Improving Human Understanding of Errors Through Enhanced Robot-to-Human Error Reporting

Nesrine Batti[1], Adrian S. Bauer[1], Anne Köpken[1],
Luisa Mayershofer[1], Peter Schmaus[1], Neal Y. Lii[1], Daniel Leidner[1]

***Abstract*— Robots are significantly enhancing a wide range of applications, from industry and elderly care to space exploration. However, the progress remains relatively slow compared to other fields due to the lack of methods for sharing knowledge and experiences, causing repeated efforts and hindering the focus on solving new challenges. To address this, it is necessary that robots are able to share their experiences through a formal approach, eliminating the need to rewrite the software stack for each robot. In this paper, we introduce a novel framework designed to encode and then exchange error-related knowledge from the robot to an operator with a particular focus on planning errors. The proposed framework is based on the OpenUSD standards, employing the Universal Scene Description (USD) format. In our proof-of-concept implementation, when an error occurs, the robot communicates the error situation to the operator, who can then make informed adjustments and teleoperate the robot to address it. This demonstrates effective error communication between a teleoperated robot and an operator.**

I. Introduction

Due to the lack of standardized methods for sharing experiential information among robots, roboticists often develop solutions for similar problems individually over and over again. This is hindering the progress of robotics research and preventing it from reaching its full potential. To address this issue without the need to force everyone to utilize the same software stack, it is essential to create a formal approach that enables both robots and roboticists to share their experiences. Humans understand that learning from mistakes — whether one's own or others' — is highly effective. Following this paradigm, our goal is to facilitate the sharing of error situations among robots to accumulate valuable experiences. In particular, we propose a concept for encoding error situations encountered by robots to promote the exchange of error-related knowledge.

Our contributions include (i) an analysis of existing specification languages and their ability to describe error scenarios, (ii) a concept to model error situation in a formal way, and (iii) a proof of concept implementation using a metaverse approach to share information between agents. We showcase our implementation as we improve the communication of planning errors from a robot to a remote human operator in a space exploration scenario.

II. Related Work

To enable heterogeneous robots to communicate error scenarios to an operator, a robot-agnostic language is necessary. We analyze existing specification languages below.

A. Canonical Robot Command Language (CRCL)

CRCL [1] is a low-level messaging language designed for the control of industrial robots and Automated Guided Vehicles (AGVs). It enables the sending of commands, specifically low-level geometric operations to the robot controller, alongside the reception of status messages. These status messages include both the commands status and the robots joints status.

B. Enhanced Task Specification Language (eTaSL)

eTaSL [2] is a constraint-based task specification language and controller for robot programming. It uses expression graphs to define and control robot manipulation or navigation tasks. The mathematical formulations that are represented in these graphs, such as geometric relations, kinematic equations, and dynamic interactions, enable the execution tasks.

C. Linear Temporal Logic MissiOn Planning (LTLMoP)

LTLMoP [3] toolkit uses Linear Temporal Logic and structured English to construct task plans that are then automatically translated into executable robot controllers.

D. ProbRobScene

ProbRobScene [4] specification language allows the description of the scene graph of the robotic manipulation environment. It enables the specification of the relational constraints between the objects in a scene, as well as specifying the positions and orientations of these objects.

E. Semantic Robot Description Language (SRDL)

SRDL [5] models robot components, actions, and capabilities using Web Ontology Language (OWL). *Components* include hardware components like sensors and actuators, as well as the kinematic chain of the robot. SRDL also facilitates the enrichment of the kinematic model with semantic information. Additionally, components cover software elements such as the control programs, alongside information objects for instance object models. *Actions* are defined as the tasks or operations that a robot can perform. *Capabilities*, refer to services a robot can provide, they are derived from the combination of its components and the actions.

These specification languages primarily focus on encoding information about a single aspect, such as the robot, executed commands, task plans, or the environment. In contrast, our research highlights the crucial need to encode error scenarios to facilitate error information sharing between different agents.

[1] German Aerospace Center (DLR), Robotics and Mechatronics Center (RMC), Münchner Str. 20, 82234 Weßling, Germany

III. Conceptual Analysis of Error Types

In this section, we introduce a taxonomy of errors, defining each type, exploring their root causes, and providing examples. Then, we evaluate the effectiveness of the specification languages discussed in Section II in modeling these errors.

A. Classification of Error Types

One of the most generic classifications of robot errors is the classification presented by Nakamura et al. [6]. Errors are categorized into four classes: *execution errors*, *planning errors*, *modeling errors* and *sensing errors*. Execution errors arise form a problem with the mechanisms of the robot usually leading to a hardware malfunction ranging from temperature changes to aged deterioration. Planning errors occur when the software representation of the system deviates from the actual system. Modeling errors surface when the geometry model used in the software does not accurately reflect the real object. Sensing errors refer to discrepancies in the information collected by the robot's sensors, usually due to incorrect calibrations or sensitivity issues. The classification of Nakamura et al. primarily targets industrial robots which are used in structured manufacturing and assembly lines for repetitive and precise tasks. In contrast, service robots are designed to interact with humans and provide assistance in various situations such as healthcare, domestic chores or space exploration. This leads service robots to operate in an unstructured and dynamic environments, making the effectiveness of pre-programmed tasks limited. To solve this, often times planning algorithms are used, such as *Task and Motion Planning (TAMP)*, to determine the optimal actions for service robots. As the actions and motions may vary from trial to trial, the likelihood of errors increases. In this work, we extend the taxonomy introduced by Nakamura et al. by redefining *execution errors*, *planning errors* and *modeling errors* to address the context of service robots to better suit their characteristics and functionalities. Please note that sensing errors are not detailed further, as they may be the root cause of other error types. We now extend the definition of each error type, list possible causes, and provide examples.

Modeling errors occur when the robot internal model of the robot itself or the model of the world (or both) do not accurately reflect the real world. This can happen due to discrepancies in initial conditions, different object states, or changing environments, where the root cause may actually be a sensing error. Additionally, incorrect or incomplete models of robot kinematics or robot dynamics may be the cause of a modeling error. For example, if a robot is tasked with opening a door, the planned actions might be to navigate to the door, grasp the handle, and open it. However, if the door is locked with a key and the robot believes it to be unlocked, the robot will fail because it was not expecting to unlock the door at first.

Planning errors occur when the planner fails to produce a plan that achieves the desired goals. These errors can result from collision restrictions, joint limits, and kinematic singularities. Again, the root cause may be a sensing error, yet this is not of relevance for the overarching problem. For example, a robot tasked with grasping a mug might fail to produce a collision-free plan due to the planner's inability to avoid singular configurations while considering joint limits and obstacles.

Execution errors occur when the ensemble of actions generated by the planner are feasible but fail to produce the expected outcome in the real world. These errors can be caused by uncertainties in the environment (which may again be related to sensing errors), inaccuracies in actuation, or hardware malfunctions. For example, if a robot is tasked with grasping a mug, an unforeseen complication might arise during execution, causing the mug to slip from the grasp of the robot.

To accurately encode different types of errors and subsequently share information about them, it is essential to express relevant information about their key components. These components can be categorized as follows:

- Scene description: Provides a detailed representation of the robot's belief state, describing the geometries of different objects, their positions within a global reference coordinate system, and their parent-child relationships.
- Object semantics: Offers contextual information about the objects in the scene, necessary for decision-making. Such as the object's type and functional properties.
- Kinematics model: Outlines the kinematic chain of the links and joints of the robot.
- Dynamics model: Builds upon the kinematics model by incorporating dynamic parameters.
- Task plan: Specifies the actions the robot must follow to achieve a desired goal on a symbolic level.
- Geometric expressions: Represent the different joints' spatial configurations attempted in simulation and then in real-world conditions to execute a task.
- Joints status: Includes the positions of the joints and the applied torques.
- Commands status: Provides detailed information about each action within the task at a granular level.

By categorizing and detailing these components, we can effectively encode error information and facilitate the sharing of this knowledge among robots and roboticists. In the following, we analyze which specification language, introduced in Section II, is able to express which error key component.

B. Comparative Analysis of Specification Languages

In Table I we present a comparative analysis of specification languages used in robotics, outlining their abilities to model the different error types. Our analysis reveals that none of the examined frameworks in Table I offer a solution. Each falls short of modeling the full spectrum of the three categories of error types—modeling, planning, and execution errors. Addressing the deficits observed, we propose a novel concept in Section IV.

IV. A Preliminary Concept to Share Error Knowledge Between Agents

In this section, we detail our concept to share error-knowledge between agents, focusing on the design justi-

	Modeling Errors		Planning Errors		Execution Errors			
	Scene Description	Objects Semantics	Kinematics	Dynamics	Task Plan	Geometric Expressions	Joints Status	Command Status
CRCL	–	–	–	–	–	✓	✓	✓
eTaSL	–	–	✓	✓	–	✓	–	–
LTLMoP	–	–	–	–	✓	–	–	–
ProbRobScene	✓	–	–	–	–	–	–	–
SRDL	✓	✓	✓	–	✓	–	–	–

TABLE I: Comparative analysis of specification languages in robotics and their ability to model the different error types.

fication of the Universal Scene Description (USD) format and introducing the Robotic Error Extension for the USD language.

A. A Metaverse Concept to Share Error Knowlegde

The concept of the metaverse, an interconnected virtual space that merges physical and digital realities, holds significant potential for enhancing the interaction between human operators and robotic systems. By bridging the physical world with the digital world, the metaverse can provide unprecedented access to the internal states of robots, enabling more intuitive control, monitoring, and collaboration. This integration can facilitate seamless information exchange, enabling humans to better understand and influence robot behavior, thus optimizing overall team performance.

USD, developed for describing complex 3D scenes, has become the de facto standard in 3D modeling and is widely adopted in tools like Maya, Blender, and robotics simulators such as Isaac Sim [7], Unity3D [8], and Gazebo [9]. USD organizes 3D data into hierarchical scene graphs, where basic elements, so-called *prims*, can hold other child prims, attributes, and relationships. *Attributes* define the characteristics and behavior of prims and can vary over time, encompassing properties like location, orientation, and visual appearance. *Relationships* establish connections within the scene, building a comprehensive scene hierarchy. Supported by the evaluation of Agbossou [10], we adopt the USD file format as the foundation of our concept. This evaluation highlights the superior capabilities of USD in encoding 3D geometry, attributes, semantics, and textures necessary for modeling various types of errors.

In robotics, USD facilitates the modeling of robots and their environments. The hierarchical structure of USD make it a robust choice for robotics applications, surpassing other 3D data formats such as DXF, Collada, and X3D. In particular, robots can be represented as nested hierarchies of prims that articulate their kinematic structures, with semantic information added to provide context to both objects and robot components. USD's extensibility allows for custom schema classes to define unique properties and structures, enabling the modeling of any 3D scene component. USD files can be generated and manipulated programmatically, allowing for dynamic updates to scene attributes without the need for new files, optimizing the maintenance and updating of 3D scenes in response to environmental changes.

B. Proof-of-Concept Implementation

In this section, we present the proof-of-concept implementation of our approach, the *Robot Error Extension for Universal Scene Description (REUSD)*, to encode error situations in USD. Focusing on planning errors for now, the REUSD framework is triggered once a planning error occurs. Initially, REUSD converts the URDF model of the robot to a USD model and semantically annotates the relevant manipulators. Using telemetry data, REUSD sets the robot to its current joints configuration and assigns an attribute to each joint, indicating its status as either OK or in Failure mode.

When a planning error occurs, the motion planner reports the encountered planning error, specifying whether it is a collision error or an unreachability one, the manipulator in question, all the joint configurations attempted during the search for a feasible solution, and for each of them the collision report detailing which robot body part and environment entity are colliding. REUSD then requests the current *World State Representation (WSR)* [11], i.e. the environment belief state of the robot and converts it into a USD scene graph. The REUSD framework queries the 6D pose of the robot in world coordinates, incorporates a reference of the robot USD model into the scene graph of the environment USD model, and positions the robot accordingly.

Based on the collision report, REUSD assigns semantic annotations to the colliding robot body part and the object. The object is tagged with an *inCollision* attribute naming the robot part it collides with, and the robot part is similarly tagged, pointing to the object in collision.

REUSD then parses the generated USD scene, authors a red material prim, and binds it to any prim that possesses an *inCollision* attribute. The final USD file includes the scene graph of the environment, the robot in the reported configuration for solving the task, and the colliding entities highlighted in red to visually indicate a collision error to the operator. This configuration results from simulating the task, during which a wide array of different poses were systematically evaluated by the planner. While a USD file can be created for every configuration tested, our framework generates one USD file from one attempted configuration for the most collided robot body part. Listing 1 shows a section of one of the auto-generated USD file. It defines a Xform for the *Solar Panel Unit 3 (SPU3)* object, specifying its position and orientation with a transformation matrix and marking it as in collision with the robot body part *right thumb finger distal*.

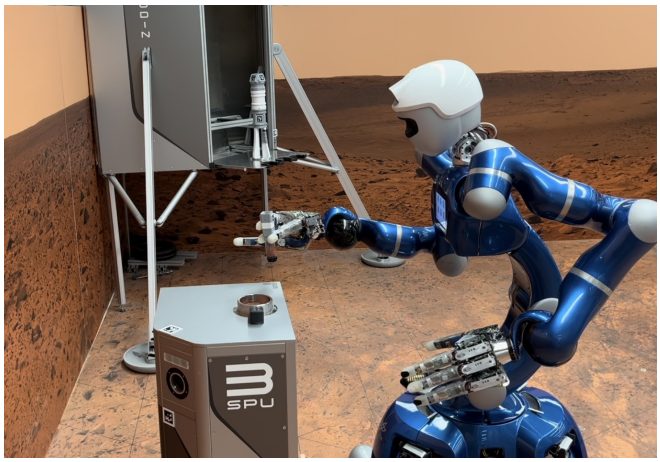

Fig. 1: The humanoid robot Rollin' Justin picking up a sample tube in a Martian mock-up environment.

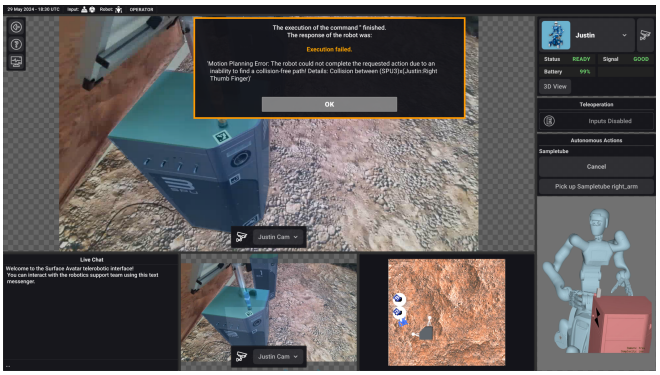

Fig. 2: The visualization of a planning error as it is displayed on the UI of the operator. The REUSD-generated error description (see Listing 1) can be visualized as an interactive 3D-view as it is suggested on the bottom right.

```
def Xform "SPU3"
{ custom string InCollision = "
      right_thumb_finger4_distal"
  matrix4d xformOp:transform =
  ( (1,0,0,0), (0,1,0,0), (0,0,1,0), (4.4,3.6,-0.02,1) )
  uniform token[] xformOpOrder = ["xformOp:transform"]

  def Mesh "SPU3_mesh" (
  prepend references = @./SPU3_object.usda@
  )
  {
    rel material:binding = </Materials/
        Material_Color_252_88_88>
    matrix4d xformOp:transform =
    ( (0,1,0,0), (-1,0,0,0), (0,0,1,0), (0,0,0,1) )
    uniform token[] xformOpOrder = ["xformOp:transform"]
  }
}
```

Listing 1: A snippet of the REUSD-generated error description defining the transformation and mesh for a Solar Panel Unit (SPU), including *collision* information and material bindings

V. Evaluation

We demonstrate our proof-of-concept implementation in a remote operation scenario originating from the Surface Avatar experiment [12]. In this experiment, the humanoid robot Rollin' Justin [13] is remote controlled by an astronaut from aboard the International Space Station (ISS) in preparation for future Mars exploration scenarios. Considering the limited situation awareness of the operator in this setup, it is crucial to communicate errors as efficient as possible.

For example, if the operator commands the robot to execute a certain high-level command (e.g. `pick sample_tube right_arm`), the robot may face a planning error, e.g. due to a collision. As the robot fails to complete the assigned task, a notification panel appears on the User Interface (UI) stating "Execution Failed". In the past, this information came without offering further details about the cause of the failure. Using our REUSD approach, a USD file encoding the error scenario is automatically generated, sent, and displayed on the UI as shown in Fig. 2. As a result, the operator gains a clearer understanding of the cause of the error, including a visual representation of which robot body part is colliding with which environmental object.

VI. Discussion and Future Work

In this workshop paper, we have demonstrated a preliminary concept to communicate different types of errors to a remote operator by leveraging a metaverse approach. The REUSD proof-of-concept implementation was showcased in a remote Human-Robot-Interaction scenario, where a planning error was displayed to an operator to enhance situation awareness in the presence of planning errors experienced by a remote robot.

Going forward, we plan to extend REUSD to cover a full range of error types. In addition, we intend to develop a recommendation system where the robot suggests corrective actions based on the type of error encountered. For instance, upon detecting a planning error, the robot could propose adjustments like repositioning and retrying, or attempting to pick up the object in question with another manipulator.

As a long-term goal, we aim to implement the communication of error situations from robot to robot. By sharing information about encountered errors, robots can adjust their operations based on the experiences of other robots within the network. This poses significant challenges, such as belief divergence when robots share their perceived states of the environment, raising several important research questions: How can robots process and use information received from other robots? How can a robot distinguish correct information from erroneous beliefs? And ultimately, how can robots and humans mutually benefit from shared error information in both directions?

In conclusion, our preliminary work shows promising potential for improving error communication in robotics. By expanding this approach and addressing the associated challenges, we hope to enhance the reliability and efficiency of robotic systems, ultimately fostering greater collaboration and understanding between robots and their human operators.

REFERENCES

[1] F. Proctor, S. Balakirsky, Z. Kootbally, T. Kramer, C. Schlenoff, and W. Shackleford, "The canonical robot command language (crcl)," *Industrial Robot: An International Journal*, vol. 43, pp. 495–502, 08 2016. doi: 10.1108/IR-01-2016-0037

[2] E. Aertbeliean and J. De Schutter, "etasl/etc: A constraint-based task specification language and robot controller using expression graphs," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 1540–1546. doi: 10.1109/IROS.2014.6942760

[3] C. Finucane, G. Jing, and H. Kress-Gazit, "Ltlmop: Experimenting with language, temporal logic and robot control," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 1988–1993. doi: 10.1109/IROS.2010.5650371

[4] C. Innes and S. Ramamoorthy, "Probrobscene: A probabilistic specification language for 3d robotic manipulation environments," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 9446–9452. doi: 10.1109/ICRA48506.2021.9562038

[5] L. Kunze, T. Roehm, and M. Beetz, "Towards semantic robot description languages," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 5589–5595. doi: 10.1109/ICRA.2011.5980170

[6] A. Nakamura, K. Nagata, K. Harada, and N. Yamanobe, "Estimation and categorization of errors in error recovery using task stratification and error classification," *Journal of Robotics, Networking and Artificial Life*, vol. 4, pp. 163–167, 2017. [Online]. Available: https://doi.org/10.2991/jrnal.2017.4.2.13. doi: 10.2991/jrnal.2017.4.2.13

[7] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac gym: High performance gpu-based physics simulation for robot learning," 2021.

[8] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents," 2020.

[9] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154 vol.3. doi: 10.1109/IROS.2004.1389727

[10] I. AGBOSSOU, "Multi-scalar urban digital twin design: Architecture and openusd standards based methodology," *PriMera Scientific Engineering*, Dec 2023. doi: 10.56831/psen-04-100

[11] A. S. Bauer, P. Schmaus, F. Stulp, and D. Leidner, "Probabilistic effect prediction through semantic augmentation and physical simulation," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 9278–9284.

[12] P. Schmaus, D. Leidner, T. Krüger, J. Grenouilleau, A. Pereira, A. Bauer, N. Bechtel, S. B. Gomez, A. Koepken, F. Lay, M. Sewtz, N. Batti, E. Ferreira, E. den Exter, R. Bayer, B. Pleintinger, R. Holderried, P. Pavelski, and N. Y. Lii, "On realizing multi-robot command through extending the knowledge driven teleoperation approach," in *Proceedings of the Inter- national Astronautical Congress, IAC, International Astronautical Federation*, September 2022.

[13] C. Borst, T. Wimbock, F. Schmidt, M. Fuchs, B. Brunner, F. Zacharias, P. R. Giordano, R. Konietschke, W. Sepp, S. Fuchs, C. Rink, A. Albu-Schaffer, and G. Hirzinger, "Rollin' justin - mobile platform with variable base," in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 1597–1598. doi: 10.1109/ROBOT.2009.5152586