Failing Gracefully: Mitigating Impact of Inevitable Robot Failures

Duc M. Nguyen, Saad A. Ghani, Andrew Marshall, Allison Andreyev, Gregory J. Stein, and Xuesu Xiao

***Abstract*— Service robots operate in household environments shared with humans, pets, and everyday objects, where they are highly susceptible to failures such as software crashes, hardware degradation, or unpredictable interactions. While roboticists strive to minimize failures, some remain inevitable, making it critical to mitigate their potential consequences for safe and reliable deployment. This paper introduces a novel safety formulation that evaluates both the probability of impactful interactions between robots and surrounding entities during failures, and the severity of their outcomes. By quantifying the impact of failures on different entities, our approach enables robots to make informed planning decisions that balance safety with task efficiency. To support systematic evaluation, we also present `FailBench`, a MuJoCo-based simulation framework for studying robot-environment interactions under diverse failure modes, including sensing issues and actuator malfunctions. Together, our safety formulation and `FailBench` provide a foundation for developing safer and more robust motion plans and learned policies in real-world household environments.**

I. Introduction

Household robots are expected to perform diverse tasks such as cleaning, fetch-and-deliver, and cooking. These tasks require operation in dynamic environments shared with humans, pets, and cluttered objects, where robots are highly susceptible to failures such as falling, dropping objects, or spilling liquids. Such failures may arise from hardware degradation, software bugs, or unpredictable conditions. For example, a spilled drink can short-circuit electronics and create a slippery floor that risks human injury; a dropped object may break fragile items; and a fall can injure a nearby person or pet. Ensuring safety in the presence of failures is therefore crucial for the widespread adoption of service robots in our homes.

Most existing work emphasizes failure prevention and recovery strategies [1]–[5], but often lacks systematic failure categorization and overlooks the *consequences* of inevitable failures. In household settings, failures extend beyond incomplete tasks (e.g., getting stuck at a doorway) to hazardous events that can damage property or endanger people. While previous failure prevention and recovery strategies can lower the chance of failures, they do not consider the consequences if an inevitable failure occurs in the planning process. One naive way to mitigate failure impact is through inflating objects or defining "no-go zones". But by trimming down the workspace to stay away from everything unnecessarily limits where the robot can operate and compromise task efficiency.

Duc M. Nguyen, Saad A. Ghani, Gregory J. Stein, and Xuesu Xiao are with the Department of Computer Science, George Mason University. Emails: {mnguy21, sghani2, gjstein, xiao}@gmu.edu.

Andrew Marshall and Allison Andreyev are high school interns affiliated with the RobotiXX Lab, George Mason University. Emails: {amarshall10003, allisonmandreyev}@gmail.com.

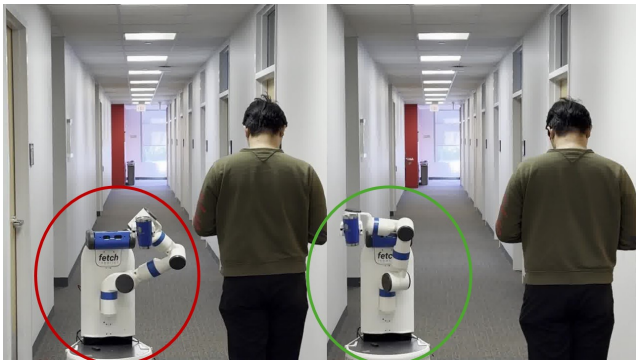

Fig. 1: Robot failure mitigation example. **Left** (red): Carrying a bottle of hot water close to a human poses a higher risk of spilling and causing injury if dropped. **Right** (green): A safer strategy positions the bottle farther away from the human to reduce potential harm in the event of failure.

Another major bottleneck in studying such failure consequences is the scarcity of real-world data during real-world service robot deployment and the lack of a standard benchmark to evaluate how robots can mitigate failure impact. Without sufficient data and a benchmark, it is difficult for robots to analyze diverse failure consequences to plan motions or to learn policies that distinguish minor and severe failures to simultaneously maintain efficiency and safety.

To enable robots to reason about and mitigate failure consequences, we propose a novel problem formulation that explicitly accounts for the *consequences* of failures during planning. Our framework evaluates potential *interactions* between robot components and environmental entities when failures happen, quantifies the *severity* of each interaction, and aggregates them into an overall impact measure. Achieving this requires reasoning not only about *when* failures occur, but also *what* they impact and *how severe* the resulting consequences are. For example, in Figure 1, a robot carrying a bottle in a hallway may position the bottle at a lateral offset from humans, not only to prevent collisions, but to limit the impact on the person if the bottle is accidentally dropped.

To provide failure data and benchmark, we present `FailBench`, a MuJoCo-based simulation framework designed to study how robots interact with their environments under sudden unexpected failures such as hardware shutdowns, sensor degradation, or actuator malfunctions. `FailBench` enables systematic analysis of failure-induced behaviors, supporting the development of safer motion plans and more robust learned policies. The framework provides: (1) a curated library of scenes and objects household robots

may commonly encounter, (2) a suite of Python implementations of common motion planners for both navigation and manipulation, and (3) a novel failure injector that introduces controlled perturbations to the robots and its environments to collect failure data and benchmark failure mitigation strategies. By offering a controlled yet diverse environment for benchmarking, `FailBench` facilitates rigorous evaluation of robot safety and resilience in the presence of failures, ultimately guiding safer real-world deployment.

In summary, this paper makes the following contributions:

- A novel formulation for failure impact assessment with a new safety metric, combining interaction modeling and severity metrics to quantify safety risks;
- `FailBench`, a MuJoCo-based benchmark to study failure mitigation by generating diverse failure data and systematically evaluating failure impacts; and
- Demonstration that our new formulation can reflect failure consequences simulated in `FailBench`, paving the way toward future research into failure mitigation for service robots.

II. Related Work

Robot Failure is a critical consideration for the widespread adoption of robotic systems, with prior work aiming to reduce failure likelihood and develop robust risk formulations [1], [6]–[8]. Failures have been categorized as *physical* (e.g., power loss, sensor errors, communication faults, end-effector issues, and control faults) or *man-made* (e.g., design flaws and user mistakes) [6], [7], and further characterized by *repairability* (field vs. non-field) and *impact* (terminal vs. non-terminal) [7]. Extensions include Guo et al.'s classification into system, operational, design, and safeguarding errors [9]. Most existing approaches treat failure primarily as a problem of task incompletion, arising from suboptimal planning or policy decisions. For example, the RoboFail dataset [10] focuses on reinforcement learning policies and defines failure as the inability to successfully complete a manipulation task. While this framework is valuable for evaluating policy robustness, it largely overlooks the critical question of what happens when failures occur during execution due to hardware faults, unexpected dynamics, or other low-level disturbances that are impossible to predict or prevent in advance.

Safety Filtering for Robotics represents a control-theoretic approach designed to ensure system safety independently of task-driven base policies. These methods operate by monitoring policies and intervening with safe control actions when systems approach unsafe states, including control barrier functions [11]–[13], HJ reachability [14], [15], and model-predictive shielding. However, traditional safety filtering approaches primarily focus on *preventing* failures through collision avoidance and constraint satisfaction. While prevention is crucial, these methods become insufficient when dealing with sudden, unpredictable hardware failures—such as joint degradation or actuator malfunctions—that cannot be easily detected or avoided in real-time. In household environments, the consequences of such failures extend far beyond direct collisions, potentially causing spilled liquids, dropped fragile objects, or cascading hazards that endanger both surroundings and humans.

Fault Injection for Risk Analysis has emerged as a complementary approach, focusing on understanding system behavior under various failure conditions. Yuliang et al. [16] presented a ROS-based failure injection framework using six different failure parameters and randomization capabilities. Their approach targets sensor signal faults (bias and noise) injected into position signals, demonstrating that fault phase (planning vs. execution) critically determines failure outcomes. Christensen et al. [17] investigated hardware fault detection using back-propagation neural networks trained on normal and faulty operational data, demonstrating robust fault identification with low false positives. While these approaches provide valuable insights into fault detection and system vulnerabilities, they primarily focus on *identifying* when failures occur rather than *quantifying the consequences* of failures that cannot be prevented or detected in time.

This work shifts the focus from preventing failures to *assessing their impact*. When robot joints degrade suddenly or hardware fails unexpectedly—scenarios where detection or avoidance is impossible—the critical question is: what are the potential consequences at a given moment? We introduce a framework to estimate failure impacts across robot states and actions, enabling robots to make decisions that account for the *severity* of failures and to choose actions that mitigate consequences when failures are unavoidable.

III. Problem Formulation

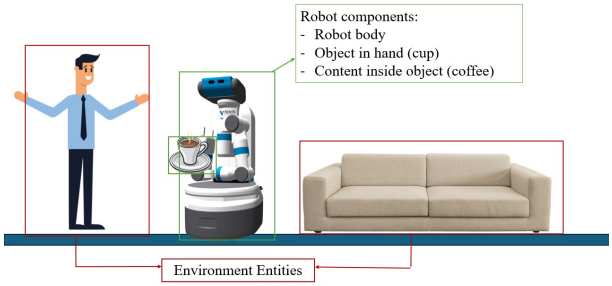

Fig. 2: Illustration of the components within the robot's workspace.

A service robot operating in human environments must minimize risks to people and objects, particularly during inevitable failures. For instance, a robot carrying a bowl of hot soup must navigate around furniture and people, as even a minor misstep could spill hot liquid and cause injury or damage. Failure-aware planning should not only enable smooth task execution but also explicitly account for the potential consequences of failures, integrating safety considerations directly into decision-making.

A. Robot Components and Environment Entities

We focus on robots operating in a household environment, representing tasks that may require both navigation and

manipulation. The robot, denoted as R, is divided into three primary components that can pose hazards: the robot body (r_1), the object being carried (r_2), and, if the object is a container, its contents (r_3). Each component $r_j \in R$ may present distinct risks when interacting with environmental entities. An entity $e_i \in E$, static or dynamic, represents any object or agent in the environment that could be affected by the robot. Figure 2 illustrates the division of the robot's workspace into components and surrounding entities.

B. Interaction between Robot Components and Entities

We define an ***interaction*** as an accident following a failure that results in damage or harm to an environmental entity. At time step t, the probability of an interaction is denoted as

$$P_t(x_t, e_i, r_j \mid \text{Failure} = \text{True}),$$

representing the likelihood that a robot component r_j interacts with an environmental entity e_i at robot state x_t when a failure occurs.

Since failures are assumed inevitable and undetectable, our framework does not attempt to minimize—or explicitly reason about—the probability of failure itself. Instead, it focuses on minimizing the impact of failures on the environment. Importantly, P_t captures the likelihood that a failure will negatively affect a specific entity, rather than the overall probability of failure.

C. Severity of Interaction

Not all interactions carry equal consequences. For example, spilling hot coffee is more hazardous than spilling iced water, even if the probabilities of interaction are similar. To capture this, we define a ***severity*** factor $S(e_i, r_j)$, quantifying the potential impact of an interaction between an environmental entity e_i and a robot component r_j during a failure. This factor enables the robot to adjust its behavior according to the potential severity of interactions.

D. Optimization Objective

We define the impact of a failure as the product of the probability of a harmful interaction and its severity. Using this, the motion planner minimizes the following objective:

$$\begin{aligned} \underset{x_1,\ldots,x_T}{\operatorname{argmin}} \sum_{t=1}^{T} \Bigg[& V(x_t, x_{t-1}) \\ & + w \cdot \sum_{r_j \in R} \sum_{e_i \in E} P_t(x_t, e_i, r_j | F) \cdot S(e_i, r_j) \Bigg], \end{aligned} \tag{1}$$

where $V(x_t, x_{t-1})$ is the cost of transitioning from state x_{t-1} to x_t, and w is a weighting parameter that balances motion efficiency against failure impact considerations.

This formulation allows the robot to evaluate the risk associated with its states and actions, balancing operational efficiency with safety. By considering both the likelihood of causing harm and the severity of potential consequences, the robot can make informed, risk-averse decisions that minimize the impact of inevitable failures.

IV. FailBench

We introduce `FailBench`, a MuJoCo [18] simulation framework for studying how robots interact with their environments under sudden, unexpected failures. `FailBench` enables systematic analysis and data generation of failure-induced behaviors, providing insights for developing proactively safe motion plans or robust learned policies. It also provides a standardized benchmark for evaluating, comparing, and developing robot planning and control strategies under various failure conditions, ultimately enhancing safety and resilience in real-world deployments.

A. Simulation

`FailBench` leverages MuJoCo as its core simulator, offering high-fidelity physics essential for accurately modeling failure behaviors. To facilitate perception-driven analyses, it provides interfaces to access ground-truth states, contact forces, and other environment metrics. Our framework incorporates scenes and object meshes curated from prior work [19], [20], and supports a variety of robot embodiments such as Clearpath Jackal, Fetch mobile manipulator, and Franka Panda Arm. Unlike existing platforms that primarily target at providing training environments for learning-based approaches, `FailBench` emphasizes failure-aware planning and policy evaluation.

B. Motion Planning

1) Navigation: Our framework provides an interface to generate floor maps for high-level path planning, supporting both search-based algorithms such as A* and Dijkstra's, as well as sampling-based planners. For low-level control of wheel-based robots, we implement the Dynamic Window Approach [21] and differential-drive controllers.

2) Manipulation: We leverage Mink [22], a Python library for differential inverse kinematics built on MuJoCo. Our framework supports arm planning in both joint-space and task-space, integrating a variety of motion planners:

- Sampling-based planners: PRM (Probabilistic Roadmaps) [23], RRT (Rapidly-exploring Random Trees) [24], RRT* [25], RRTConnect [26], and TRRT (Transition-based RRT) [26], and
- Optimization-based planners: STOMP (Stochastic Trajectory Optimization for Motion Planning) [27] and CHOMP (Covariant Hamiltonian Optimization for Motion Planning) [28].

This setup enables researchers to leverage, evaluate, and compare a wide range of navigation and manipulation strategies within the same simulation platform. We plan to continue expanding the library of planners in future releases.

C. Failure Injector

A key feature of `FailBench` is its failure injector, which introduces controlled perturbations to the robot and its environment. Injecting failures enables systematic evaluation of robot safety across a variety of failure modes, including sensor noise, hardware malfunctions, and performance degradation. By simulating such scenarios, `FailBench` helps

TABLE I: Failure Injector for `FailBench`

	Failure Type	Description	Onset
Actuator Failures	Complete Shutdown	Total loss of joint control authority	Instant
	Loose Joint	Reduced damping and control	Gradual
	Limited ROM	Restricted range of motion	Instant
	Partial Degradation	Reduced control authority	Gradual
	Gradual Degradation	Slow deterioration of performance	Gradual
	Stuck Position	Joint locked at current position	Instant
	Increased Friction	Higher resistance in joint motion	Gradual
Sensor Failures	Noisy Sensor	Added measurement noise	Gradual
	Bias Drift	Accumulating systematic errors	Gradual
	Control Delay	Network/computational latency	Gradual
	Sensor Dropout	Intermittent loss of measurements	Instant
	Scale Factor Error	Incorrect measurement scaling	Gradual
	Dead Zone	No response in measurement range	Instant
End-Effector Failures	Gripper Failure	End-effector malfunction	Instant
	Weak Grip	Reduced grasping force	Gradual
	Stuck Open/Closed	Gripper locked in position	Instant
Power System Failures	Battery Degradation	Reduced available power	Gradual
	Voltage Drop	Insufficient power delivery	Gradual
	Power Loss	Complete system shutdown	Instant

researchers assess and improve the robustness of both motion plans and learned policies. Currently, we support actuator failures such as *complete shutdown* and *loose joint*, sensor failures including *noisy sensor* and *bias drift*, and end-effector failures such as *gripper failure*. More failures will be expanded in future work based on the community's feedback.

1) Overview: Our failure injection framework provides a comprehensive taxonomy of three primary categories: actuator, sensor, and end-effector failures (Table I). Each failure type is characterized by its onset pattern (instant or gradual), failure degrees, and configurable parameters that control the failure's magnitude and duration. Instant failures occur immediately upon injection, while gradual failures develop or persist over time during execution.

The injector operates by modifying the underlying MuJoCo simulation parameters in real-time, enabling realistic physical simulation of robot malfunctions. Unlike approaches that only add noise to control signals, our framework directly alters actuator gains, joint constraints, sensor feedback, and mechanical properties to accurately represent the physics of actual hardware failures. This approach ensures that failure propagation and system responses mirror real-world behaviors for maximum failure fidelity.

Failures can be injected at any point during trajectory execution, with precise timing control and the ability to affect single joints, multiple joints simultaneously, or the entire system. The framework supports both deterministic failure scenarios for controlled testing and stochastic failure injection for robustness evaluation under uncertain conditions.

2) Failure Types: Table I summarizes the various failure modes currently included for injection in our simulation.

Actuator Failures represent loss or degradation of motor control authority across multiple failure modes. *Complete shutdown* removes all control from specified joints, simulating motor driver failures or emergency stops. *Loose joint* failures model reduced damping and control precision from mechanical backlash or worn components. *Limited ROM* restricts joint movement ranges, simulating mechanical constraints. Progressive failures include *partial degradation* and *gradual degradation*, which reduce control authority over time, while *stuck position* locks joints at their current configuration. *Increased friction* models wear-related resistance in joint mechanisms.

Sensor Failures affect the robot's perception of its own state and environment. *Noisy sensor* failures add measurement noise to joint position feedback. *Bias drift* introduces accumulating systematic errors that model sensor calibration drift over time. *Control delay* simulates network latency or computational delays in the control loop. Additional failure modes include *sensor dropout* for intermittent measurement loss, *scale factor error* for incorrect measurement scaling due to calibration issues, and *dead zone* failures where sensors become unresponsive within specific ranges.

End-Effector Failures specifically target manipulation capabilities. *Gripper failure* encompasses complete end-effector malfunction, while *weak grip* models progressive reduction in grasping force. *Stuck open/closed* scenarios simulate mechanical failures that lock the gripper in fixed positions, directly impacting object manipulation success.

Power System Failures model energy-related degradation common in mobile robots. *Battery degradation* reduces available power over time, *voltage drop* causes insufficient power delivery affecting actuator performance, and *power loss* simulates complete system shutdown scenarios.

Each failure type includes configurable failure degrees (minor, moderate, severe, and critical) that scale the failure's impact proportionally. Duration parameters allow for temporary failures that automatically restore after a specified time, or permanent failures that persist until explicitly restored. This flexibility enables researchers to study both transient disturbances and persistent fault conditions.

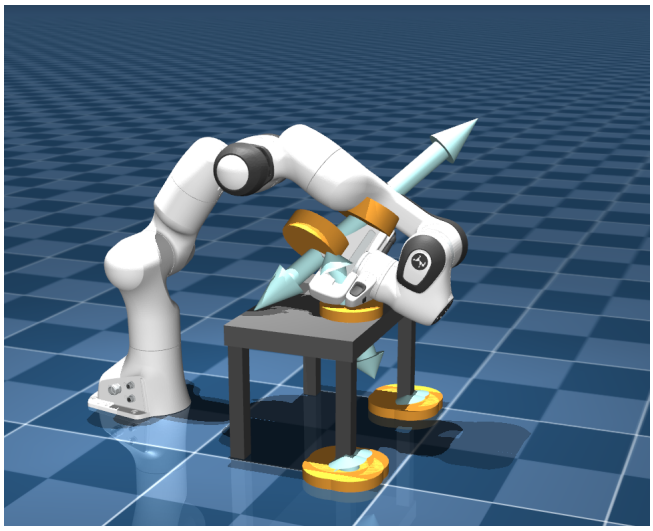

Fig. 3: Visualization of contact detection after shoulder joint complete shutdown failure. The robotic arm has collided with the table, generating multiple contact points (shown in yellow) with associated force vectors, demonstrating how `FailBench` quantifies physical interaction during failure scenarios.

D. Data generation

In `FailBench`, failure aftereffects are quantified through contact events, which serve as measurable proxies for the physical impact of robot failures. This approach directly aligns with the interaction framework defined in our problem formulation (Section III). Leveraging the MuJoCo physics engine, `FailBench` captures both contact point positions and their corresponding contact forces, providing comprehensive data for failure impact assessment. Figure 3 illustrates how contact events are detected and measured following a robot failure.

`FailBench` provides a comprehensive open-source framework with pre-configured robots, planners, environments, and object assets to streamline experimentation and accelerate research in robot safety.

V. Failure Impact Analysis

To evaluate the failure impact formulation proposed in Section III, we develop computational techniques that quantify the potential impacts arising from robot–environment interactions during failure events.

A. Interaction Probability Estimation

Algorithm 1 outlines a general procedure for estimating interaction probabilities under joint failures in robotic manipulators. The algorithm models both individual joint failures and complete system collapse scenarios, where disabled joints cause downstream components to move freely under gravitational forces. Interaction likelihood is computed based on geometric overlap analysis in the horizontal plane between the robot's swept volume and environmental entities.

In MuJoCo, rigid bodies are represented as collections of *geoms*, so Algorithm 1 is applied at the geom level. For

Algorithm 1 Generalized Probability of Interaction Computation

Require: Robot component r_j, environment entity e_i, robot state x_t

1: Compute $b_{r_j} \in \mathbb{R}^{8\times 3}$, the axis-aligned bounding box corners of r_j in global coordinates
2: Extract $\min(b_{r_j}) \in \mathbb{R}^3, \max(b_{r_j}) \in \mathbb{R}^3$, the minimum and maximum coordinate values over the 8 corners and compute projected area on the X-Y plane, A_{r_j}
3: Compute $b_{e_i} \in \mathbb{R}^{8\times 3}$
4: Extract $\min(b_{e_i}) \in \mathbb{R}^3, \max(b_{e_i}) \in \mathbb{R}^3$ and compute A_{e_i}
5: Compute overlap area $A_{\text{overlap}_{i,j}}$ in X-Y plane :
  5a: $\text{min\_maxs}_{i,j} = \min(\max(b_{r_j}), \max(b_{e_i}))$
  5b: $\text{max\_mins}_{i,j} = \max(\min(b_{r_j}), \min(b_{e_i}))$
  5c: $\text{ax\_ov}_{i,j} = \max(\text{min\_maxs}_{i,j} - \text{max\_mins}_{i,j}, 0)$
  5d: $A_{\text{overlap}_{i,j}} = \prod_{a=1}^{2} \text{ax\_ov}_{i,j}[a]$
6: Compute probability:

$$P_t = \max\left(\frac{A_{\text{overlap}_{i,j}}}{A_{r_j}}, \frac{A_{\text{overlap}_{i,j}}}{A_{e_i}}\right)$$

7: **if** $\min(b_{r_j})[3] > \max(b_{e_i})[3]$ **or** $\text{ax\_ov}_{i,j}[3] > 0$ **then**
8: **return** P_t
9: **else**
10: **return** 0
11: **end if**

a robot component r_j, let $\{g_k^j\}_{k=1}^{N_j}$ denote its associated geoms, where N_j is the number of geoms. Similarly, $\{g_l^i\}_{l=1}^{N_i}$ denotes the geoms of environment entity e_i.

Steps 1–4 of the algorithm are computed for each geom: for each g_k^j and g_l^i, we calculate the 8 corners of their axis-aligned bounding boxes, $b_{g_k^j}$ and $b_{g_l^i}$, extract minimum and maximum coordinates, and compute the projected area on the X-Y plane, $A_{g_k^j}$ and $A_{g_l^i}$.

In Step 5, the overlap area between geom pairs (g_k^j, g_l^i) is computed in the X–Y plane. Specifically: (5a) finds the smaller of the two maximum coordinates, (5b) finds the larger of the two minimum coordinates, (5c) computes the difference along each axis, treating negative values as zero to get $\text{ax\_ov}_{l,k}$, and (5d) multiplies these differences along the X-Y axis to obtain the total overlap area $A_{\text{overlap}_{g_l^i, g_k^j}}$.

Body-level areas are then obtained by summing over geoms:

$$A_{r_j} = \sum_{k=1}^{N_j} A_{g_k^j}, \quad A_{e_i} = \sum_{l=1}^{N_i} A_{g_l^i},$$

$$A_{\text{overlap}_{i,j}} = \sum_{l=1}^{N_i} \sum_{k=1}^{N_j} A_{\text{overlap}_{g_l^i, g_k^j}}.$$

Finally, the probability of interaction between r_j and e_i is computed using Step 6. Step 7 ensures that e_i is below r_j along the Z axis by comparing minimum and maximum Z coordinates over all geoms.

While Algorithm 1 is general and capable of handling arbitrary failure types (including joint or actuator failures), in this paper we focus exclusively on *object drop failures*. That is, we simulate the scenario where the robot slips the object it is carrying, without considering full manipulator collapse. This simplifies the experiment while still allowing us to evaluate the impact of failures on the environment.

B. Trajectory Analysis

We evaluate our failure impact framework on four trajectories for a tabletop pick-and-place task in `FailBench` using the *Franka Emika Panda* robot arm.

1) Experimental Setup: The manipulation task requires the robot to pick up an object (green) from the table and place it on top of another object (blue), as shown in Figure 4. We sample intermediate waypoints and employ RRT-Connect to generate feasible trajectories for the Franka Emika Panda robot arm. To assess the risk of object dropping during trajectory execution, we evaluate interaction probabilities using Algorithm 1. Object severity values are predefined in this analysis, with red objects representing high severity entities (value 10) and white objects indicating standard severity levels (value 2).

For each trajectory, we compute the motion cost as the cumulative Euclidean distance between consecutive configurations, $\sum_{i=1}^{N-1} \|q_i - q_{i+1}\|_2$, where $\|\cdot\|_2$ denotes the Euclidean norm and N is the number of waypoints. Safety cost represents the cumulative expected impact across potential drop interactions during failure scenarios, as defined in our optimization objective (Eq. (1), with weight w set to 1). To validate our framework, we simulate each trajectory 60 times with a 25% probability of failure occurrence per trajectory execution, as shown in Figure 5. When a failure occurs, the timestep for object dropping is randomly sampled from the trajectory duration. The observed safety (OBS) represents the average measured safety cost across all simulation runs for each trajectory.

2) Analysis: Table II reveals distinct trade-offs between motion efficiency and safety across the four evaluated trajectories. Trajectory 4 (green) achieves the lowest safety cost (0.236) but incurs the highest motion cost (1.958), resulting in the highest total cost (2.194). Conversely, trajectory 2 (orange) provides optimal balance, achieving both the lowest motion cost (1.280) and total cost (1.71) despite higher safety cost than trajectories 3 (pink) and 4 (green). Trajectory 1 (light blue) demonstrates suboptimal performance with the second-highest motion cost (1.567) and total cost (1.914), while trajectory 3 (pink) serves as a reasonable middle-ground solution with moderate performance across all metrics.

The observed safety (OBS) results from simulation validation generally align with our theoretical safety cost predictions, validating the framework's predictive capability. Trajectory 4 demonstrates both the lowest theoretical safety cost (0.236) and observed safety (0.27), confirming its superior safety performance. Similarly, trajectory 3 shows low observed safety (0.4) consistent with its moderate theoretical safety cost (0.363). However, trajectory 2 exhibits higher observed safety (5.6) than expected from its theoretical cost (0.430), while trajectory 1 shows intermediate observed safety (3.33) despite having a lower theoretical safety cost (0.347) than trajectory 2. These discrepancies highlight the complexity of real failure scenarios and suggest opportunities for refining the interaction probability models.

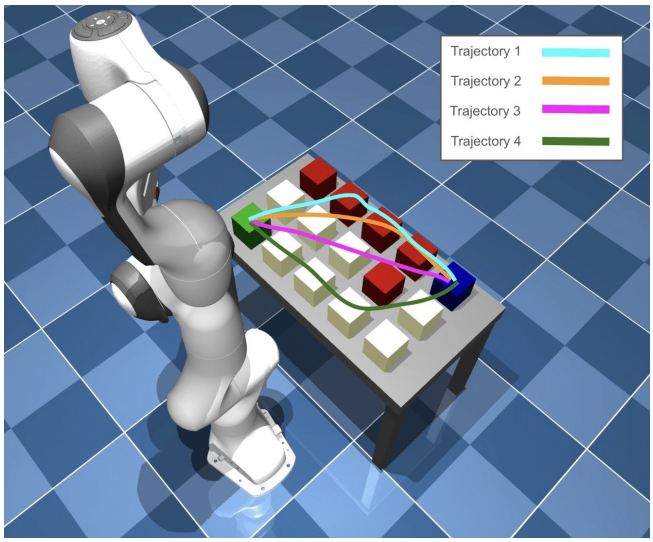

Fig. 4: Trajectories carrying an object during pick-and-place execution. Trajectory 1 (light blue), 2 (orange), 3 (pink), and 4 (green) tradeoffs safety and efficiency.

This analysis demonstrates the practical utility of our failure impact assessment framework for evaluating trajectory safety in the presence of inevitable robot failures. While traditional planners optimize solely for geometric objectives, our framework reveals failure impact costs that could inform future failure-aware planner design. Such quantitative insights could guide the development of planners that explicitly mitigate failure consequences while balancing efficiency objectives. The framework's ability to assess different trajectory candidates provides a foundation for integrating failure impact mitigation into motion planning algorithms, enabling principled decisions that minimize harm when failures occur based on application requirements and environmental vulnerability.

TABLE II: Trajectory comparison showing theoretical safety predictions versus empirical validation through simulation.

Trajectory	Motion Cost	Safety Cost	Total Cost	OBS
1 (light blue)	1.567	0.347	1.914	3.33
2 (orange)	**1.280**	0.430	**1.71**	5.6
3 (pink)	1.359	0.363	1.722	0.4
4 (green)	1.958	**0.236**	2.194	**0.27**

VI. Future Work

While we demonstrate the potential of our safety mitigation framework and `FailBench` to evaluate robot safety through our proposed failure impact assessment, integrating

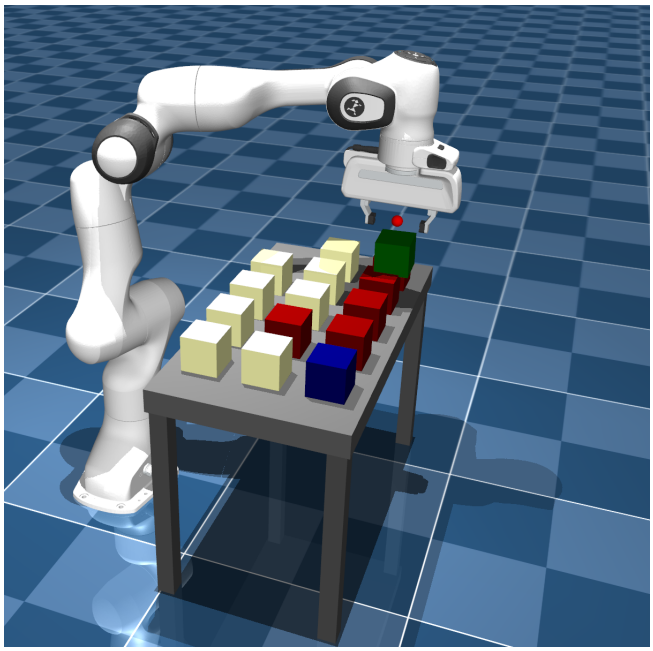

Fig. 5: Visualization of object drop failure during pick-and-place execution. The robot has released the green object mid-trajectory, demonstrating the type of failure scenario in our analysis.

such evaluation capabilities into practical planning systems presents several key challenges for future work.

Environmental Knowledge Requirements. Our framework assumes complete environmental knowledge, including object identity, material properties, and vulnerability characteristics that determine failure impact severity. However, such comprehensive prior knowledge is rarely available in practice. Visual-Language Models (VLMs) offer a promising solution by automatically estimating interaction severity from visual input. VLMs can perceive and annotate unlabeled 3D objects with semantic descriptions including material properties, fragility, and estimated significance [29]. This visual grounding enables more reliable assessment than textual descriptions alone, as VLMs observe actual object appearance, size, and context. Future work could leverage VLMs to automatically infer vulnerability parameters, enabling practical deployment of failure impact assessment.

Failure Interaction Modeling. Our interaction probability model relies on geometric analysis and lacks the complexity needed for realistic failure impact prediction in dynamic environments. Addressing this limitation requires accurate modeling of robot failure behaviors, physical dynamics of manipulated objects, and interactions with dynamic agents [30]. Since `FailBench` enables systematic simulation of post-failure states, future work can leverage this capability to collect comprehensive contact data between robot components and environment entities, enabling learning of probability distributions using neural networks. Physical world models [31] present another promising direction for predicting failure outcomes and component interactions. Recent advances in latent-space Hamilton-Jacobi reachability [32] offer valuable insights for learning failure interaction representations directly from raw observations.

Integration with Motion Planning. A critical challenge lies in effectively incorporating failure impact assessment into motion planning algorithms. Traditional sampling-based planners excel at finding feasible paths but struggle with complex cost landscapes inherent in failure-aware planning. Optimization-based approaches face convergence challenges due to irregular cost surfaces created by discontinuous failure events. Future work should explore planning in latent spaces to achieve smoother cost representations and improved computational efficiency, enabling practical deployment of failure-aware trajectory generation.

Task-Level Failure Awareness. Ultimately, intelligent systems should consider failure impact not only during trajectory generation but throughout high-level task planning. Systems should prioritize actions that mitigate failure impact across entire task sequences [33]. For example, when transporting fragile objects, optimal strategies might first acquire protective equipment to minimize damage risk. Extending our evaluation framework to Task and Motion Planning (TAMP) [34] represents a natural progression toward comprehensive failure-aware robotics, where our benchmarking capabilities can guide the development of safer autonomous systems.

VII. Conclusions

This paper presents a comprehensive framework for evaluating robot safety by explicitly accounting for failure consequences in robotic systems. We introduce a novel formulation that quantifies both the probability and severity of robot-environment interactions during failures, providing a systematic approach to assess how robots handle inevitable malfunctions while protecting their surroundings. Our method evaluates the impact of failures on different entities in the environment, establishing rigorous metrics for safety assessment in service robotics.

To enable systematic evaluation of failure scenarios, we also develop `FailBench`, a MuJoCo-based simulation framework that facilitates controlled study of robot-environment interactions under diverse failure modes, including sensing issues, hardware malfunctions, and actuator degradation. `FailBench` serves as a comprehensive benchmarking platform for robot safety research, featuring curated household environments, standardized motion planning implementations, diverse failure injection modes, and quantitative metrics for failure impact assessment. This framework enables researchers to systematically assess robot failure data and compare different approaches to failure handling and safety-aware robotics.

Our evaluation demonstrates the framework's capability to assess robot safety across various scenarios and planning algorithms. To integrate our framework into motion planning, we identify several challenges and pathways for advancing failure-aware robotics. Integration with Visual-Language Models for automatic scene understanding, learning-based

probability models trained on `FailBench` data, and extension to Task and Motion Planning represent promising directions enabled by our benchmarking foundation.

These contributions establish a critical foundation for safer, more reliable service robots operating in dynamic household environments. By shifting focus from pure failure prevention to systematic consequence evaluation, our work provides the tools and metrics necessary for developing robots that can be rigorously assessed for their ability to gracefully handle inevitable failures while maintaining safety and user trust.

REFERENCES

[1] X. Xiao, J. Dufek, and R. R. Murphy, "Robot risk-awareness by formal risk reasoning and planning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2856–2863, 2020.

[2] L. De Filippis, G. Guglieri, and F. Quagliotti, "A minimum risk approach for path planning of uavs," *Journal of Intelligent & Robotic Systems*, vol. 61, pp. 203–219, 2011.

[3] M. Zabarankin, S. Uryasev, and P. Pardalos, *Optimal risk path algorithms*. Springer, 2002.

[4] M. Ono and B. C. Williams, "An efficient motion planning algorithm for stochastic dynamic systems with constraints on probability of failure." in *AAAI*, 2008, pp. 1376–1382.

[5] C. Cornelio and M. Diab, "Recover: A neuro-symbolic framework for failure detection and recovery," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2024, pp. 12 435–12 442.

[6] 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.

[7] J. Carlson, R. R. Murphy, and A. Nelson, "Follow-up analysis of mobile robot failures," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, vol. 5. IEEE, 2004, pp. 4987–4994.

[8] X. Xiao, J. Dufek, and R. Murphy, "Explicit motion risk representation," in *2019 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE, 2019, pp. 278–283.

[9] B. H. GUO, Z. Yonger, Y. M. Goh, and J.-Y. Lim, "Errors in human-robot interaction accidents: A taxonomy and network analysis," in *International conference on construction engineering and project management*. Korea Institute of Construction Engineering and Management, 2024, pp. 1088–1095.

[10] Z. Liu, A. Bahety, and S. Song, "Reflect: Summarizing robot experiences for failure explanation and correction," *arXiv preprint arXiv:2306.15724*, 2023.

[11] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *2019 18th European control conference (ECC)*. IEEE, 2019, pp. 3420–3431.

[12] K. P. Wabersich and M. N. Zeilinger, "Predictive control barrier functions: Enhanced safety mechanisms for learning-based control," *IEEE Transactions on Automatic Control*, vol. 68, no. 5, pp. 2638–2651, 2022.

[13] M. Gupta and X. Xiao, "T-cbf: Traversability-based control barrier function to navigate vertically challenging terrain," in *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2025.

[14] K. Margellos and J. Lygeros, "Hamilton–jacobi formulation for reach–avoid differential games," *IEEE Transactions on automatic control*, vol. 56, no. 8, pp. 1849–1861, 2011.

[15] I. M. Mitchell, A. M. Bayen, and C. J. Tomlin, "A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games," *IEEE Transactions on automatic control*, vol. 50, no. 7, pp. 947–957, 2005.

[16] Y. Ma, P. Grimmeisen, and A. Morozov, "Case study: Ros-based fault injection for risk analysis of robotic manipulator," in *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2023, pp. 1–6.

[17] A. L. Christensen, R. O'Grady, M. Birattari, and M. Dorigo, "Fault detection in autonomous robots based on fault injection and learning," *Autonomous Robots*, vol. 24, no. 1, pp. 49–67, 2008.

[18] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2012, pp. 5026–5033.

[19] S. Nasiriany, A. Maddukuri, L. Zhang, A. Parikh, A. Lo, A. Joshi, A. Mandlekar, and Y. Zhu, "Robocasa: Large-scale simulation of everyday tasks for generalist robots," *arXiv preprint arXiv:2406.02523*, 2024.

[20] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, S. Nasiriany, and Y. Zhu, "robosuite: A modular simulation framework and benchmark for robot learning," *arXiv preprint arXiv:2009.12293*, 2020.

[21] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE robotics & automation magazine*, vol. 4, no. 1, pp. 23–33, 2002.

[22] K. Zakka, "Mink: Python inverse kinematics based on MuJoCo," May 2025. [Online]. Available: https://github.com/kevinzakka/mink

[23] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 2002.

[24] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects: Steven m. lavalle, iowa state university, a james j. kuffner, jr., university of tokyo, tokyo, japan," *Algorithmic and computational robotics*, pp. 303–307, 2001.

[25] J. Nasir, F. Islam, and Y. Ayaz, "Adaptive rapidly-exploring-random-tree-star (rrt*)-smart: algorithm characteristics and behavior analysis in complex environments," 2013.

[26] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium conference. IEEE international conference on robotics and automation. Symposia proceedings (Cat. No. 00CH37065)*, vol. 2. IEEE, 2000, pp. 995–1001.

[27] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 4569–4574.

[28] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "Chomp: Covariant hamiltonian optimization for motion planning," *The International journal of robotics research*, vol. 32, no. 9-10, pp. 1164–1193, 2013.

[29] R. Kabra, L. Matthey, A. Lerchner, and N. Mitra, "Leveraging VLM-based pipelines to annotate 3d objects," in *Forty-first International Conference on Machine Learning*, 2024. [Online]. Available: https://openreview.net/forum?id=5Pcl5qOOfL

[30] A. Rudenko, L. Palmieri, M. Herman, K. M. Kitani, D. M. Gavrila, and K. O. Arras, "Human motion trajectory prediction: A survey," *The International Journal of Robotics Research*, vol. 39, no. 8, pp. 895–935, 2020.

[31] G. Zhou, H. Pan, Y. LeCun, and L. Pinto, "Dino-wm: World models on pre-trained visual features enable zero-shot planning," *arXiv preprint arXiv:2411.04983*, 2024.

[32] K. Nakamura, L. Peters, and A. Bajcsy, "Generalizing safety beyond collision-avoidance via latent-space reachability analysis," *arXiv preprint arXiv:2502.00935*, 2025.

[33] R. Dhakal, D. M. Nguyen, T. Silver, X. Xiao, and G. J. Stein, "Anticipatory task and motion planning," *arXiv preprint arXiv:2407.13694*, 2024.

[34] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," *Annual review of control, robotics, and autonomous systems*, vol. 4, no. 1, pp. 265–293, 2021.