions is 0.54–0.58 (*Mdn* 0.57), both of which are higher than their corresponding values in the previous experiment, 0.34–

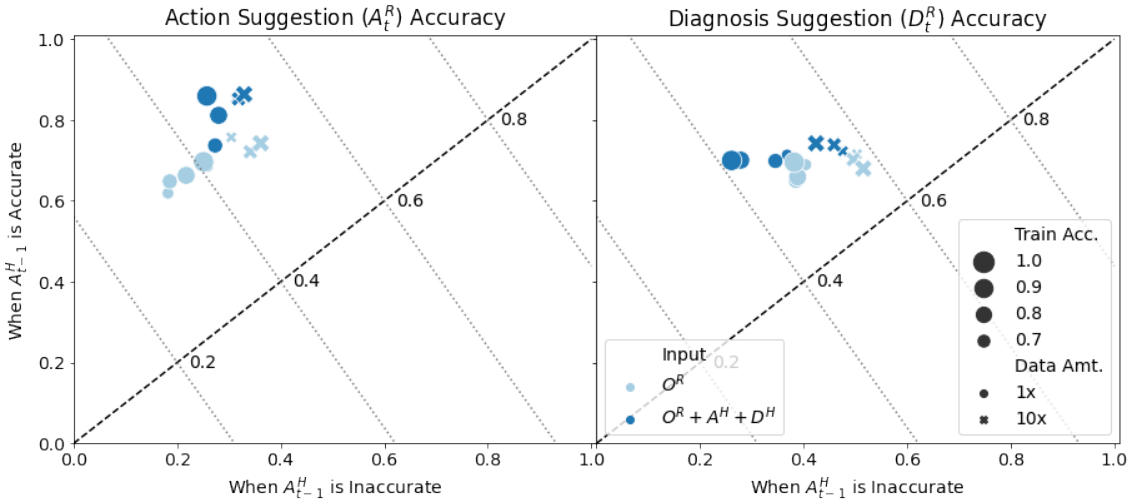


Figure 6.7: The effect of model inputs, training dataset accuracy, and the size of the training dataset on suggestions model accuracy when A_{t-1}^H is inaccurate (X-axis) vs. when A_{t-1}^H is accurate (Y-axis).

0.47 (*Mdn* 0.41) and 0.42–0.51 (*Mdn* 0.48) respectively. Interestingly, we find that the improvement to overall accuracy is achieved as a result of improved robustness. The diagnosis suggestions models in particular[12], achieve accuracies of 0.26–0.40 (*Mdn* 0.38) when trained on the smaller dataset and when the operator is inaccurate (X-axis), but manage achieve accuracies of 0.42–0.52 (*Mdn* 0.49) when trained on the larger dataset. By contrast, the accuracies of the models when the operator is accurate (Y-axis) are similar when trained on either dataset, with a range of 0.65–0.71 (*Mdn* 0.70) for models trained on the smaller dataset and a range of 0.68–0.74 (*Mdn* 0.72) for those trained on the larger dataset. The results show that **overall accuracy improves with a larger training dataset**, most likely as a result of greater data diversity and greater coverage of states and transitions within a domain.

Finally, we continue to find trends observed in the previous experiment. We find that for the models trained with 10x data, including operator inputs improves the overall accuracy of action suggestions models from 0.47–0.50 (*Mdn* 0.48) to 0.51–0.52 (*Mdn* 0.51). By contrast, excluding operator inputs improves the overall accuracy of diagnosis suggestions

[12]The improvement in overall action suggestions model accuracy is a result of improved robustness *and* improved accuracy when the operator is accurate (Y-axis).

models from 0.54–0.56 (*Mdn* 0.56) to 0.57–0.58 (*Mdn* 0.57). We also find that highly accurate training data improves accuracy of action suggestions models—the best model with 10x training data had a 90% accurate training dataset resulting in an overall accuracy of 0.52—while noisy training data improves the robustness and overall accuracy of diagnosis suggestions models—the best model with 10x training data had a 70% accurate training dataset resulting in an overall accuracy of 0.58.

In the following experiments, we use the smaller dataset to train our models in order to maintain a consistent set of reference models to compare across all the experiments.

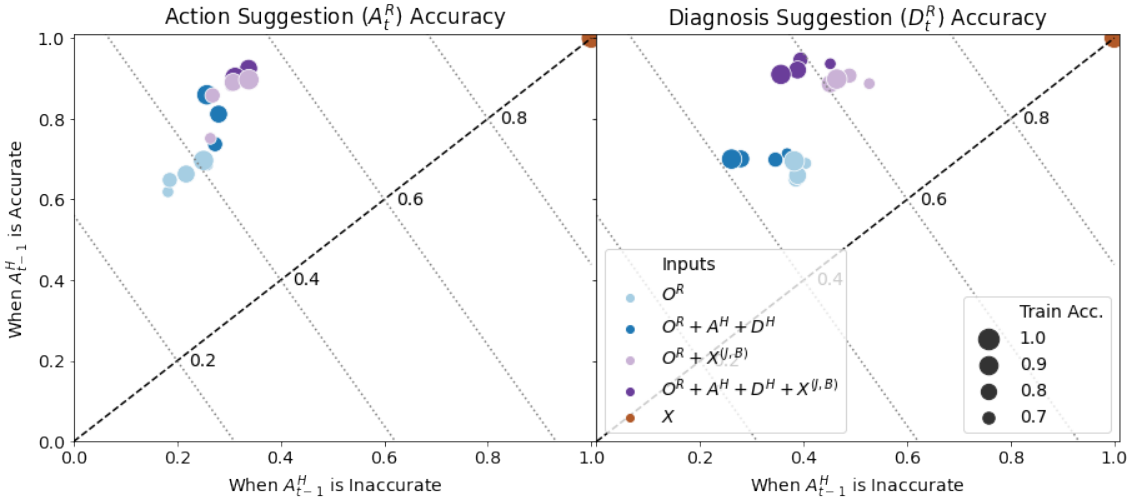


Figure 6.8: The effect of informative model inputs and training dataset accuracy on suggestions model accuracy when A_{t-1}^H is inaccurate (X-axis) vs. when A_{t-1}^H is accurate (Y-axis).

6.5.3 Informative model inputs

As shown in Figure 6.8, we find that action suggestion accuracies improve from 0.34–0.47 (*Mdn* 0.41) to 0.44–0.55 (*Mdn* 0.52) with the addition of $X_t^{(J,B)}$ to O_t^R, and diagnosis suggestions accuracies improve from 0.42–0.51 (*Mdn* 0.48) to 0.55–0.66 (*Mdn* 0.61). In fact, with perfect information, i.e. the state information X_t, the accuracies of both action and diagnosis suggestions models reach 1.0. Interestingly, we find that the overall accuracy is mostly a result of models being able to leverage human inputs better when the operator

is more accurate. The diagnosis suggestions models, in particular [12], achieve overall accuracies of 0.89–0.95 (*Mdn* 0.91) with $X_t^{(J,B)}$ and when the operator is accurate (Y-axis) compared to 0.65–0.71 (*Mdn* 0.70) without $X_t^{(J,B)}$. By contrast, the diagnosis suggestions models achieve accuracies in 0.36–0.53 (*Mdn* 0.45) with $X_t^{(J,B)}$ and when the operator is inaccurate (X-axis), which is similar to the range 0.26–0.40 (*Mdn* 0.38) achieved without $X_t^{(J,B)}$. In conclusion, we find that **informative features that can help identify a failure are crucial to improving suggestion accuracy**, likely due to the increased information provided to the models from such features. The result increases the importance of conducting systematic fault forecasting, through FMEA, FTA, etc., when developing decision support models.

We also continue to observe trends highlighted in the previous experiments. Including operator inputs for the models provided with $X_t^{(J,B)}$ still improves action suggestions accuracy from 0.44–0.54 (*Mdn* 0.50) to 0.51–0.55 (*Mdn* 0.53). Additionally, excluding operator inputs still leads to improved accuracy for diagnosis suggestions, from 0.55–0.62 (*Mdn* 0.58) to 0.61–0.66 (*Mdn* 0.63). Finally, we find that higher accuracy of training data leads to higher accuracy of action suggestions—the best model with access to $X_t^{(J,B)}$ achieved an overall accuracy of 0.55 and was trained on a 90% accurate dataset—but noisy training data improves the robustness and overall accuracy of diagnosis suggestions—the best model with access to $X_t^{(J,B)}$ achieved an overall accuracy of 0.66 and was trained on a 70% accurate dataset.

In the following experiments, we do not provide the models with the informative features of $X_t^{(J,B)}$ in order to maintain a consistent set of reference models to compare across all the experiments.

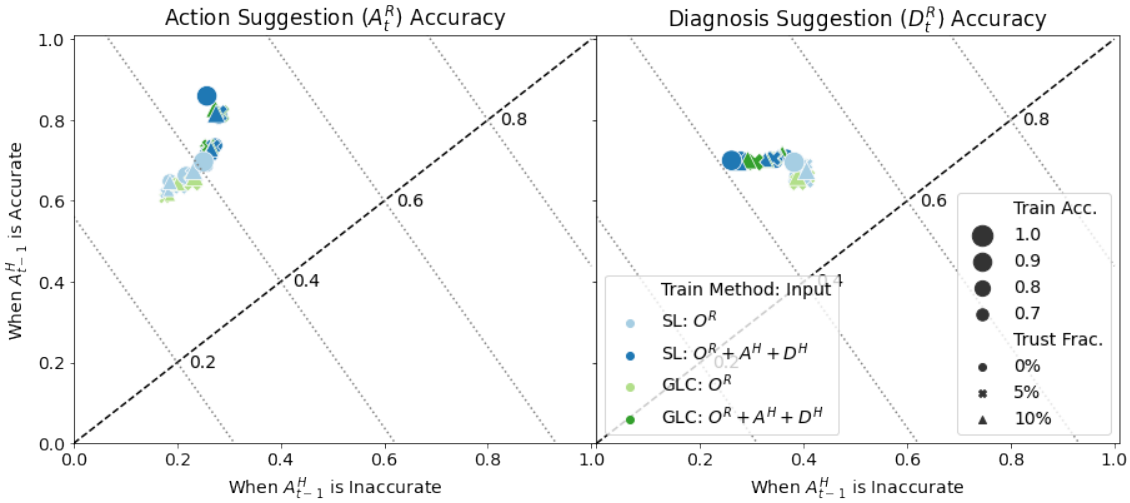


Figure 6.9: The effect of model inputs, training dataset accuracy, and the fraction of 'trusted' (100% accurate) data in the training dataset on suggestions model accuracy when A_{t-1}^H is inaccurate (X-axis) vs. when A_{t-1}^H is accurate (Y-axis).

6.5.4 Trusted fraction of data

In Figure 6.9, we find that the overall action suggestions accuracy of including 0% of a trusted subset of training data[13], 0.34–0.47 (*Mdn* 0.41), remains mostly unchanged with a 5% trusted subset, 0.33–0.47 (*Mdn* 0.41), or a 10% trusted subset, 0.34–0.47 (*Mdn* 0.40). Similarly, the overall diagnosis suggestions accuracy with a 0% trusted subset of data, 0.42–0.51 (*Mdn* 0.48), remains steady with a 5% trusted subset of data, 0.43–0.51 (*Mdn* 0.49), and a 10% trusted subset of data, 0.42–0.50 (*Mdn* 0.49). We also observe that changing the method of training to GLC [197] in order learn noise models using the trusted data does not improve suggestion accuracies. Overall action suggestion accuracy is 0.33–0.47 (*Mdn* 0.40) when trained with GLC on a 5% trusted subset of data, and 0.34–0.47 (*Mdn* 0.40) when trained with GLC on a 10% trusted subset of data. Similarly, diagnosis suggestions models trained using GLC achieving accuracies of 0.44–0.50 (*Mdn* 0.48) with a 5% trusted subset of data and 0.44–0.50 (*Mdn* 0.49) with a 10% trusted subset. The results indicate an **inability of common techniques in unstructured learning to learn from the**

[13]Models trained with 100% accurate data are coded as having a 0% trusted subset of data for consistency with previous experiments.

highly structured noise that characterizes datasets of interventions. In particular, the noise is not independent or identically distributed in such datasets, with the mistakes made by operators at one timestep, label noise, manifesting as feature noise or causing an unexpected domain shift in the features at the next timestep. Standard techniques in learning from noisy data are unable to deal with such challenges.

Since the accuracies of the suggestions models were largely unaffected by the trusted subsets of data or the manner of training, the trends observed in previous experiments on the effects of operator inputs and the effects of the accuracy of the training dataset continue to hold. In subsequent experiments, we do not include a trusted subset of data to train our models and report results.

	A^H_{t-1} **mistakes accuracy**			D^H_t **mistakes accuracy**		
Train Acc.	O^R_t	$O^R_t + A^H_{t-1} + D^H_{t-1}$	X_t	O^R_t	$O^R_t + A^H_{t-1} + D^H_t$	X_t
0.7	0.68 (0.001)	**0.57 (0.025)**	0.90 (0.023)	0.57 (0.009)	0.54 (0.013)	**0.61 (0.015)**
0.8	0.67 (0.006)	0.55 (0.015)	0.90 (0.037)	0.58 (0.013)	**0.56 (0.022)**	**0.61 (0.014)**
0.9	0.68 (0.004)	0.54 (0.027)	0.88 (0.044)	**0.59 (0.010)**	**0.56 (0.009)**	0.60 (0.009)
1.0	**0.71 (0.014)**	**0.57 (0.029)**	**0.92 (0.012)**	0.56 (0.009)	0.55 (0.010)	0.60 (0.005)

Table 6.1: Mean (and std. dev.) accuracy of the mistakes model in determining operator action or diagnosis mistakes depending on the inputs to the model and the accuracy of the training dataset. Values in bold show the highest mean accuracy for a given input to the mistakes model.

6.5.5 Using mistake estimates

In Table 6.1, we find that the mistakes model can be highly accurate in estimating a human operator's action mistakes when it is provided X_t as input: it achieves an accuracy of 0.92 in detecting the mistakes when trained with 100% accurate data. By contrast, the model struggles to detect an operator's diagnosis mistakes, achieving at most an accuracy of 0.61 in the task, even when provided with X_t. The discrepancy is likely a result of the fact that the value labels, V_t (see Section 6.3.3), that the model is trained to predict, are

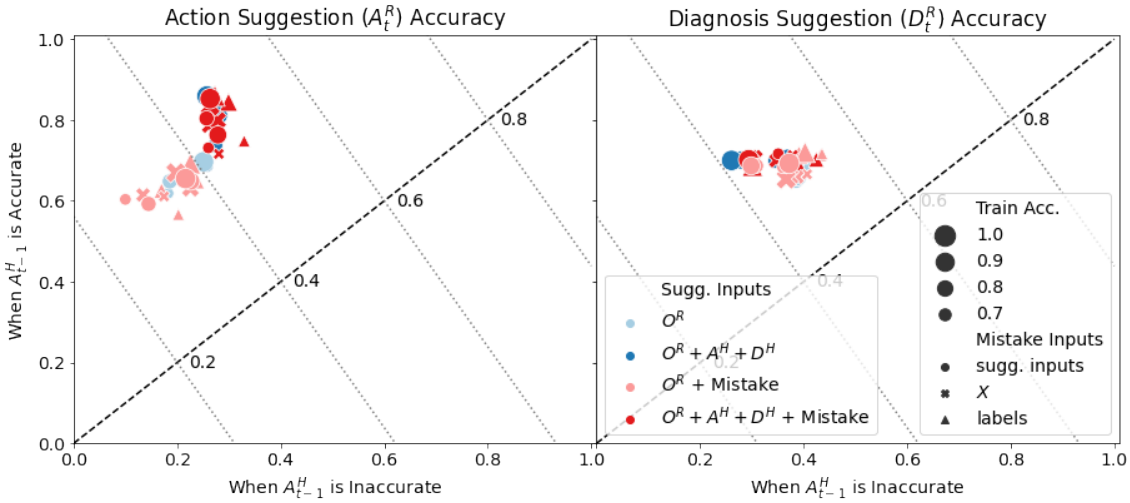


Figure 6.10: The effect of suggestions model inputs, training dataset accuracy, and the sources of mistakes data on suggestions model accuracy when A_{t-1}^H is inaccurate (X-axis) vs. when A_{t-1}^H is accurate (Y-axis).

by construction influenced more by operator actions than by diagnoses. We also observe that the inputs provided to the mistakes model greatly influences model accuracy, with the model achieving an accuracy of at most 0.57 when provided with the operator's inputs, $A_\cdot^H$ and $D_\cdot^H$, compared to 0.71 when the inputs are ommitted. Finally, we find that the accuracy of the mistakes model is improved with increased training dataset accuracy—the highest accuracy in detecting action mistakes often occurs with 100% accurate training data.

The results above inform our analysis of Figure 6.10, where we examine the accuracy of the suggestions models when provided with the mistake inputs. We find that the overall accuracy of action suggestions, 0.34–0.47 (*Mdn* 0.41), remains largely unchanged by the addition of mistakes data, 0.28–0.47 (*Mdn* 0.40), even when the mistakes model is provided with X_t as input, 0.31–0.47 (*Mdn* 0.40), or when the mistakes model is replaced with a binary 100% accurate label of mistakes, 0.33–0.49 (*Mdn* 0.43). Similarly, the overall diagnosis suggestions accuracy, 0.42–0.51 (*Mdn* 0.48), remains largely unaffected with the addition of mistakes data, 0.44–0.49 (*Mdn* 0.46), even when the mistakes model is provided with X_t as input, 0.44–0.50 (*Mdn* 0.47), or when the model is replaced with binary labels of the mistakes, 0.44–0.54 (*Mdn* 0.52). The results show that **providing suggestions models**

with data on operator mistakes does not improve accuracy, likely because the models are not provided with labels on what else to suggest in the event of a mistake.

Since the accuracies of the suggestions models were largely unaffected by the addition of the mistakes inputs, the trends observed in the previous experiments continue to hold with regards to the effects on accuracy of using operator inputs as features and of varying the training dataset accuracy. In our last experiment, when we use the mistakes model, we provide it with as input features consisting of O_t^R, $A_\cdot^H$, and $D_\cdot^H$ where appropriate, in order to eschew the use of contrived inputs to the mistakes model.

Overall Accuracy: 0.41	Acc. when A_{t-1}^H **compliant**	Acc. when A_{t-1}^H **non-compliant**
Acc. when A_{t-1}^H **accurate**	0.87 (1328)	0.56 (1702)
Acc. when A_{t-1}^H **inaccurate**	0.04 (379)	0.27 (5091)

(a) Inputs: O_t^R

Overall Accuracy: 0.47	Acc. when A_{t-1}^H **compliant**	Acc. when A_{t-1}^H **non-compliant**
Acc. when A_{t-1}^H **accurate**	0.92 (1852)	0.77 (1178)
Acc. when A_{t-1}^H **inaccurate**	0.09 (379)	0.27 (5091)

(b) Inputs: $O_t^R + A_{t-1}^H + D_t^H$

Table 6.2: The accuracy of non-recurrent *action suggestions* models trained with 100% accurate data partitioned by the accuracy of the operators's action in the previous timestep and their compliance with the model's action suggestions in the previous timestep. Numbers in parentheses indicate the number of data points in each cell: compliance is model-dependent, and hence can be different across columns for each model.

6.5.6 Checking operator compliance with suggestions

In Table 6.2, we examine the accuracy of two non-recurrent action suggestions models in greater detail. We find that the human operator's compliance with the suggestions greatly affects suggestion accuracy. In particular, a model's suggestions are *most* accurate when the

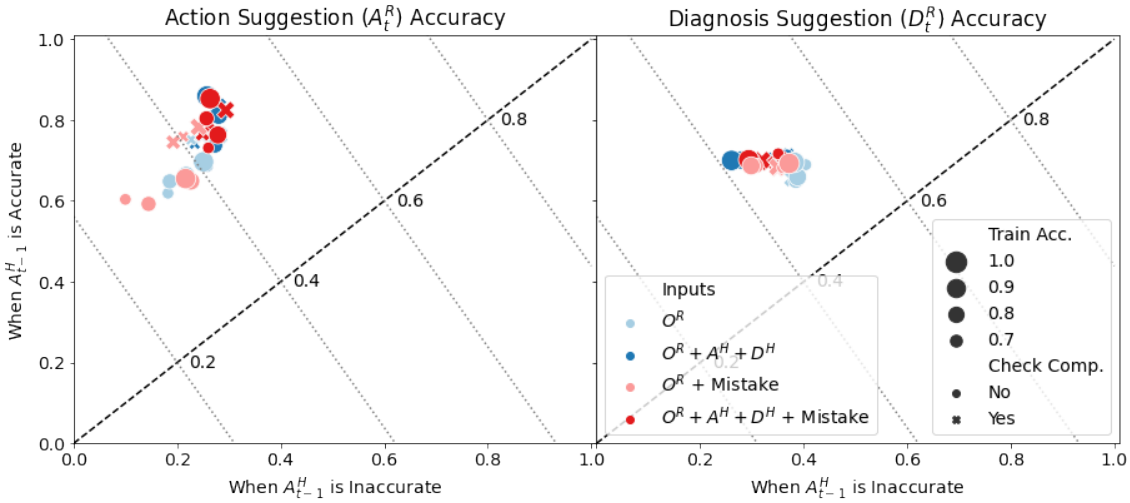


Figure 6.11: The effect of model inputs, training dataset accuracy, and the presence or absence of a operator compliance feature on suggestions model accuracy when A_{t-1}^H is inaccurate (X-axis) vs. when A_{t-1}^H is accurate (Y-axis).

operator is *accurate and compliant* with the model's suggestions, and the suggestions are *least* accurate when the operator is *inaccurate and compliant* with the model's suggestions. The trend holds for the other models not shown in Table 6.2 and the situation therefore indicates a positive feedback effect of operator compliance on model accuracy. As a result, we analyze the effects of adding 'compliance-check' features, $A_{t-1}^H == A_{t-1}^R$ and $D_{t-1}^H == D_{t-1}^R$, below.

In Figure 6.11, we find that the overall accuracy of action suggestions, 0.34–0.47 (*Mdn* 0.41), remains largely unchanged despite an improved lower bound upon adding compliance features, 0.41–0.48 (*Mdn* 0.43). The trend holds when mistake inputs are added too, with the overall accuracy of models incorporating mistakes data, 0.28–0.47 (*Mdn* 0.40), remaining steady despite an improved lower bound to model performance when compliance features are added, 0.39–0.48 (*Mdn* 0.43). Similarly, the overall accuracy of diagnosis suggestions, 0.42–0.51 (*Mdn* 0.48), remains unaffected with the addition of compliance features, 0.44–0.49 (*Mdn* 0.48). The trend again holds when mistakes data is also added to the models, with the accuracy remaining steady at 0.44–0.49 (*Mdn* 0.46) without the compliance features and at 0.46–0.48 (*Mdn* 0.47) with them. The results above show that

checking operator compliance does not improve accuracy, likely because the models are still not provided with labels on what to suggest given an operator's compliant or non-compliant behaviour. More importantly, we find that the compliance features carry redundant information.

The evidence for the latter claim is in the improvement to the lower bounds for overall action suggestions accuracy. The improvement is a result of an improved ability of models without operator inputs to leverage the compliance features when the operator is accurate (Y-axis). Specifically, the Y-axis accuracy of all models that do not use operator inputs increases from 0.59–0.70 (*Mdn* 0.65) to 0.74–0.78 (*Mdn* 0.76) with the addition of the compliance features. In fact, the accuracy achieves that of models that use operator inputs as features: 0.69–0.86 (*Mdn* 0.78) without compliance features and 0.74–0.83 (*Mdn* 0.77) with them.

Finally, we continue to observe the trends in the previous experiments with regards to the effects on accuracy of using operator inputs as features and of varying the training dataset accuracy.

6.6 Summary & Conclusions

Our goal has been to answer the two research questions introduced in Section 6.2.3, which are aimed at finding the inputs that can be provided to decision support models to improve their performance in the face of inaccurate operators (RQ1), and investigating the ability of unstructured machine learning techniques to remain accurate and robust in such settings (RQ2). In this section, we summarize our results above with respect to the research questions.

On the subject of model inputs, RQ1, we find that **the choice of features, particularly their ability to provide information about a failure, has a huge influence on decision support accuracy**. Firstly, our results in Experiment 3 show that choosing informative features improves model accuracy inspite of operator errors. The result is a strong

endorsement for conducting fault forecasting methods, such as FMEA, FTA, etc. prior to developing intervention decision support. Secondly, our results across all experiments show that the human operator's inputs, if informative of the failure, can also be of help. In particular, we find that operator input is more helpful for action suggestions than for diagnosis suggestions because an operator's current diagnosis and previous action provide *direct* information about the next action to take, while an operator's previous action and diagnosis might only provide *indirect* information about the next diagnosis. Lastly, our results in Experiment 1 show that the undue influence of operator mistakes in a history of inputs, through recurrent information, can be less informative of failures than not using such a history.

Addressing both RQ1 and RQ2, we also find that **data diversity can greatly improve decision support model accuracy**, especially by improving model robustness. The claim is best justified by the results of Experiment 2, where the diversity of data available in the larger datasets greatly improved model robustness for both diagnosis and action suggestions. However, we also find support for the claim across all of the experiments, where diagnosis suggestions achieve the highest robustness when trained with noisy, and therefore diverse, data. We suspect that more accurate, and hence less diverse, datasets are better for action suggestions because of the importance of operator inputs to action suggestions (mentioned above). Additional experiments can verify the hypothesis.

Finally, addressing RQ2, we find that **decision support models likely need to encode domain models in order to improve accuracy**, particularly when there is a lack of access to informative features or a large training dataset. Experiments 5 & 6 provide the strongest evidence for the claim, where our decision support models were unable to leverage information about operator mistakes or compliance, even when such information was perfectly accurate. However, we also find evidence in the results of Experiment 4, where a state-of-the-art method in robust learning was unable to leverage the particularly structured nature of the noise in our domain to improve the accuracy of decision support models. As men-

tioned in Section 6.1, prior work in troubleshooting assumes access to models of $\mathcal{T}$, $P_{\mathcal{O}^R}$, or P_D in the POMDP specification, and future work in developing decision support can also leverage such models.

In conclusion, in this chapter, we contribute an initial survey of considerations and techniques for training decision support models that can use human operator inputs while remaining robust to them. We find that interleaved diagnosis and recovery is a complex problem requiring collaboration between decision support models and a human in the face of significant uncertainties. With our best models achieving an accuracy of at most 0.51, we conclude that current techniques in unstructured machine learning are unable to meet the challenge of learning to assist users in such a domain. Nevertheless, we provide some initial guidelines to develop models using current methods in order to reduce the developer burden of creating decision support for remote operators handling interventions. In the next and concluding chapter of this dissertation, we use our insights to highlight additional avenues for research that can also be used to improve robot reliability while reducing the demand on humans.

CHAPTER 7
CONCLUSIONS AND FUTURE WORK

Throughout this dissertation, we have made contributions to improve robot reliability while reducing the demand on the humans who bear the responsibility of ensuring the improvement.

In Chapter 2, we formalized a development paradigm to help reduce the *frequency* of failures. Driven by our insight that recovering from failures sometimes necessitates complex responses from the robot, we designed a system that could facilitate flexibility in recovery behaviour design. In particular, our Recovery-Driven Development (RDD) approach eased developer burden for specifying such complex recoveries in order to allow rapid and iterative improvements to robot robustness.

In Chapter 3–Chapter 5, we introduced systems for gauging human interruptibility and studied the interactions between operators and decision support models during an intervention in an effort to reduce the *duration* of failures and to reduce the burden on operators and robot developers. We found that estimating a human's availability to assist a robot, through measures of interruptibility, and providing them with assistance during the intervention, through action and diagnosis suggestions, can reduce the duration. Then, in an effort to reduce developer burden of creating decision support models in Chapter 6, we found that interventions engender an interleaved diagnosis and recovery process with significant uncertainties arising from collaboration with imperfect humans and from incomplete information available to the robot. Our experiments revealed that in the absence of accurate domain models, using features that are directly informative of the failure can greatly improve the development of accurate and robust decision support models.

As such, our work (1) highlights a wide range of considerations for improving robot reliability while remaining cognizant of the human stakeholders in the process, and (2)

provides a breadth of methods and approaches that can be used to such an end. However, the insights from this work also produce open questions for future work. We discuss some of these questions below.

Extending RDD: In Chapter 2, we realized the RDD process in an executive level that sequences robot behaviours through scripting. While a scripting implementation is suitable to the *recipe-based* tasks that are commonly considered in robotics, there are many categories of tasks that do not conform to a simple recipe. Examples of such tasks include continuous monitoring tasks, which may have multiple objectives, human-robot collaborative tasks, which involve multiple agents, and social tasks, which have multiple objectives and multiple agents. In such domains, another formalism for executive design, such as state machines [46, 212] or behaviour trees [50, 213] might be more appropriate. Such executive designs are also iteratively developed and need to allow for incremental adjustments [212, 213]. Future work can therefore investigate if focusing the incremental improvements to those executive level designs through the two-prongs of RDD, *specification* and *refinement*, assists developers in improving robot reliability. Research also needs to be conducted into automatically learning task structures from data in order to reduce the effort of task specification in any of the executive level designs [214, 215].

The complexities of human interruptibility: Our research in Chapter 3 highlights the fact that many factors beyond just the social and contextual cues play a role in interruption timing: as mentioned in the chapter, even the two wizards in the WOZ condition of our study, who underwent identical training and instruction, did not entirely agree on the appropriate timing of interruptions. Some of the additional factors include differences in personality, the urgency of the task needing attention, etc. [68], and these should be explored in future research. Continuing work is also needed to explore the causal mechanisms by which robot interruptions might affect human performance [102, 87], and to model the optimal way for a robot to behave during an interruption [76].

Improvements to the quality of information in UI: In this work, we do not examine

the user experience (UX) of operators when they are notified of a failure; in the parlance of Section 6.2.1, we do not evaluate $\mathcal{O}^H$ and $P_{\mathcal{O}^H}$. However, prior work has found significant improvements to operator compliance and accuracy by changing the information available to the operator [130, 131] and by providing operators with explanations of failure [202]. Therefore, continued research is necessary to ensure that operators are provided with high-quality information in order to accurately resolve errors.

Learning better models from intervention data: Our research in Chapter 6 highlights the deficiencies of current machine learning methods in learning robust models from a dataset of human interventions. Future work can therefore explore methods to overcoming those deficiencies. For instance, future research can attempt to learn probabilistic models of an intervention domain from the data, in order to then apply the techniques developed in prior work for troubleshooting using such a domain model [136]. Alternatively, prior work has found a correspondence between scheduling and failure troubleshooting [180], thereby suggesting that structured learning approaches to scheduling [216] can perhaps prove successful in providing decision support during intervention troubleshooting. Finally, future work can also explore methods of data augmentation through counterfactual data generation and of personalizing decision support models to operator expertise [204], in order to learn robust decision support models from intervention data.

Choosing the appropriate human: In this dissertation, we have assumed that the roles and capabilities of human operators are finite, distinct, and static. However, such is rarely the case and depending on the operating environment of the robot, the roles and capabilities of operators may be infinite, overlapping, and dynamic [217]. Therefore, the problem of *choosing* the appropriate human operator given the environment, the robot error, and the assumed capabilites of human operators, remains an area of active research [39].

Considering a broader class of failures: In this dissertation, we limit ourselves to studying interventions that are generated as a result of failures in a task. However, robots deployed in the wild face threats to reliability as a result of interaction failures too [218].

Therefore, future research should evaluate methods for humans to assist robots with interaction failures, while also ensuring that such assistance does not burden the human.

Fault Detection: We assume the ability to differentiate between moments when the robot is in an error and when it is not through the use of automated fault detection; however this need not be the case. In fact, automatic and accurate fault detection is an active area of robotics research [28, 29, 30] and improved fault detection can greatly improve robot reliability by surfacing errors that might otherwise have remained undetected. Similarly, in this dissertation, we do not consider situations where an operator actively monitors the robot and is able to provide fault detection [219], which can then be used for automated failure recoveries. Future work can further explore the link between improving reliability and the manner of detecting failures.

Ethics & Trust: This dissertation does not address the large body of work on trust and ethics that might be relevant to the goal of improving robot reliability under human supervision. As one example, issues of trust and of framing the robot failure could greatly impact both the duration of a current failure and the frequency with which operators might be willing to help in the future [220]. As another example, inappropriate use of metrics for reliability, such as Mean-Time Completing Interventions (MTCI), could also place unethical stress on human operators [221]. And finally, the machine learning methods used in this work are susceptible to learning inappropriate biases through their datasets [222, 223]. As such, continuing work from this dissertation should also look to address these additional concerns.

The objective of this dissertation, as with all research in robotics, is to transition robots from the laboratory and into the real-world so that they are able to operate in *that* environment reliably. However, this dissertation also considers the effects of such a transition on the human stakeholders in the process. By building upon the contributions of this dissertation, we can move towards a world where robots are operating reliably in real-world environments without making unreasonable demands on humans.

Appendices

APPENDIX A

UNDERSTANDING ACCURACY-INACCURACY PLOTS

We provide here an intuition to understanding the results presented in the Accuracy-Inaccuracy plots in Section 6.5. The plots are used to compare the performance of different models on a dataset of human interventions: we use a toy example that is pictorially depicted in Figure A.1 to show how this is done. Note that we ignore time index matching, i.e. evaluating model accuracy at time t based on operator accuracy at $t-1$, for ease of illustration.

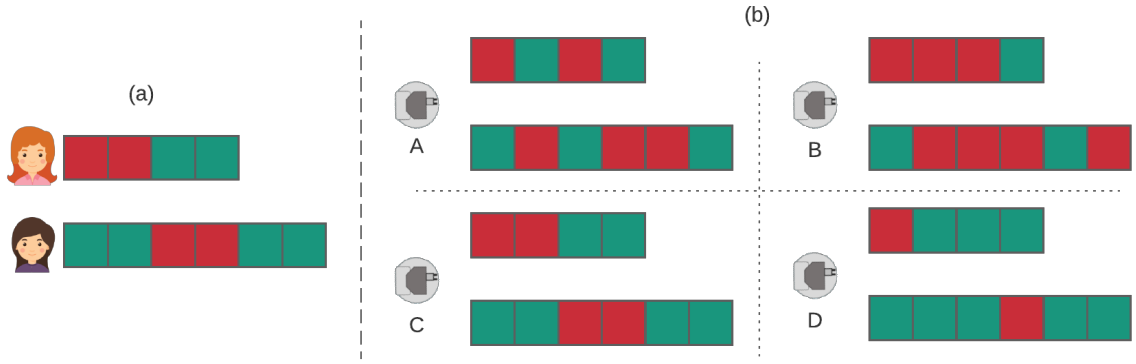


Figure A.1: A toy dataset of actions taken by operators or suggested by models during two hypothetical interventions; red is an incorrect action (or action suggestion) and green is a correct action (or action suggestion). (a) Assumed human operator actions. (b) Assumed suggestions from four different decision support models evaluated on the dataset.

The toy dataset consists of 10 (assumed) actions taken by two human operators over the course of two interventions. Figure A.1 shows whether each of the actions was correct or not with a green or red colour. As such, the humans display an accuracy of 0.67 in their action selection. We also assume four toy models, A–D, that showcase varying suggestion accuracies when evaluated on data from the same interventions. The performance of the models is summarized in Table A.1.

The Accuracy-Inaccuracy plots are meant to pictorially depict the information in the above table in order to facilitate an easier comparison of the models. Figure A.2(a) high-

Model	Overall Accuracy	Acc. when operator accurate (Y-axis)	Acc. when operator inaccurate (X-axis)
A	0.5	0.5	0.5
B	0.3	0.5	0.0
C	0.6	1.0	0.0
D	0.8	1.0	0.5

Table A.1: Model performance within the toy example.

lights some of the key properties of the plot:

1. Model robustness increases as X-axis accuracy increases and model peak accuracy increases as Y-axis accuracy increases.

2. Models on the X=Y line are unaffected by the accuracy of the operator.

3. Models on ths same dotted isoclines, which have a slope of -1.33 in our toy example, have the same overall accuracy but might be differently affected by operator performance.

The properties are exemplified in Figure A.2(b), which situates the models from Table A.1 within the Accuracy-Inaccuracy space.

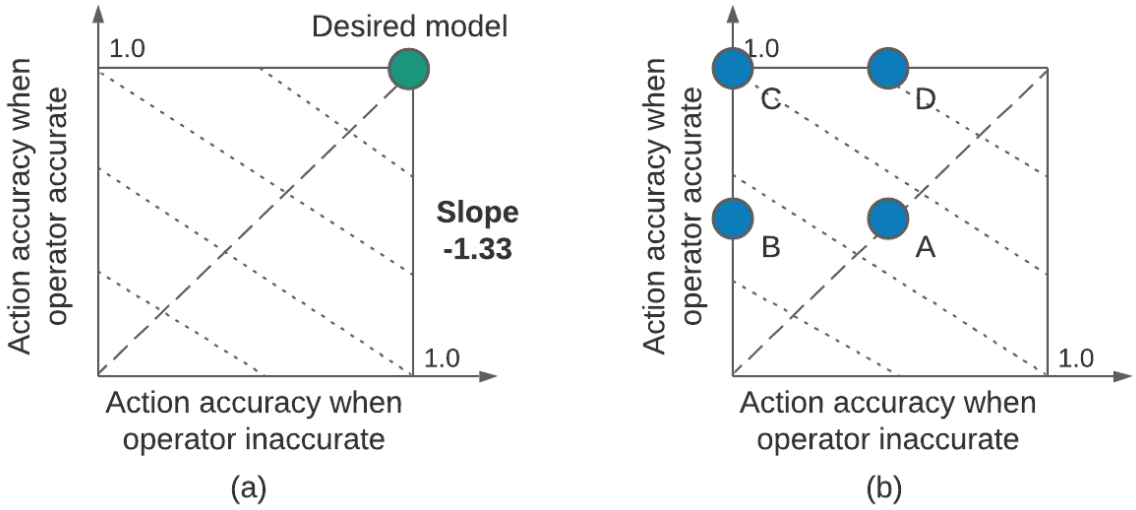


Figure A.2: Toy Accuracy-Inaccuracy plots for evaluating model performance. (a) The Accuracy-Inaccuracy space from the toy example in Figure A.1. (b) The performance of the four models from that figure within this space.

REFERENCES

[1] J. Forlizzi and C. DiSalvo, "Service robots in the domestic environment: A study of the roomba vacuum in the home," in *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, 2006, pp. 258–265.

[2] J. R. Orejana, B. A. MacDonald, H. S. Ahn, K. Peri, and E. Broadbent, "Healthcare robots in homes of rural older adults," in *International Conference on Social Robotics*, Springer, 2015, pp. 512–521.

[3] E. J. Carter, S. Reig, X. Z. Tan, G. Laput, S. Rosenthal, and A. Steinfeld, "Death of a robot: Social media reactions and language usage when a robot stops operating," in *Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*, 2020, pp. 589–597.

[4] A. G. Ozkil, Z. Fan, S. Dawids, H. Aanes, J. K. Kristensen, and K. H. Christensen, "Service robots for hospitals: A case study of transportation tasks in a hospital," in *2009 IEEE international conference on automation and logistics*, IEEE, 2009, pp. 289–294.

[5] T. Mettler, M. Sprenger, and R. Winter, "Service robots in hospitals: New perspectives on niche evolution and technology affordances," *European Journal of Information Systems*, vol. 26, no. 5, pp. 451–468, 2017.

[6] Y. Choi, M. Choi, M. Oh, and S. Kim, "Service robots in hotels: Understanding the service quality perceptions of human-robot interaction," *Journal of Hospitality Marketing & Management*, vol. 29, no. 6, pp. 613–635, 2020.

[7] S. Satake, K. Hayashi, K. Nakatani, and T. Kanda, "Field trial of an information-providing robot in a shopping mall," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, Sep. 2015, pp. 1832–1839.

[8] M. Niemelä, P. Heikkilä, H. Lammi, and V. Oksman, "Shopping mall robots–opportunities and constraints from the retailer and manager perspective," in *International Conference on Social Robotics*, Springer, 2017, pp. 485–494.

[9] M. Schneier and R. Bostelman, *Literature review of mobile robots for manufacturing*. US Department of Commerce, National Institute of Standards and Technology . . ., 2015.

[10] P. Sapaty, "Military robotics: Latest trends and spatial grasp solutions," *International Journal of Advanced Research in Artificial Intelligence*, vol. 4, no. 4, pp. 9–18, 2015.

[11] M. Ai-Chang, J. Bresina, L. Charest, A. Chase, J.-J. Hsu, A. Jonsson, B. Kanefsky, P. Morris, K. Rajan, J. Yglesias, *et al.*, "Mapgen: Mixed-initiative planning and scheduling for the mars exploration rover mission," *IEEE Intelligent Systems*, vol. 19, no. 1, pp. 8–12, 2004.

[12] R. Welch, D. Limonadi, and R. Manning, "Systems engineering the curiosity rover: A retrospective," in *2013 8th international conference on system of systems engineering*, IEEE, 2013, pp. 70–75.

[13] J. Maki, D. Gruel, C. McKinney, M. Ravine, M. Morales, D. Lee, R. Willson, D. Copley-Woods, M. Valvo, T. Goodsall, *et al.*, "The mars 2020 engineering cameras and microphone on the perseverance rover: A next-generation imaging system for mars exploration," *Space Science Reviews*, vol. 216, no. 8, pp. 1–48, 2020.

[14] R. Bloss, "Mobile hospital robots cure numerous logistic needs," *Industrial Robot: An International Journal*, 2011.

[15] J. Hu, A. Edsinger, Y.-J. Lim, N. Donaldson, M. Solano, A. Solochek, and R. Marchessault, "An advanced medical robotic system augmenting healthcare capabilities-robotic nursing assistant," in *2011 IEEE international conference on robotics and automation*, IEEE, 2011, pp. 6264–6269.

[16] M. Hägele, K. Nilsson, J. N. Pires, and R. Bischoff, "Industrial robotics," in *Springer handbook of robotics*, Springer, 2016, pp. 1385–1422.

[17] J. Lawton, "Collaborative robots," *International Society of Automation*, pp. 12–14, 2016.

[18] D. Bohus and E. Horvitz, "Dialog in the open world," in *Proceedings of the 2009 international conference on Multimodal interfaces - ICMI-MLMI '09*, New York, New York, USA: ACM Press, 2009, p. 31.

[19] D. Brščić, T. Ikeda, and T. Kanda, "Do you need help? a robot providing information to people who behave atypically," *IEEE Transactions on Robotics*, vol. 33, no. 2, pp. 500–506, 2017.

[20] R. R. Murphy, *Disaster robotics*. MIT press, 2014.

[21] Z. H. Khan, A. Siddique, and C. W. Lee, "Robotics utilization for healthcare digitization in global covid-19 management," *International journal of environmental research and public health*, vol. 17, no. 11, p. 3819, 2020.

[22] R. R. Murphy, V. B. M. Gandudi, and J. Adams, "Applications of robots for covid-19 response," *arXiv preprint arXiv:2008.06976*, 2020.

[23] J. Guiochet, M. Machin, and H. Waeselynck, "Safety-critical advanced robots: A survey," *Robotics and Autonomous Systems*, vol. 94, pp. 43–52, Aug. 2017.

[24] D. Crestani, K. Godary-Dejean, and L. Lapierre, "Enhancing fault tolerance of autonomous mobile robots," *Robotics and Autonomous Systems*, vol. 68, pp. 140–155, Jun. 2015.

[25] A. Sauppé and B. Mutlu, "The Social Impact of a Robot Co-Worker in Industrial Settings," in *CHI*, ACM Press, 2015, pp. 3613–3622.

[26] J. M. Beer, A. D. Fisk, and W. A. Rogers, "Toward a framework for levels of robot autonomy in human-robot interaction," *Journal of human-robot interaction*, vol. 3, no. 2, pp. 74–99, 2014.

[27] B. Mutlu and J. Forlizzi, "Robots in organizations," in *Proceedings of the 3rd international conference on Human robot interaction - HRI '08*, New York, New York, USA: ACM Press, 2008, p. 287.

[28] E. Khalastchi and M. Kalech, "On Fault Detection and Diagnosis in Robotic Systems," *ACM Computing Surveys*, vol. 51, no. 1, pp. 1–24, Jan. 2018.

[29] D. Park, Y. Hoshi, and C. C. Kemp, "A Multimodal Anomaly Detector for Robot-Assisted Feeding Using an LSTM-Based Variational Autoencoder," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1544–1551, Jul. 2018.

[30] R. Hornung, H. Urbanek, J. Klodmann, C. Osendorfer, and P. van der Smagt, "Model-free robot anomaly detection," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, Sep. 2014, pp. 3676–3683.

[31] J. A. Shah, J. H. Saleh, and J. A. Hoffman, "Analytical basis for evaluating the effect of unplanned interventions on the effectiveness of a human–robot system," *Reliability Engineering & System Safety*, vol. 93, no. 8, pp. 1280–1286, Aug. 2008.

[32] M. Johnson, J. M. Bradshaw, P. J. Feltovich, C. M. Jonker, M. B. Van Riemsdijk, and M. Sierhuis, "Coactive Design: Designing Support for Interdependence in Joint Activity," *Journal of Human-Robot Interaction*, vol. 3, no. 1, p. 43, Mar. 2014.

[33] M. Johnson, B. Shrewsbury, S. Bertrand, T. Wu, D. Duran, M. Floyd, P. Abeles, D. Stephen, N. Mertins, A. Lesman, J. Carff, W. Rifenburgh, P. Kaveti, W. Straatman, J. Smith, M. Griffioen, B. Layton, T. de Boer, T. Koolen, P. Neuhaus, and J. Pratt, "Team IHMC's Lessons Learned from the DARPA Robotics Challenge Trials," *Journal of Field Robotics*, vol. 32, no. 2, pp. 192–208, Mar. 2015.

[34] M. DeDonato, V. Dimitrov, R. Du, R. Giovacchini, K. Knoedler, X. Long, F. Polido, M. A. Gennert, T. Padır, S. Feng, H. Moriguchi, E. Whitman, X. Xinjilefu, and

C. G. Atkeson, "Human-in-the-loop Control of a Humanoid Robot for Disaster Response: A Report from the DARPA Robotics Challenge Trials," *Journal of Field Robotics*, vol. 32, no. 2, pp. 275–292, Mar. 2015.

[35] C. G. Atkeson, B. P. W. Babu, N. Banerjee, D. Berenson, C. P. Bove, X. Cui, M. DeDonato, R. Du, S. Feng, P. Franklin, M. Gennert, J. P. Graff, P. He, A. Jaeger, J. Kim, K. Knoedler, L. Li, C. Liu, X. Long, T. Padir, F. Polido, G. G. Tighe, and X. Xinjilefu, "No falls, no resets: Reliable humanoid behavior in the DARPA robotics challenge," in *Humanoids*, IEEE, Nov. 2015, pp. 623–630.

[36] H. A. Yanco, A. Norton, W. Ober, D. Shane, A. Skinner, and J. Vice, "Analysis of Human-robot Interaction at the DARPA Robotics Challenge Trials," *Journal of Field Robotics*, vol. 32, no. 3, pp. 420–444, May 2015.

[37] DRC-Teams, *What Happened at the DARPA Robotics Challenge?* 2015.

[38] S. McGuire, P. M. Furlong, C. Heckman, S. Julier, D. Szafir, and N. Ahmed, "Failure is Not an Option: Policy Learning for Adaptive Recovery in Space Operations," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1639–1646, Jul. 2018.

[39] S. McGuire, P. M. Furlong, T. Fong, C. Heckman, D. Szafir, S. J. Julier, and N. Ahmed, "Everybody needs somebody sometimes: Validation of adaptive recovery in robotic space operations," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1216–1223, 2019.

[40] S. Banerjee, A. Daruna, D. Kent, W. Liu, J. Balloch, A. Jain, A. Krishnan, M. A. Rana, H. Ravichandar, B. Shah, N. Shrivatsav, and S. Chernova, "Taking recoveries to task: Recovery-driven development for recipe-based robot tasks," *ISRR*, 2019.

[41] S. Banerjee and S. Chernova, "Temporal Models for Robot Classification of Human Interruptibility," in *Int. Conf. on Autonomous Agents & Multiagent Systems*, IFAAMAS, 2017, pp. 1350–1359.

[42] S. Banerjee, A. Silva, K. Feigh, and S. Chernova, "Effects of interruptibility-aware robot behavior," *arXiv preprint arXiv:1804.06383*, 2018.

[43] S. Banerjee, M. Gombolay, and S. Chernova, "A tale of two suggestions: Action and diagnosis recommendations for responding to robot failure," in *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, IEEE, pp. 398–405.

[44] S. S. Srinivasa, D. Berenson, M. Cakmak, A. Collet, M. R. Dogar, A. D. Dragan, R. A. Knepper, T. Niemueller, K. Strabala, M. Vande Weghe, and J. Ziegler, "Herb 2.0: Lessons Learned From Developing a Mobile Manipulator for the Home," *Proceedings of the IEEE*, vol. 100, no. 8, pp. 2410–2428, Aug. 2012.

[45] S. Lemaignan, A. Hosseini, and P. Dillenbourg, "PYROBOTS, a toolset for robot executive control," in *IROS*, IEEE, Sep. 2015, pp. 2848–2853.

[46] J. Bohren and S. Cousins, "The SMACH High-Level Executive [ROS News]," *Robotics & Automation Magazine*, vol. 17, no. 4, pp. 18–20, Dec. 2010.

[47] C. Eppner, S. Höfer, R. Jonschkowski, R. Martín-Martín, A. Sieverling, V. Wall, and O. Brock, "Lessons from the amazon picking challenge: Four aspects of building robotic systems.," in *Robotics: Science and Systems*, 2016.

[48] M. Beetz, L. Mösenlechner, and M. Tenorth, "CRAM x2014; A Cognitive Robot Abstract Machine for everyday manipulation in human environments," in *Int. Conf. on Intelligent Robots and Systems*, IEEE, Oct. 2010, pp. 1012–1017.

[49] D. Kortenkamp, R. Simmons, and D. Brugali, "Robotic Systems Architectures and Programming," in *Springer Handbook of Robotics*, 2016, pp. 283–306.

[50] V. Berenz and S. Schaal, "The Playful Software Platform: Reactive Programming for Orchestrating Robotic Behavior," *Robotics & Automation Magazine*, vol. 25, no. 3, pp. 49–60, Sep. 2018.

[51] D. Szafir, B. Mutlu, and T. Fong, "Designing planning and control interfaces to support user collaboration with flying robots," *Int. Journal of Robotics Research*, vol. 36, no. 5-7, pp. 514–542, Jun. 2017.

[52] A. Degroote and S. Lacroix, "ROAR: Resource oriented agent architecture for the autonomy of robots," in *ICRA*, IEEE, May 2011, pp. 6090–6095.

[53] M. Klotzbücher, P. Soetens, and H. Bruyninckx, "Orocos rtt-lua: an execution environment for building real-time robotic domain specific languages," in *Int. Workshop on Dynamic languages for Robotics and Sensors*, vol. 8, 2010.

[54] H. Kress-Gazit, G. Fainekos, and G. Pappas, "Temporal-Logic-Based Reactive Mission and Motion Planning," *Transactions on Robotics*, vol. 25, no. 6, pp. 1370–1381, Dec. 2009.

[55] S. Zaman, G. Steinbauer, J. Maurer, P. Lepej, and S. Uran, "An integrated model-based diagnosis and repair architecture for ROS-based robot systems," in *ICRA*, IEEE, May 2013, pp. 482–489.

[56] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: An open-source robot operating system," in *ICRA workshop on open source software*, Kobe, Japan, vol. 3, 2009, p. 5.

[57] R. B. Rusu and S. Cousins, "Point cloud library (pcl)," in *ICRA*, 2011, pp. 1–4.

[58] W. Wohlkinger and M. Vincze, "Ensemble of shape functions for 3d object classification," in *Int. Conf. on Robotics and Biomimetics*, IEEE, 2011, pp. 2987–2992.

[59] S. Chitta, I. Sucan, and S. Cousins, "Moveit![ros topics]," *Robotics & Automation Magazine*, vol. 19, no. 1, pp. 18–19, 2012.

[60] A. ten Pas and R. Platt, "Using geometry to detect grasp poses in 3d point clouds," in *Robotics Research*, Springer, 2018, pp. 307–324.

[61] D. Kent and R. Toris, "Adaptive autonomous grasp selection via pairwise ranking," in *IROS*, IEEE, 2018, pp. 2971–2976.

[62] B. Akgun, M. Cakmak, J. W. Yoo, and A. L. Thomaz, "Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective," in *HRI*, ACM, 2012, pp. 391–398.

[63] M. Wise, M. Ferguson, D. King, E. Diehr, and D. Dymesich, "Fetch and freight: Standard platforms for service robot applications," in *Workshop on Autonomous Mobile Service Robots*, 2016.

[64] C. Speier, J. S. Valacich, and I. Vessey, "The effects of task interruption and information presentation on individual decision making," in *Proceedings of the eighteenth international conference on Information systems*, Association for Information Systems, 1997, pp. 21–36.

[65] D. McFarlane and K. Latorella, "The Scope and Importance of Human Interruption in Human-Computer Interaction Design," *Human-Computer Interaction*, vol. 17, no. 1, pp. 1–61, Mar. 2002.

[66] G. Mark, D. Gudith, and U. Klocke, "The cost of interrupted work," in *Proceeding of the twenty-sixth annual CHI conference on Human factors in computing systems - CHI '08*, New York, New York, USA: ACM Press, 2008, p. 107.

[67] N. B. Sarter, "Multimodal Support for Interruption Management: Models, Empirical Findings, and Design Recommendations," *Proceedings of the IEEE*, vol. 101, no. 9, pp. 2105–2112, Sep. 2013.

[68] A. J. Rivera, "A socio-technical systems approach to studying interruptions: Understanding the interrupter's perspective," *Applied Ergonomics*, vol. 45, no. 3, pp. 747–756, May 2014.

[69] Y. Miyata and D. A. Norman, "Psychological issues in support of multiple activities," *User centered system design: New perspectives on human-computer interaction*, pp. 265–284, 1986.

[70] H. Stern, V. Pammer, and S. N. Lindstaedt, "A preliminary study on interruptibility detection based on location and calendar information," *Proc. CoSDEO*, vol. 11, 2011.

[71] L. D. Turner, S. M. Allen, and R. M. Whitaker, "Interruptibility prediction for ubiquitous systems," in *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing - UbiComp '15*, New York, New York, USA: ACM Press, 2015, pp. 801–812.

[72] K. Fischer, B. Soto, C. Pantofaru, and L. Takayama, "Initiating interactions in order to get help: Effects of social framing on people's responses to robots' requests for assistance," in *Robot and Human Interactive Communication, 2014 RO-MAN: The 23rd IEEE International Symposium on*, IEEE, 2014, pp. 999–1005.

[73] S. Rosenthal, M. M. Veloso, and A. K. Dey, "Is Someone in this Office Available to Help Me?" *Journal of Intelligent & Robotic Systems*, vol. 66, no. 1-2, pp. 205–221, Apr. 2012.

[74] D. C. McFarlane, "Comparison of Four Primary Methods for Coordinating the Interruption of People in Human-Computer Interaction," *Human-Computer Interaction*, vol. 17, no. 1, pp. 63–139, Mar. 2002.

[75] C. A. Monk, D. A. Boehm-Davis, G. Mason, and J. G. Trafton, "Recovering From Interruptions: Implications for Driver Distraction Research," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 46, no. 4, pp. 650–663, Dec. 2004.

[76] P. Saulnier, E. Sharlin, and S. Greenberg, "Exploring minimal nonverbal interruption in HRI," in *2011 RO-MAN*, IEEE, Jul. 2011, pp. 79–86.

[77] Y.-S. Chiang, T.-S. Chu, C. D. Lim, T.-Y. Wu, S.-H. Tseng, and L.-C. Fu, "Personalizing robot behavior for interruption in social human-robot interaction," in *2014 IEEE International Workshop on Advanced Robotics and its Social Impacts*, IEEE, Sep. 2014, pp. 44–49.

[78] C. Mollaret, A. Mekonnen, F. Lerasle, I. Ferrané, J. Pinquier, B. Boudet, and P. Rumeau, "A multi-modal perception based assistive robotic system for the elderly," *Computer Vision and Image Understanding*, Mar. 2016.

[79] A. Nigam and L. D. Riek, "Social context perception for mobile robots," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, Sep. 2015, pp. 3621–3627.

[80] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[81] G. E. Hinton, "Connectionist learning procedures," in *Machine Learning, Volume III*, Elsevier, 1990, pp. 555–610.

[82] L.-P. Morency, A. Quattoni, and T. Darrell, "Latent-Dynamic Discriminative Models for Continuous Gesture Recognition," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, Jun. 2007, pp. 1–8.

[83] S. K. Card, T. P. Moran, and A. Newell, "Computer text-editing: An information-processing analysis of a routine cognitive skill," *Cognitive Psychology*, vol. 12, no. 1, pp. 32–74, Jan. 1980.

[84] P. D. Adamczyk and B. P. Bailey, "If not now, when?" In *Proceedings of the 2004 conference on Human factors in computing systems - CHI '04*, New York, New York, USA: ACM Press, 2004, pp. 271–278.

[85] S. T. Iqbal and B. P. Bailey, "Leveraging characteristics of task structure to predict the cost of interruption," in *Proceedings of the SIGCHI conference on Human Factors in computing systems - CHI '06*, New York, New York, USA: ACM Press, 2006, p. 741.

[86] J. G. Trafton, A. Jacobs, and A. M. Harrison, "Building and Verifying a Predictive Model of Interruption Resumption," *Proceedings of the IEEE*, vol. 100, no. 3, pp. 648–659, Mar. 2012.

[87] G. Trafton, L. Hiatt, A. Harrison, F. Tanborello, S. Khemlani, and A. Schultz, "ACT-R/E: An Embodied Cognitive Architecture for Human-Robot Interaction," *Journal of Human-Robot Interaction*, vol. 2, no. 1, pp. 30–55, Mar. 2013.

[88] S. Satake, T. Kanda, D. F. Glas, M. Imai, H. Ishiguro, and N. Hagita, "How to approach humans?" In *Proceedings of the 4th ACM/IEEE international conference on Human robot interaction - HRI '09*, New York, New York, USA: ACM Press, 2009, p. 109.

[89] Y. Kato, T. Kanda, and H. Ishiguro, "May I help you?" In *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction - HRI '15*, New York, New York, USA: ACM Press, 2015, pp. 35–42.

[90] C. Shi, M. Shiomi, T. Kanda, H. Ishiguro, and N. Hagita, "Measuring Communication Participation to Initiate Conversation in Human–Robot Interaction," *International Journal of Social Robotics*, vol. 7, no. 5, pp. 889–910, Nov. 2015.

[91] M. E. Foster, A. Gaschler, and M. Giuliani, "Automatically Classifying User Engagement for Dynamic Multi-party Human-Robot Interaction," *International Journal of Social Robotics*, Jul. 2017.

[92] E. S. Short, M. L. Chang, and A. Thomaz, "Detecting Contingency for HRI in Open-World Environments," in *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction - HRI '18*, New York, New York, USA: ACM Press, 2018, pp. 425–433.

[93] A. Garrell, M. Villamizar, F. Moreno-Noguer, and A. Sanfeliu, "Teaching Robot's Proactive Behavior Using Human Assistance," *International Journal of Social Robotics*, vol. 9, no. 2, pp. 231–249, Apr. 2017.

[94] V. Chu, K. Bullard, and A. L. Thomaz, "Multimodal real-time contingency detection for HRI," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, Sep. 2014, pp. 3327–3332.

[95] E. R. Sykes, "A Cloud-based Interaction Management System Architecture for Mobile Devices," *Procedia Computer Science*, vol. 34, pp. 625–632, 2014.

[96] J. Fogarty, S. E. Hudson, C. G. Atkeson, D. Avrahami, J. Forlizzi, S. Kiesler, J. C. Lee, and J. Yang, "Predicting human interruptibility with sensors," *ACM Transactions on Computer-Human Interaction*, vol. 12, no. 1, pp. 119–146, Mar. 2005.

[97] E. Horvitz and J. Apacible, "Learning and reasoning about interruption," in *Proceedings of the 5th international conference on Multimodal interfaces - ICMI '03*, New York, New York, USA: ACM Press, 2003, p. 20.

[98] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *International journal of computer vision*, vol. 42, no. 3, pp. 145–175, 2001.

[99] A. Kolbeinsson, P. Thorvald, and J. Lindblom, "Coordinating the interruption of assembly workers in manufacturing," *Applied Ergonomics*, vol. 58, pp. 361–371, Jan. 2017.

[100] T. Grundgeiger, D. Liu, P. Sanderson, S. Jenkins, and T. Leane, "Effects of Interruptions on Prospective Memory Performance in Anesthesiology," *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 52, no. 12, pp. 808–812, Sep. 2008.

[101] B. C. Lee and V. G. Duffy, "The Effects of Task Interruption on Human Performance: A Study of the Systematic Classification of Human Behavior and Interruption Frequency," *Human Factors and Ergonomics in Manufacturing & Service Industries*, vol. 25, no. 2, pp. 137–152, Mar. 2015.

[102] T. Grundgeiger and P. Sanderson, "Interruptions in healthcare: Theoretical views," *International Journal of Medical Informatics*, vol. 78, no. 5, pp. 293–307, May 2009.

[103] P. M. Sanderson and T. Grundgeiger, "How do interruptions affect clinician performance in healthcare? Negotiating fidelity, control, and potential generalizability in the search for answers," *International Journal of Human-Computer Studies*, vol. 79, pp. 85–96, Jul. 2015.

[104] D. A. Epstein, D. Avrahami, and J. T. Biehl, "Taking 5: Work-Breaks, Productivity, and Opportunities for Personal Informatics for Knowledge Workers," in *CHI '16*, San Jose, CA: ACM Press, 2016.

[105] E. M. Altmann and J. G. Trafton, "Memory for goals: an activation-based model," *Cognitive Science*, vol. 26, no. 1, pp. 39–83, Jan. 2002.

[106] M. A. McDaniel and G. O. Einstein, "Strategic and automatic processes in prospective memory retrieval: a multiprocess framework," *Applied Cognitive Psychology*, vol. 14, no. 7, S127–S144, 2000.

[107] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, Sep. 1995.

[108] M.-L. Zhang and Z.-H. Zhou, "Ml-knn: A lazy learning approach to multi-label learning," *Pattern recognition*, vol. 40, no. 7, pp. 2038–2048, 2007.

[109] J. Lafferty, A. McCallum, and F. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proceedings of the eighteenth international conference on machine learning, ICML*, vol. 1, 2001, pp. 282–289.

[110] Sy Bor Wang, A. Quattoni, L.-P. Morency, D. Demirdjian, and T. Darrell, "Hidden Conditional Random Fields for Gesture Recognition," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2 (CVPR'06)*, vol. 2, IEEE, 2006, pp. 1521–1527.

[111] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[112] P. Casale, O. Pujol, and P. Radeva, "Human activity recognition from accelerometer data using a wearable device," in *Iberian Conference on Pattern Recognition and Image Analysis*, Springer, 2011, pp. 289–296.

[113] Z. He and L. Jin, "Activity recognition from acceleration data based on discrete consine transform and svm," in *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, IEEE, 2009, pp. 5041–5044.

[114] M.-C. Chuang, R. Bala, E. A. Bernal, P. Paul, A. Burry, *et al.*, "Estimating gaze direction of vehicle drivers using a smartphone camera.," in *CVPR Workshops*, 2014, pp. 165–170.

[115] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[116] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[117] L. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[118] D. L. Vail, M. M. Veloso, and J. D. Lafferty, "Conditional random fields for activity recognition," in *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems - AAMAS '07*, New York, New York, USA: ACM Press, 2007, p. 1.

[119] E. T. Hall, *The Hidden Dimension*. Anchor Books, 1969.

[120] C. Dondrup, N. Bellotto, F. Jovan, and M. Hanheide, "Real-time multisensor people tracking for human-robot spatial interaction," in *Workshop on Machine Learning for Social Robotics at International Conference on Robotics and Automation (ICRA)*, ICRA/IEEE, 2015.

[121] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer-Verlag New York, 2006.

[122] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," *arXiv preprint arXiv:1612.08242*, 2016.

[123] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1499–1503, 2016.

[124] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*, Springer, 2014, pp. 740–755.

[125] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[126] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields," *arXiv preprint arXiv:1611.08050*, 2016.

[127] S. G. Hart and L. E. Staveland, "Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research," in, 1988, pp. 139–183.

[128] Y. Benjamini and Y. Hochberg, "Controlling the false discovery rate: A practical and powerful approach to multiple testing," *Journal of the royal statistical society. Series B (Methodological)*, pp. 289–300, 1995.

[129] S. Honig and T. Oron-Gilad, "Understanding and resolving failures in human-robot interaction: Literature review and model development," *Frontiers in psychology*, vol. 9, p. 861, 2018.

[130] N. Du, K. Y. Huang, and X. J. Yang, "Not all information is equal: Effects of disclosing different types of likelihood information on trust, compliance and reliance, and task performance in human-automation teaming," *Human Factors*, 2019.

[131] X. J. Yang, V. V. Unhelkar, K. Li, and J. A. Shah, "Evaluating effects of user experience and system transparency on trust in automation," in *HRI*, IEEE, 2017, pp. 408–416.

[132] S. R. Dixon, C. D. Wickens, and J. S. McCarley, "On the independence of compliance and reliance: Are automation false alarms worse than misses?" *Human factors*, vol. 49, no. 4, pp. 564–572, 2007.

[133] M. Arvan, B. Fahimnia, M. Reisi, and E. Siemsen, "Integrating human judgement into quantitative forecasting methods: A review," *Omega*, vol. 86, pp. 237–252, 2019.

[134] E. Rogers and R. R. Murphy, "Tele-assistance for semi-autonomous robots," 1994.

[135] S. A. Patel and A. K. Kamrani, "Intelligent decision support system for diagnosis and maintenance of automated systems," *Computers & industrial engineering*, vol. 30, no. 2, pp. 297–319, 1996.

[136] H. Warnquist, J. Kvarnström, and P. Doherty, "Exploiting fully observable and deterministic structures in goal POMDPs," in *Twenty-Third International Conference on Automated Planning and Scheduling*, 2013.

[137] D. Norman, *The design of everyday things: Revised and expanded edition*. Basic books, 2013.

[138] A. Degani, I. Barshi, and M. G. Shafto, "Information organization in the airline cockpit: Lessons from flight 236," *Journal of cognitive engineering and decision making*, vol. 7, no. 4, pp. 330–352, 2013.

[139] M. Lenz, E. Auriol, and M. Manago, "Diagnosis and decision support," in *Case-Based Reasoning Technology: From Foundations to Applications*, M. Lenz, H.-D. Burkhard, B. Bartsch-Spörl, and S. Wess, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 51–90.

[140] O. Pettersson, L. Karlsson, and A. Saffiotti, "Model-Free Execution Monitoring in Behavior-Based Robotics," *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, vol. 37, no. 4, pp. 890–901, Aug. 2007.

[141] D. Park, H. Kim, Y. Hoshi, Z. Erickson, A. Kapusta, and C. C. Kemp, "A multimodal execution monitor with anomaly classification for robot-assisted feeding," in *IROS*, IEEE, Sep. 2017, pp. 5406–5413.

[142] H. Wu, S. Luo, L. Chen, S. Duan, S. Chumkamon, D. Liu, Y. Guan, and J. Rojas, "Endowing Robots with Longer-term Autonomy by Recovering from External Disturbances in Manipulation through Grounded Anomaly Classification and Recovery Policies," *arXiv: 1809.03979*, Sep. 2018. arXiv: 1809.03979.

[143] J. S. Breese and D. Heckerman, "Decision-Theoretic Troubleshooting: A Framework for Repair and Experiment," in *UAI*, Morgan Kaufmann Publishers Inc., 1996, pp. 124–132.

[144] R. R. Murphy and D. Hershberger, "Handling sensing failures in autonomous mobile robots," *IJRR*, vol. 18, no. 4, pp. 382–400, 1999.

[145] L. Parker and B. Kannan, "Adaptive Causal Models for Fault Diagnosis and Recovery in Multi-Robot Teams," in *IROS*, IEEE, Oct. 2006, pp. 2703–2710.

[146] C. Sammut, "Behavioral Cloning," in *Encyclopedia of Machine Learning*, Boston, MA: Springer US, 2011, pp. 93–97.

[147] A. Mohseni-Kabir, C. Rich, S. Chernova, C. L. Sidner, and D. Miller, "Interactive hierarchical task learning from a single demonstration," in *HRI*, 2015, pp. 205–212.

[148] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard, "Recent advances in robot learning from demonstration," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, pp. 297–330, 2020.

[149] M. Gombolay, A. Bair, C. Huang, and J. Shah, "Computational design of mixed-initiative human–robot teaming that considers human factors: Situational aware-

ness, workload, and workflow preferences," *IJRR*, vol. 36, no. 5-7, pp. 597–617, 2017.

[150] M. Gombolay, X. J. Yang, B. Hayes, N. Seo, Z. Liu, S. Wadhwania, T. Yu, N. Shah, T. Golen, and J. Shah, "Robotic assistance in the coordination of patient care," *The International Journal of Robotics Research*, vol. 37, no. 10, pp. 1300–1316, 2018.

[151] 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," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, Nov. 2013, pp. 3535–3542.

[152] P. Struss, "Model-based decision support systems-conceptualization and general architecture," in *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, Springer, 2020, pp. 588–600.

[153] R. Reiter, "A theory of diagnosis from first principles," *Artificial Intelligence*, vol. 32, no. 1, pp. 57–95, Apr. 1987.

[154] B. Andersen and T. Fagerhaug, *Root cause analysis: simplified tools and techniques*. Quality Press, 2006.

[155] S. A. Alharthi, O. Alsaedi, Z. O. Toups, J. Tanenbaum, and J. Hammer, "Playing to wait: A taxonomy of idle games," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, ser. CHI '18, New York, NY, USA: Association for Computing Machinery, 2018, pp. 1–15.

[156] M. E. Fonteyn, B. Kuipers, and S. J. Grobe, "A description of think aloud method and protocol analysis," *Qualitative Health Research*, vol. 3, no. 4, pp. 430–441, 1993.

[157] P. G. Polson, C. Lewis, J. Rieman, and C. Wharton, "Cognitive walkthroughs: A method for theory-based evaluation of user interfaces," *International Journal of Man-Machine Studies*, vol. 36, no. 5, pp. 741–773, May 1992.

[158] S. Banerjee and S. Chernova, "Fault diagnosis in robot task execution," in *AAAI Spring Symposium Series*, 2019.

[159] A. Abdollahi, K. R. Pattipati, A. Kodali, S. Singh, S. Zhang, and P. B. Luh, "Probabilistic Graphical Models for Fault Diagnosis in Complex Systems," in, 2016, pp. 109–139.

[160] M. C. Hughes, W. T. Stephenson, and E. Sudderth, "Scalable adaptation of state complexity for nonparametric hidden markov models," in *Advances in Neural In-*

formation Processing Systems, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28, Curran Associates, Inc., 2015, pp. 1198–1206.

[161] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[162] S. Coppers, K. Luyten, D. Vanacken, D. Navarre, P. Palanque, and C. Gris, "Fortunettes: Feedforward about the future state of gui widgets," *Proc. HCI*, vol. 3, no. EICS, pp. 1–20, 2019.

[163] B. Lafreniere, P. K. Chilana, A. Fourney, and M. A. Terry, "These aren't the commands you're looking for: Addressing false feedforward in feature-rich software," in *UIST*, 2015, pp. 619–628.

[164] M. Natarajan and M. Gombolay, "Effects of anthropomorphism and accountability on trust in human robot interaction," in *HRI*, 2020, pp. 33–42.

[165] A. B. Beck, A. D. Schwartz, A. R. Fugl, M. Naumann, and B. Kahl, "Skill-based Exception Handling and Error Recovery for Collaborative Industrial Robots.," in *FinE-R IROS*, 2015, pp. 5–10.

[166] B. Nushi, E. Kamar, E. Horvitz, and D. Kossmann, "On human intellect and machine failures: Troubleshooting integrative machine learning systems," in *AAAI*, 2017.

[167] M. K. Lee, S. Kiesler, J. Forlizzi, S. Srinivasa, and P. Rybski, "Gracefully mitigating breakdowns in robotic services," in *HRI*, IEEE, Mar. 2010, pp. 203–210.

[168] D. J. Brooks, M. Begum, and H. A. Yanco, "Analysis of reactions towards failures and recovery strategies for autonomous robots," in *RO-MAN*, IEEE, 2016, pp. 487–492.

[169] M. Vasic and A. Billard, "Safety issues in human-robot interactions," in *ICRA*, IEEE, 2013, pp. 197–204.

[170] J. K. Kruschke, "Rejecting or accepting parameter values in bayesian estimation," *Advances in Methods and Practices in Psychological Science*, vol. 1, no. 2, pp. 270–280, 2018.

[171] J. Brooke, "Sus-a quick and dirty usability scale," *Usability evaluation in industry*, vol. 189, no. 194, pp. 4–7, 1996.

[172] P.-C. Bürkner, "Advanced Bayesian multilevel modeling with the R package brms," *The R Journal*, vol. 10, no. 1, pp. 395–411, 2018.

[173] B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell, "Stan: A probabilistic programming language," *Journal of statistical software*, vol. 76, no. 1, 2017.

[174] A. Vehtari, A. Gelman, and J. Gabry, "Practical bayesian model evaluation using leave-one-out cross-validation and waic," *Statistics and computing*, vol. 27, no. 5, pp. 1413–1432, 2017.

[175] J. Cohen, "Statistical power analysis for the social sciences," 1988.

[176] D. Makowski, M. S. Ben-Shachar, S. Chen, and D. Lüdecke, "Indices of effect existence and significance in the bayesian framework," *Frontiers in Psychology*, vol. 10, p. 2767, 2019.

[177] R. Wiczorek and D. Manzey, "Supporting attention allocation in multitask environments: Effects of likelihood alarm systems on trust, behavior, and performance," *Human factors*, vol. 56, no. 7, pp. 1209–1221, 2014.

[178] D. Arnott and G. Pervan, "Eight key issues for the decision support systems discipline," *Decision Support Systems*, vol. 44, no. 3, pp. 657–672, 2008.

[179] D. Kirchner, "Self-healing in autonomous robot teams," Ph.D. dissertation, Universitätsbibliothek Kassel, 2017.

[180] V. Lín, "Scheduling results applicable to decision-theoretic troubleshooting," *International Journal of Approximate Reasoning*, vol. 56, pp. 87–107, 2015.

[181] V. Raghavan, M. Shakeri, and K. Pattipati, "Optimal and near-optimal test sequencing algorithms with realistic test models," *IEEE Transactions on systems, man, and cybernetics*, vol. 29, no. 1, pp. 11–26, 1999.

[182] P. Perrault, V. Perchet, and M. Valko, "Finding the bandit in a graph: Sequential search-and-stop," in *Proceedings of Machine Learning Research*, K. Chaudhuri and M. Sugiyama, Eds., ser. Proceedings of Machine Learning Research, vol. 89, PMLR, Apr. 2019, pp. 1668–1677.

[183] S. Nikolaidis, D. Hsu, and S. Srinivasa, "Human-robot mutual adaptation in collaborative tasks: Models and experiments," *The International Journal of Robotics Research*, vol. 36, no. 5-7, pp. 618–634, 2017.

[184] M. Chen, S. Nikolaidis, H. Soh, D. Hsu, and S. Srinivasa, "Trust-aware decision making for human-robot collaboration: Model learning and planning," *ACM Transactions on Human-Robot Interaction (THRI)*, vol. 9, no. 2, pp. 1–23, 2020.

[185] D. Karimi, H. Dou, S. K. Warfield, and A. Gholipour, "Deep learning with noisy labels: Exploring techniques and remedies in medical image analysis," *Medical Image Analysis*, vol. 65, p. 101 759, 2020.

[186] J. Uesato, A. Kumar, C. Szepesvari, T. Erez, A. Ruderman, K. Anderson, K. (Dvijotham, N. Heess, and P. Kohli, "Rigorous Agent Evaluation: An Adversarial Approach to Uncover Catastrophic Failures," in *International Conference on Learning Representations*, 2019.

[187] A. Malinin and M. Gales, "Reverse KL-Divergence training of prior networks: Improved uncertainty and adversarial robustness," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 14 547–14 558.

[188] D. Hendrycks, M. Mazeika, S. Kadavath, and D. Song, "Using self-supervised learning can improve model robustness and uncertainty," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 15 663–15 674.

[189] J. M. Wooldridge, "On the application of robust, regression-based diagnostics to models of conditional means and conditional variances," *Journal of econometrics*, vol. 47, no. 1, pp. 5–46, 1991.

[190] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 23–30.

[191] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, and S. Birchfield, "Training deep networks with synthetic data: Bridging the reality gap by domain randomization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 969–977.

[192] X. Zhu and X. Wu, "Class noise vs. attribute noise: A quantitative study," *Artificial intelligence review*, vol. 22, no. 3, pp. 177–210, 2004.

[193] B. Frénay and M. Verleysen, "Classification in the presence of label noise: A survey," *IEEE transactions on neural networks and learning systems*, vol. 25, no. 5, pp. 845–869, 2013.

[194] D. Arpit, S. Jastrzkebski, N. Ballas, D. Krueger, E. Bengio, M. S. Kanwal, T. Maharaj, A. Fischer, A. Courville, Y. Bengio, and S. Lacoste-Julien, "A closer look at memorization in deep networks," D. Precup and Y. W. Teh, Eds., ser. Proceedings

of Machine Learning Research, vol. 70, International Convention Centre, Sydney, Australia: PMLR, Aug. 2017, pp. 233–242.

[195] A. Drory, S. Avidan, and R. Giryes, "On the resistance of neural nets to label noise," *arXiv preprint arXiv:1803.11410*, vol. 2, 2018. arXiv: 1803.11410.

[196] M. Li, M. Soltanolkotabi, and S. Oymak, "Gradient descent with early stopping is provably robust to label noise for overparameterized neural networks," S. Chiappa and R. Calandra, Eds., ser. Proceedings of Machine Learning Research, vol. 108, Online: PMLR, Aug. 2020, pp. 4313–4324.

[197] D. Hendrycks, M. Mazeika, D. Wilson, and K. Gimpel, "Using trusted data to train deep networks on labels corrupted by severe noise," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., Curran Associates, Inc., 2018, pp. 10 456–10 465.

[198] S. Sukhbaatar, J. Bruna, M. Paluri, L. Bourdev, and R. Fergus, "Training convolutional networks with noisy labels," *Workshop at ICLR 2015*, 2015. arXiv: 1406.2080 [cs.CV].

[199] M. Charikar, J. Steinhardt, and G. Valiant, "Learning from untrusted data," in *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, 2017, pp. 47–60.

[200] N. Konstantinov and C. Lampert, "Robust Learning from Untrusted Sources," in *Proceedings of the 36th International Conference on Machine Learning*, May 2019. arXiv: 1901.10310.

[201] M. Ren, W. Zeng, B. Yang, and R. Urtasun, "Learning to reweight examples for robust deep learning," J. Dy and A. Krause, Eds., ser. Proceedings of Machine Learning Research, vol. 80, Stockholmsmässan, Stockholm Sweden: PMLR, Jul. 2018, pp. 4334–4343.

[202] D. Das, S. Banerjee, and S. Chernova, "Explainable ai for robot failures: Generating explanations that improve user assistance in fault recovery," in *Int. Conf. on Human-Robot Interaction*, 2021.

[203] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," *arXiv preprint arXiv:1708.05866*, 2017.

[204] R. Paleja, A. Silva, L. Chen, and M. Gombolay, "Interpretable apprenticeship learning from heterogeneous decision-making via personalized embeddings," *arXiv preprint arXiv:1906.06397*, 2019.

[205] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[206] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[207] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

[208] G. Camporese, P. Coscia, A. Furnari, G. M. Farinella, and L. Ballan, "Knowledge distillation for action anticipation via label smoothing," *arXiv preprint arXiv:2004.07711*, 2020.

[209] R. Azzam, Y. Alkendi, T. Taha, S. Huang, and Y. Zweiri, "A stacked lstm based approach for reducing semantic pose estimation error," *IEEE Transactions on Instrumentation & Measurement*, 2020.

[210] W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl, "Fault tree handbook," Nuclear Regulatory Commission Washington dc, Tech. Rep., 1981.

[211] J. Peeters, R. Basten, and T. Tinga, "Improving failure analysis efficiency by combining FTA and FMEA in a recursive manner," *Reliability Engineering & System Safety*, vol. 172, pp. 36–44, Apr. 2018.

[212] P. Schillinger, S. Kohlbrecher, and O. von Stryk, "Human-Robot Collaborative High-Level Control with an Application to Rescue Robotics," in *IEEE International Conference on Robotics and Automation*, Stockholm, Sweden, May 2016.

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

[214] R. A. Gutierrez, V. Chu, A. L. Thomaz, and S. Niekum, "Incremental task modification via corrective demonstrations," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 1126–1133.

[215] K. Chen, D. Kent, N. Shrivatsav, H. Ravichandar, and S. Chernova, "Learning probabilistic hierarchical task networks from unannotated demonstrations," in *Conference on Robot Learning*, 2020.

[216] Z. Wang and M. Gombolay, "Learning scheduling policies for multi-robot coordination with graph attention networks," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4509–4516, 2020.

[217] H. Ravichandar, K. Shaw, and S. Chernova, "Strata: Unified framework for task assignments in large teams of heterogeneous agents," *Autonomous Agents and Multi-Agent Systems*, vol. 34, pp. 1–25, 2020.

[218] S. Andrist, D. Bohus, E. Kamar, and E. Horvitz, "What went wrong and why? diagnosing situated interaction failures in the wild," in *International Conference on Social Robotics*, Springer, 2017, pp. 293–303.

[219] R. M. Aronson and H. Admoni, "Gaze for error detection during human-robot shared manipulation," in *Fundamentals of Joint Action workshop, Robotics: Science and Systems*, 2018.

[220] A. Washburn, A. Adeleye, T. An, and L. D. Riek, "Robot errors in proximate hri: How functionality framing affects perceived reliability and trust," *ACM Transactions on Human-Robot Interaction (THRI)*, vol. 9, no. 3, pp. 1–21, 2020.

[221] J. Dzieza, *Robots aren't taking our jobs — they're becoming our bosses.*

[222] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan, "A survey on bias and fairness in machine learning," *arXiv preprint arXiv:1908.09635*, 2019.

[223] Y. Li and N. Vasconcelos, "Repair: Removing representation bias by dataset resampling," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9572–9581.