A Framework for Robot Self-Assessment of Expected Task Performance

Tyler Frasca and Matthias Scheutz

***Abstract*—We propose a self-assessment framework which enables a robot to estimate how well it will be able to perform a known or possibly novel task. The robot simulates the task to generate a state distribution of possible outcomes and determines (1) the likelihood of overall success, (2) the most probable failure location, and (3) the expected time to task completion. We evaluate the framework on the "FetchIt!" mobile manipulation challenge which requires the robot to fetch a variety of parts around a small enclosed arena. By comparing the simulated and actual task resulting state distributions, we show that the robot can effectively assess its expected performance which can be communicated to humans.**

***Index Terms*—Human-robot teaming, methods and tools for robot system design, simulation and animation.**

I. Introduction

UNDERSTANDING what robots can or cannot do is essential for fluid human-robot interactions as well as effective robot task performance. Yet, it is often difficult if not impossible for humans to make inferences about a robot's expected performance on a task they have not seen the robot perform because they lack detailed information about the robot's action success and failure probabilities. A robot's ability to engage with a human in dialogues about its task performance would therefore significantly improve human mental models of robot operation and thus the likelihood of mission success (as robots would then likely be deployed in ways that play to their strengths). Prerequisite to having such dialogues is a robot's ability to self-assess, i.e., introspect its operation, monitor its progress, and generate performance predictions.

Additionally, a robot with self-assessment capability can ultimately improve its performance and human counterparts on current and future tasks, because it will be able to predict and resolve potential issues prior to failure. For example, the robot could inform a human of potential problems and propose alternatives, or simply ask for help. Past work on self-assessment has typically focused on low-level sensorimotor error and fault detection [1], [2], [3], [4]; less work however, has focused on self-assessment for task performance which is critical for human-robot collaboration. Consider a robot tasked with fetching supplies for a medical care unit in a disaster relief mission, or gathering a set of tool to repair a broken component on a ship. To complete these tasks, the robot needs to execute a combination of various manipulation, perception, and navigation actions. Therefore, if a robot wants to predict its expected overall performance, then it needs to consider its performance across all action types. Understanding expected task performance, including probability of success, time-to-completion, and failure locations, requires the assessment of all parts of a task execution, not just a singular sensorimotor component.

We thus present a comprehensive framework that will enable more informed human-robot interactions by allowing humans to ask questions about the robot's task performance including: 1) What is the likelihood of completing a task? 2) Which actions are most likely to fail? 3) How long will it take to complete an action? We evaluate the framework by comparing expected vs actual performance measures in a challenging mobile manipulation task. The proposed framework answers these questions by introspecting on the actions that need to be performed to accomplish the task. It enables more sophisticated task executions where robots proactively determine changes of their plans, based on their self-assessment, which they can communicate and justify to humans by pointing to higher success probabilities of the alternatives.

II. Related Work

Self-Assessment is a process for humans to know their capabilities and predict expected performance from past experiences. It can be beneficial during teaching, learning, planning, and execution. Developing robots with self-assessment capabilities will provide them with the ability to know their own limits, predict their expected performance on a task, and ultimately improve human-robot interaction. Again, consider the robot tasked with fetching medical supplies. It needs to execute several actions in order to complete the overall task, so to assess its expected task performance, the robot needs to consider its performance across all actions.

Although there has not been much work on robot self-assessment per se, related work has focused on predicting failures for autonomous driving [4], [5], anomalous event detection in manipulation [3] and navigation [6], human-robot interaction to project a robot's intention [7], and architectural configurations [8]. However, most of these approaches focus on lower-level sensorimotor information instead of higher-level task information.

For example, [2] propose a framework for introspecting on an autonomous robot's vision system to learn a model of the

Manuscript received 23 June 2022; accepted 15 October 2022. Date of publication 2 November 2022; date of current version 14 November 2022. This letter was recommended for publication by Associate Editor H. Mori and Editor T. Ogata upon evaluation of the reviewers' comments. This work was supported by the U.S. Office of Naval Research under Grant N00014-18-1-2503 *(Corresponding author: Tyler Frasca.)*

Tyler Frasca is with the Department of Computer Science, Tufts University, Boston, MA 02155 USA (e-mail: tyler.frasca@tufts.edu).

Matthias Scheutz is with the Faculty of Computer Science, Tufts University, Boston, MA 02155 USA (e-mail: matthias.scheutz@tufts.edu).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2022.3219024, provided by the authors.

Digital Object Identifier 10.1109/LRA.2022.3219024

2377-3766 © 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: TUFTS UNIV. Downloaded on February 07,2023 at 21:12:05 UTC from IEEE Xplore. Restrictions apply.

expected performance and determine its ability to rely on what it detects. Similarly, [1] develop a self-evaluation vision system, ALERT, to predict if it is likely to produce an unreliable response to an input. If the system predicts an unreliable response, then it can attempt to fail gracefully.

Similarly, [9] concentrate on acquiring object models and when to extend the model through probabilistic measures including, observed detection success, predicted detection success, and model completeness. By knowing its limits, the robot can improve performance in future tasks by gathering more knowledge. [10] present an approach which allows a robot to assess how good part recognition and pose estimation are. From there the robot is able to determine if it should continue with task execution or if it should ask for assistance from human. [6] develop a model for self-assessment based on novelty detection techniques. The author presents a neural architecture to learn sensorimotor contingencies from two navigation strategies then detect extraneous sensorimotor patterns in novel situations. [11] extends this architecture to improve skills by asking humans for help.

Though the aforementioned frameworks can assess aspects of sensorimotor components, they are each limited to their specific function and are unable to assess overall task proficiency when a combination of different types actions are required, such as is the case for fetching supplies. [12] and [13] propose initial frameworks for reasoning about task proficiency. [12] focus on detection of proficiency by assessing if a robot's actions take it closer to the desired goal during execution, whereas [13] concentrate on human-robot dialogues about task performance before, during, and after execution. Similar to both of these works, our approach focuses on self-assessment for task performance. However, our approach assesses the entire task by simulating the task and generating detailed information about how well the robot complete the task. Specifically, we present a framework which simulates task execution and assesses the resulting action traces and state distributions. The robot can then determine the likelihood of success, where it foresees issues, and the time-to-completion.

III. The Self-Assessment Framework

The three goals of the self-assessment framework are to allow robots to determine: 1) the likelihood of plans or action sequences completing successfully, i.e., to reach a goal state from a given initial state, 2) which action is most likely to fail and why, 3) the expected time to completion. The challenge here is that the system will typically not have sufficient knowledge about the various states it can be in upon completing any one of its actions, especially if the state space is large as is typically the case with robots. Moreover, it will often not be possible for the robot to explore particular actions in certain states (e.g., because the robot cannot easily get itself into the state or make necessary environmental changes to bring about the state in which it intends to execute an action). Rather, what the robot can do is to keep track of the *relevant* pre-conditions of any action it performs, i.e., the conditions that enable action execution (irrespective of other environmental factors, even ones that might have modulatory influence on action execution). And it can track the effects its action execution has on *relevant* post-conditions (i.e., the outcomes the action should accomplish with additional side effects that might negate pre-conditions), recording the frequencies of ending up in these states as it keeps executing the same action. Doing this every time an action is executed will allow the robot to generate conditional probability distributions that reflect action outcomes and directly link pre- and post-conditions irrespective of other aspects of the environment. From the robot's perspective, these conditional probabilities are the closest it can get to the true state transition function of the environment and it is the best model it can use to estimate its ability to successfully complete a plan or action sequence. In the following, we will make these ideas formally precise, starting with preliminaries, followed by the definition of "success probability" and "success duration" for plans or action sequences.

A. Preliminaries

Let $\mathcal{L}$ be a first-order language with a finite number of individual constants, action symbols, and predicate symbols defined over finite models which is used to formalize a robot's operating domain, i.e., the task environment in which the robot operates with all of its objects and attributes, relevant relations, and applicable actions with their parameters. We define Υ to represent the set of atomic propositions in $\mathcal{L}$ (without action expressions) describing all possible aspects of the environment, including the physical states of the robot. We assume that the robot has a repertoire of temporally extended "primitive actions" that can be characterized by four formulas in $\mathcal{L}$ in disjunctive normal form (DNF):[1] *pre-conditions* Π that if met will enable the execution of the action, *operating conditions* Ω that are necessary (but not necessarily sufficient) for the successful execution of the action, *success conditions* Σ which will be true when the action completes successfully, and *failure conditions* Φ which reflect the different world states the system can be in when the action fails.[2] We can then use primitive actions to define *action scripts* which are extensions of "programs" in quantifier-free first-order dynamic logic with empty assignments, see [14]. Specifically, consider the dynamic logic formula $\Pi^\alpha \rightarrow \langle\alpha\rangle\Psi^\alpha$ which states that if Π^α (the pre-conditions) are true, then one execution path of the program α will make Ψ^α (the post- conditions) true. If we consider all Ψ^α for which $\Pi^\alpha \rightarrow \langle\alpha\rangle\Psi^\alpha$ and single out the one $\Psi^\alpha = \Sigma$ that represents the intended action outcome, the remaining $\Psi^\alpha = \Phi$ (with unintended outcomes) represent the different failure conditions.

An *action script* α is thus a six-tuple $\langle args, body, \Pi^\alpha, \Omega^\alpha, \Sigma^\alpha, \Phi^\alpha\rangle$ consisting of a sequence of action parameters $args = (x_1, x_2, \ldots, x_n)$ and a sequence $body = \alpha_1(\ldots); \alpha_2(\ldots); \ldots; \alpha_n(\ldots)$ of primitive actions or action scripts $\alpha_i(\ldots)$ (with any number of parameters, including x_i), or "conditional gotos": IF ϕ GOTO n where n is the n-th element in the sequence (with the meaning that execution is resumed at the n-th entry in the script if ϕ is true). In addition, the four formulas represent pre-conditions Π^α, success Σ^α and failure Φ^α conditions, and operating conditions Ω^α.

Finally, we formalize the robot's task environment as a *Markov Decision Process* (MDP): $\mathcal{M} = \langle S, s_0, A, \mathcal{A}(s), \mathcal{T}\rangle$

[1]The disjunctive normal form is more permissive than a simple conjunction of proposition in that, by consisting of the disjunction of conjuncts, it allows for the specification of alternative states.

[2]Note that operating conditions are different from pre-conditions which might change as they action is being executed. For example, the pre-condition for "pouring" might be $filled(cup)$, the post-condition might be $empty(cup)$, and the operating condition might be $holding(cup)$, and as soon as pouring starts the cup will be no longer $filled$, even though it will not be immediately $empty$ either.

Authorized licensed use limited to: TUFTS UNIV. Downloaded on February 07,2023 at 21:12:05 UTC from IEEE Xplore. Restrictions apply.

where

- $S_{\mathcal{M}}$ is a finite set of (consistent) states $s \subseteq \Upsilon$ (to make it possible to track all true propositions which are needed later to define pre- and post-conditions for a task).
- $s_0 \in S_{\mathcal{M}}$ is the initial state of before task execution.
- $A_{\mathcal{M}}$ is a finite set of robot actions.
- $\mathcal{A}(s)_{\mathcal{M}} : S_{\mathcal{M}} \rightarrow 2^{A_{\mathcal{M}}}$ is mapping each state $s \in S_{\mathcal{M}}$ to the set of actions available to the agent at s (with all possible argument bindings based on the objects in $S_{\mathcal{M}}$).
- $\mathcal{T}_{\mathcal{M}} : S_{\mathcal{M}} \times A_{\mathcal{M}} \times S_{\mathcal{M}} \rightarrow [0,1]$ is the conditional probability $P(s'|s,a)$ of transitioning to s' when executing action a in s.

We drop the subscript $\mathcal{M}$ when there is no ambiguity about the components of MDP referred to and drop the superscripts α if the reference to the action script is clear from context.

B. Success Probability and Duration of Action Scripts

Given an action script α and a task environment $\mathcal{M}$, we would like to determine the *success probability distribution* $\mathbf{P}(\Sigma|\alpha,\Pi)$, a central notion of performance self-assessment. We will define this distribution inductively, starting with primitive actions $a \in A$ (that cannot be decomposed into smaller actions) and then defining it for action scripts. Π is in DNF, i.e., of the form where each disjunct Π_i represents an alternative pre-condition for action execution and each conjunct $\pi_{i,j}$ represents *aspects* of a state $s \in S$ (not necessarily the full state) that s must meet in order for a to be executable (in s), i.e., for any s such that $a \in \mathcal{A}(s)$, $\{\pi_{i,1}, \pi_{i,2}, \ldots, \pi_{i,k_i}\} \subseteq s$ must hold for a to be executable. We will thus use disjuncts (of conditions) and sets of conjuncts (of disjuncts) interchangeably (as the sets are always finite).

We next split conjunctions of success conditions Σ_j into conditionally independent subsets (conjunctions) Σ_{j,c_j} of conditions (this could result in as many sets as there are conjuncts, or possibly keep the original formula if there are no conditionally dependent conjuncts). For each $\Sigma_{j,c_j} \in \Sigma_j$ we then consider $P(\sigma_{j,c_j}|a,\Pi_i)$, i.e., the conditional probability that executing a will make Σ_{j,c_j} true given the particular pre-condition. Note that each Σ_{j,c_j} and Π_i amounts to a partitioning of S with $\Pi = [s]_{\Pi_i \subseteq s}$ and $\Sigma_{j,c_j} = [s']_{\Sigma_{j,c_j} \subseteq s'}$. Hence, if $\mathcal{T}$ were available to the robot, it could simply obtain the distribution $\mathbf{P}(\Sigma_{j,c_j}|a,\Pi_i)$ from the individual probabilities $\mathcal{T}(s,a,s')$ for all s and s' such that $\Pi_i \subseteq s$ and $\Sigma_{j,c_j} \subseteq s'$, i.e., from the aspects of states s/s' that are not determined by Π_j/Σ_{j,c_j}. However, as stated earlier, $\mathcal{T}$ is likely not available, hence the robot needs to determine $\mathbf{P}(\Sigma_{j,c_j}|a,\Pi_i)$ by tracking executions of a that make Σ_{j,c_j} true when Π_i holds, for all Π_i and Σ_{j,c_j} of all actions $a \in A$, or at least those actions that it will need to execute for the tasks under consideration.

Note that since the success conditions are conditionally independent, we obtain

$$\mathbf{P}(\Sigma_j|a,\Pi_i) = \prod_{c_j} \mathbf{P}(\Sigma_{j,c_j}|a,\Pi_i)$$

i.e., the product of the probability distributions of the conditionally independent success conditions. Doing this for all combinations of disjuncts Σ_j and Π_i then yields $\mathbf{P}(\Sigma|a,\Pi)$ for $a \in A$. Note that for actions a with multiple disjunct pre-conditions we can get better success estimates by eliminating false disjuncts Π_f from the distribution $\mathbf{P}(\Sigma|\alpha,\Pi-\Pi_f)$ (because the robot cannot be in those states). We can then also directly extend this to action scripts α consisting only of action sequences $\alpha_1(\ldots);\alpha_2(\ldots);\ldots;\alpha_n(\ldots)$:

$$\mathbf{P}(\Sigma^{\alpha}|\alpha,\Pi^{\alpha}) = \prod_i \mathbf{P}(\Sigma^{\alpha_i}|\alpha_i,\Pi^{\alpha_i})$$

since each action α_i will need to succeed for the overall script execution to succeed.

Tracking the effects of actions given certain pre-conditions as compared to the classical state transitional model enables focusing on the aspects of the states that are relevant for an action, whereas the classical state transition model leads to an explosion of conditional probabilities because of irrelevant aspects of states. Consider an action α that is dependent on only the pre-condition π and has only the success effect σ. The traditional approach to learning a state transition model would have to learn the conditional probabilities $P(s'|s,\alpha)$ for all pairs of s and s' and could not tell them apart, whereas our approach only requires one value $P(\sigma|\pi,\alpha)$.

For scripts containing conditional "goto" instructions, we can generate new scripts from execution traces, i.e., if a conditional instruction fails, we simply skip it in the calculation, and if it succeeds, we continue to factor in the actions from the "goto" location. For example, if $\alpha_3 = \text{IF } \phi \text{ GOTO } 5$, then α_4 is omitted in the above calculation. If "goto" instructions lead to loops, then the probability distribution of the action sequence within the loop needs to be repeatedly factored in the above calculation.

Finally, we are also interested in defining the *expected duration* $\mathbb{E}[\Delta(\alpha)]$ of α where $\Delta()$ is the duration distribution of executions of α which is given by $\sum_{\alpha_i \in \alpha} \mathbb{E}[\Delta(\alpha_i)]]$, i.e., the sum of the respective expected distributions of the actions α_i in α. To calculate these values, the robot again has to keep track of its action executions, in this case execution times for all primitive actions $a \in A$ (or again, only those that are needed in the task), from which it will get the distributions $\Delta(a)$ needed for calculating the overall duration distribution and thus the expected execution time. Analogous to keeping track of the execution success of a for each Π_i and Σ_{j,c_j} we can refine time keeping by tracking execution times separately for the specific pre-conditions Π_i and success conditions Σ_{j,c_j}, i.e., $\Delta(a,\Pi_i,\Sigma(s)_{j,c_j})$.

C. Simulating Action Scripts

We assume the robot learns through experience or is provided $\mathbf{P}(\Sigma^a|a,\Pi_i^a)$ for all task-relevant actions $a \in A$ as well as $\Delta(a,\Pi^a)$ and can access these distributions. We do not assume that the robot has the relevant failure distribution $\mathbf{P}(\Phi^a|a,\Pi_i^a)$ even though the script can be used to update it, as we will show below. We can then use the success and duration distributions for primitive actions to sample from the overall distributions of executions of action script α using *simulations* as follows:

Notice, in Algorithm 1, the recursive invocation of "Simulate" in line 21 whenever an action α is not a primitive. Also noteworthy is that after determining the applicable pre-conditions (leaving out disjuncts that are not true in s) and sampling from the respective success and duration distributions, the state "update" in line 32 requires the robot to integrate the new state information Σ^{α_k} with its current simulation state s. This means removing any proposition in s that is inconsistent with propositions Σ^{α_k} which can be accomplished in the implementation by explicitly tracking which positions need to be removed (e.g., in the manner of STRIPS operators).

Authorized licensed use limited to: TUFTS UNIV. Downloaded on February 07,2023 at 21:12:05 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1: Simulate(α, s_0).

input: α, the action script
  s_o, the initial state of the environment
output: success, duration, final state, and action trace
$s \leftarrow s_0$, initialize the current state
$d \leftarrow 0$, initialize the current duration
$\tau \leftarrow createTreeNode(\alpha)$, action trace
outcome $\leftarrow$ success
$c \leftarrow \Pi^\alpha$, get the current pre-condition
if $c \notin s$ **then**
  return failure,d, s, τ
end if
if primitive(α) **then**
  $d^\alpha \leftarrow$ sample from $\Delta(\alpha, \Pi^\alpha)$
  $d \leftarrow d + d^\alpha$
  $p^\alpha \leftarrow$ sample from $\mathbf{P}(\alpha, \Pi^\alpha)$
  if $p^\alpha \notin \Sigma^\alpha$ **then**
    outcome $\leftarrow$ failure
  end if
else
  for all $\alpha_k \in body^\alpha$ **do**
  $(outcome, d^{\alpha_k}, s, \tau^{\alpha_k}) \leftarrow$ Simulate(α_k, s)
  $d \leftarrow d + d^{\alpha_k}$
  $addNode(\tau, \tau^{\alpha_k})$
  if outcome $\neq$ success **then**
    return outcome, d, s, τ
  end if
end for
if outcome $=$ success **then**
  $p^\alpha \leftarrow \Sigma^\alpha$
end if
end if
$s \leftarrow$ update s with p^α
return outcome, d, s, τ

While the above simulation cannot account for unexpected outcomes (errors due to aspects of environmental states that are not modeled explicitly in the pre-, operating-, and success conditions), it is able to generate different distributions of failure states based on execution failures that occur at different places in the script. Specifically, whenever the simulation of a script α aborts due to (simulated) execution failures, the resultant failure state s can be used to update $\mathbf{P}(\Phi^\alpha|\alpha, \Pi^\alpha)$ (where the initial distribution can be obtained from the failure distributions of primitive actions to the extent that they are available). If enough simulations of α are run, the robot will be able to collect comprehensive data resulting in a distribution of possible failure states of α which can be used for various types of analyses.

D. Refined Success and Failure Discovery Through Simulation

One argument against repeatedly running simulations on action scripts would be to point out that one could simply calculate the various probabilities of failures at different points in the action sequence $\alpha_1; \alpha_2; \ldots; \alpha_k; \ldots; \alpha_n$ of a script α using $\mathbf{P}(\Phi^{\alpha_k}|\alpha_k, \Pi^{\alpha_k}) \cdot \prod_{i=1}^{k-1} \mathbf{P}(\Sigma^{\alpha_i}|\alpha_i, \Pi^{\alpha_i})$.

However, this method fails to account for the "long-term effects" of action failures on actions later in the execution path, which may or may not result in script failures. Script simulations, on the other hand, can uncover such long-term effects and thus result in much more refined estimates of success and failure distributions.

For example, consider the following segment of an action script which makes a robot pick up the caddy in one location and then bringing it to another location where it needs to add items into the caddy.

```
...
pre: empty(gripper),reachable(caddy)
pick-up(caddy)
post: holding(caddy),up(arm)

pre: not(at(assembly-location)) or
     not(reachable(desk))
move-to(assembly-location)
post: at(assembly-location),
     reachable(desk)
pre: holding(caddy),reachable(desk)
put-down(caddy,desk)
post: on(caddy,desk),down(arm)
...
```

Now suppose the caddy slipped when the robot grasped it, so that the success condition of $holding(caddy)$ for "grasp(caddy)" is false. However, since $holding(caddy)$ is not needed for the subsequent "move-to(assembly-location)" action, script execution can continue. Yet, when the subsequent action "put-down(caddy,desk)" is attempted, the pre-condition $holding(caddy)$ is not met and execution fails at that point. The returned failure state will thus contain $\neg holding(caddy) \wedge at(assembly\text{-}location)$ and this state can be added to the $\mathbf{P}(\Phi^\alpha|\alpha, \Pi^\alpha)$. Note that, conversely, if $up(arm)$ does not hold, script execution can continue as $up(arm)$ is not needed as a pre-condition for any of the subsequent actions and that the overall script might complete successfully, thus contributing to the $\mathbf{P}(\Sigma^\alpha|\alpha, \Pi^\alpha)$ instead of $\mathbf{P}(\Phi^\alpha|\alpha, \Pi^\alpha)$.[3]

E. How to Use the Framework

Fig. 1 provides an overview on how to use the proposed framework. During the **Setup Phase**, the user defines and implements primitive actions and action scripts for the robot including pre-, post-, and operating-conditions. Then the robot can be instructed to perform the actions to learn the associated performances models, including the success/failure distribution and the time-to-completion distribution. While this training step is not strictly necessary because the system updates performance models every time an action is executed (cp. to Section III-C), thus getting more and more accurate estimates over time, it is better to start with pre-trained models to be able to get reasonable estimates right away (not surprisingly, with untrained models, initial estimates can deviate significantly from actual performance). After training, the models can be used for generating performance estimates in the **Assessment Phase**. For example, as shown in Fig. 1, the user might ask the robot about the probability of successfully fetching a screw. The robot will then simulate the fetch action down to all involved primitive actions, sample their success distribution and the time-to-completion

[3]Of course, if $\neg arm(up)$ would cause problems with navigation, then the *actual* performance of the script would still be impacted, but the robot could not know that because it has to rely on what is being explicitly modeled in its action script. In this case, the operating condition $arm(up)$ would have to be added to the script.

Authorized licensed use limited to: TUFTS UNIV. Downloaded on February 07,2023 at 21:12:05 UTC from IEEE Xplore. Restrictions apply.

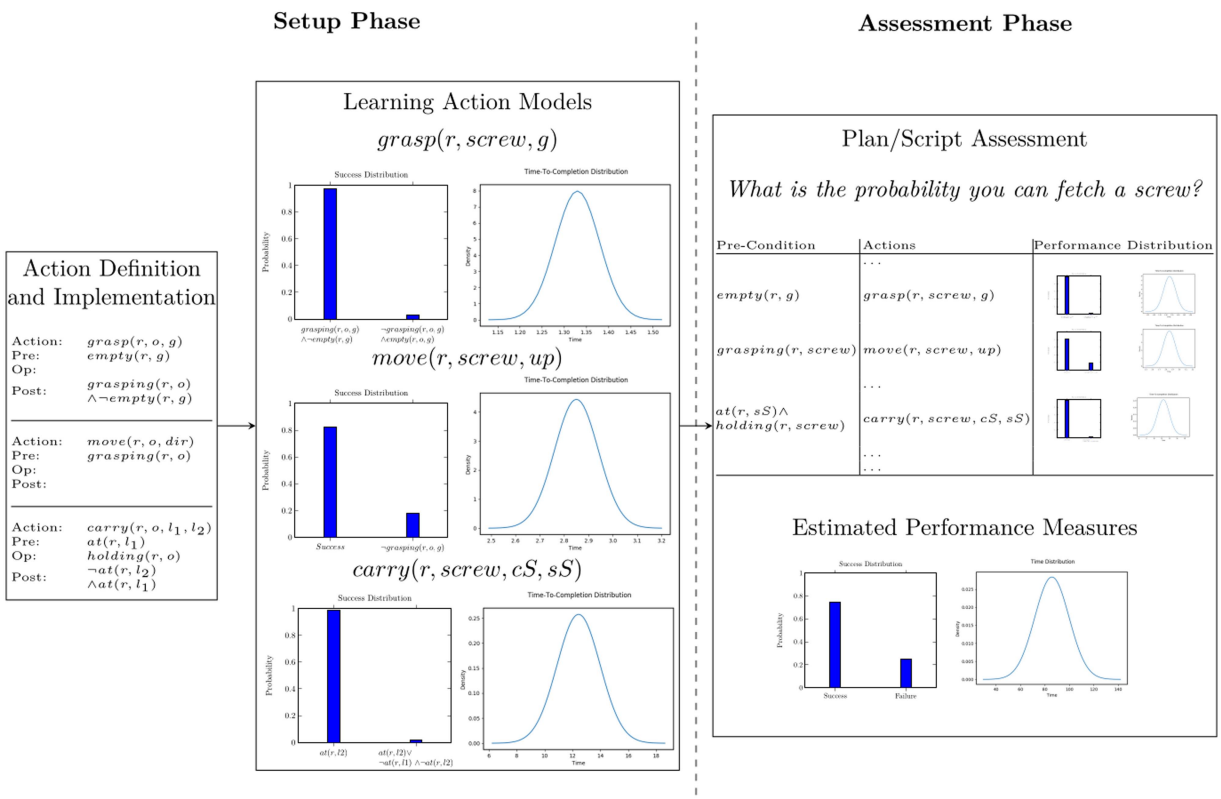

Fig. 1. A overview of how to use the proposed framework for assessing the robot performance (see text for details).

distribution until it has generated a performance estimate for the overall action measures as shown on bottom right.

IV. Evaluation

We evaluate the framework by integrating it into a robotic architecture and demonstrating the robot provides accurate performance information, when queried through natural language, about two different real world tasks, explained below. Note, we are using natural language instructions to demonstrate the framework in a HRI context, but the details for the natural language are not in the purview of this paper. There are several architectural requirements for the performance assessment algorithm, including a goal managing component. This component accepts goals from other components, maintains current and past goals, reasons about plan to accomplish the goal, and executes the goal plan. Additionally, the system needs to maintain an agent's beliefs throughout execution, including internal and external environmental states. It is critical the architecture can independently maintain beliefs for each simulation so there is no interference. For these reasons, we implement the assessment algorithm in the DIARC cognitive robotic architecture [15] which has a sophisticated goal managing system.

A. FetchIt! Mobile Manipulation Challenge

We use the well-defined FetchIt! Mobile Manipulation Challenge at ICRA 2019[4] as the testbed for our framework. The competition has a diverse set of tasks focused on manipulation and perception of small, varying shaped objects and navigation. The core task is for the robot to assemble a caddy with five objects (2 screws, 1 small gear, 1 large gear, 1 gearbox top, 1 gearbox bottom) located at stations around an arena and place the caddy on the inspection station. The small 3 m x 3 m arena, as seen in Fig. 2, has six tables around the perimeter effectively making an operating area of 2 m x 2 m which makes it difficult for the Fetch robot to navigate; since it has a base diameter of. 559 m. To assemble the caddy, the robot must navigate to each object station, pick up an object, carry it back to the caddy station, and place the object in the correct caddy compartment. Once the robot has placed the required objects in the caddy, the robot must carry the caddy to the designated inspection station.

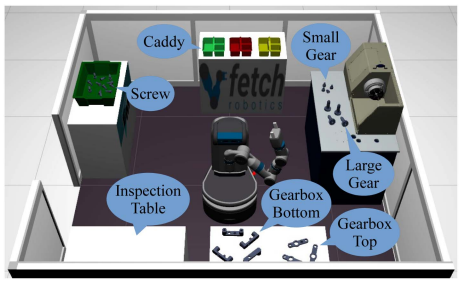

Fig. 2. FetchIt mobile manipulation challenge arena layout.

B. Assessing Performance and Results

We assume a static environment and the robot has predefined domain knowledge including:

- map of the arena including object station locations

[4]https://www.icra2019.org/competitions/fetch-it-mobile-manipulation-challenge

Authorized licensed use limited to: TUFTS UNIV. Downloaded on February 07,2023 at 21:12:05 UTC from IEEE Xplore. Restrictions apply.

TABLE I
PRELIMINARY ACTIONS AND ASSOCIATED SUCCESS POST-CONDITIONS THE ROBOT KNOWS HOW TO PERFORM. {AGENT, OBJECT, LOCATION, CADDY} REPRESENT ACTION PARAMETERS. FOR EXAMPLE, 'OBJECT' MAY CORRESPOND TO A SCREW

Actions	**Post-Conditions**
find(agent, object)	see(agent,object)
grasp(agent, object)	grasping(agent, object)
pickup(agent, object)	holding(agent, object)
release(agent, object)	¬grasping(agent,object),¬holding(agent, object)
goTo(agent, location1, location2)	at(agent, location1),¬at(agent,location2)
place(agent, object, caddy)	in(object,caddy)
deliver(agent, caddy)	on(caddy, table)
carry(agent, object, location)	
openGripper(agent)	
closeGripper(agent)	
fetch(agent, object, caddy)	
assemble(agent, caddy)	

TABLE II
DETAILED VIEW OF THE 'FETCH' AND 'ASSEMBLE' ACTION SCRIPTS INCLUDING THE PRE-CONDITIONS, POST-CONDITIONS, AND STEPS

Name:	**Fetch**	**Assemble**
Parameters:	agent object caddy	agent caddy
Pre-Conditions:	at(agent, caddy)	at(agent,caddy)
Post-Conditions:		
Steps:	goTo(agent, object, caddy) pickUp(agent, object) carry(agent, object, caddy) placeIn(agent, object, caddy)	fetch(agent, screw, caddy) fetch(agent, screw, caddy) fetch(agent, smallgear, caddy) fetch(agent, largegear, caddy) fetch(agent, gearboxtop, caddy) fetch(agent, gearboxbottom, caddy) deliver(agent, caddy)

- obstacle free arena
- adequate lighting conditions
- object shape, size, color

The robot has a set of primitive perception, manipulation, and navigation actions and action scripts, presented in Table I. Table II shows the 'fetch' and 'assemble' actions in detail.[5] We use off the shelf ROS packages MoveIt! and AMCL for manipulation and navigation motion planning and execution.

Although the robot knows how to execute the actions required to assemble a caddy a priori, the robot learns the action transition distributions online by attempting to fetch each object 25 times and then assemble a caddy 25 times. The robot attempts to fetch each object 25 times, because it may fail while assembling a caddy and will have less experience fetching some of the objects. For each attempt, the robot starts in front of and oriented toward the caddy station with its arm raised as seen in Fig. 3. From the starting pose, the robot executes the task and updates its experiences as it finishes each action.

When the robot executes the fetch task and completes it successfully, the robot will be back to where it started, ready to execute the next task. However, if the robot fails to complete the task, then we instructed it to reset its arm and navigate back to the caddy station. After the robot attempts to assemble a caddy, we command it to navigate back to the caddy and reset its arm, because the task finishes with the robot delivering the caddy to the inspection station.

With the learned transition distributions we query the performance measures, through natural language, for both the fetch and assemble tasks using the following schema:

[5]An example of the tasks and the performance assessment interactions can be viewed at https://youtu.be/TOccnYz8r-s

Fig. 3. Fetch starts in front of the caddy station with its arm raised.

TABLE III
EXPECTED AND ACTUAL TASK PERFORMANCE MEASURES FOR $fetch(robot, Largegear, caddy)$

	Success Probability	Duration (s)	Highest Failure Location
Expected	0.6	83.176±5.399	pickup(robot, largegear)
Actual	0.6	82.676±4.553	pickup(robot, largegear)

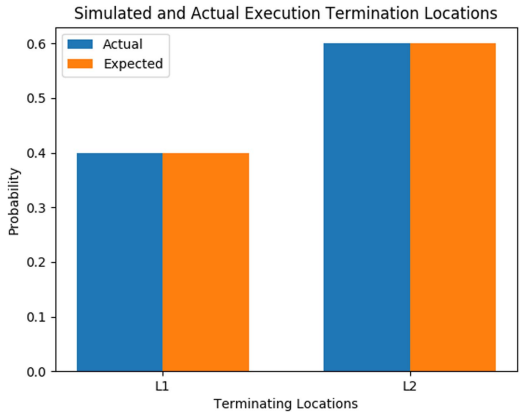

Fig. 4. Locations in the fetch task which resulted in termination of execution during simulation and actual execution; where L1 denotes a failure while picking up the large gear, and L2 denotes successful execution.

- What are your expected performance measures for task t?

where t is the task the robot should assess. We evaluate both actions to demonstrate how the algorithm accurately assesses its own performance for two tasks of different complexities.

C. Fetch Task Assessment

We first instruct the robot to assess its expected performance before executing the task to fetch a large gear:

- What are your expected performance measures for task fetch a large gear?

Upon receiving the instruction, the robot simulated the task 15 times which generated the expected success probability, time-to-completion, and most likely failure location shown in Table III. Then, we commanded it to fetch a large gear 15 times to get the actual performance measures. Fig. 4 shows the resulting states of the simulated and actual executions. The assessment framework accurately predicted the success probability and that the agent was most likely to fail when trying to pick up the large gear.

Additionally, we calculated the Kullback-Leibler divergence $D_{KL}(P||Q)$ to measure the statistical distance, or information

Authorized licensed use limited to: TUFTS UNIV. Downloaded on February 07,2023 at 21:12:05 UTC from IEEE Xplore. Restrictions apply.

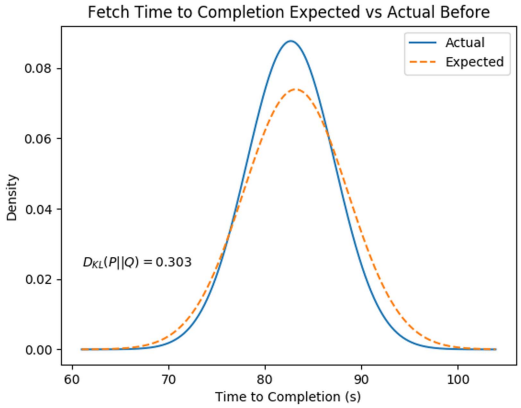

Fig. 5. The 15 simulated executions have a similar time-to-completion distribution to that of 15 actual executions of the fetch task. The two distributions have a Kullback-Leibler divergence value of 0.303.

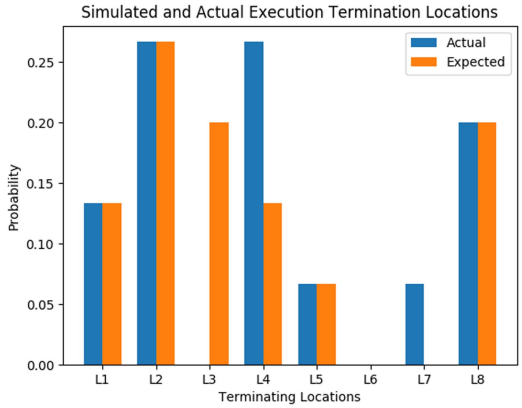

Fig. 6. Locations in the assemble task which resulted in termination of execution during simulation and actual execution; where L1, L2, L3, L4, L5, L6, and L7 denote failures when executing the following tasks respectively: 'fetch(screw),' 'fetch(screw),' 'fetch(smallgear),' 'fetch(largegear), 'fetch(gearboxtop),' 'fetch(gearboxbottom),' 'deliver(caddy)'. L8 denotes successful execution.

TABLE IV
EXPECTED AND ACTUAL TASK PERFORMANCE MEASURES FOR $assemble(robot, caddy)$

	Success Probability	Duration (s)	Highest Failure Location
Expected	0.2	583.052±17.224	$fetch(robot, screw, caddy)$
Actual	0.2	598.886±21.901	$fetch(robot, screw, caddy)$

divergence, between the expected time-to-completion distribution Q and the actual time-to-completion distribution P, shown in Fig. 5. The $D_{KL}(P||Q)$ between the actual and expected time-to-completion for the full fetch task is 0.305. The KL divergence for continuous distributions range from 0 to ∞ where values closer to 0 means that Q better models P.

D. Assemble Task Assessment

In addition, we evaluated the performance assessment framework on the more complex 'assemble' task. We instructed the robot to assess its expected performance:

- What are your expected performance measures for task assemble a caddy?

As with the fetch assessment, the robot simulated the assemble task 15 times to generate the expected success probability, duration, and most likely failure location shown in Table IV. Then, we commanded the robot to assemble a caddy 15 times to get the actual performance measures. Fig. 6 shows the resulting states of the simulated and actual executions. For the most part, the simulations failed in similar ways. However, during the evaluation, the robot unexpectedly failed to deliver the caddy to the inspection table, because the motion planner failed to calculate a trajectory for the arm to grasp and thus pick up the caddy. More training data should provide the assessment framework with more failure cases and thus be more accurate.

We obtained a statistical distance D_{KL} of 4.904 between the actual and expected time-to-completion for the 'assemble' task, as shown in Fig. 7. This higher distance makes sense, because the robot executed several more actions.

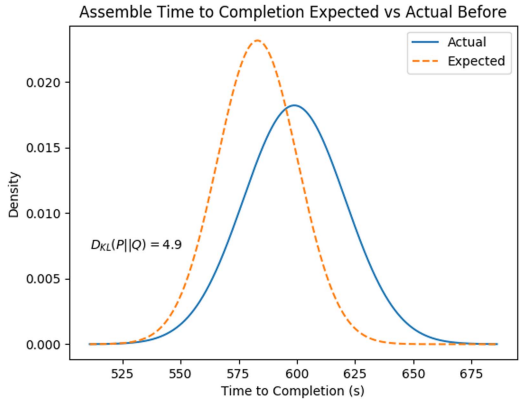

Fig. 7. The 15 simulated executions have a similar time-to-completion distribution to that of 15 actual executions of the assemble task. The two distributions have a Kullback-Leibler divergence value of 4.9.

V. DISCUSSION

The evaluation in the Fetch task showed that the proposed methods for performance assessment provides useful estimates of success and failure of action sequences or plans, as well as reasonable estimates of the overall duration. It is worth pointing out that the proposed methods are general and can be integrated into any robotic architecture that provides representations of executable actions in terms of pre-conditions (required for action execution) and post-conditions that capture action success and action failures. Given such representations, it is straightforward to add bookkeeping mechanisms that every time an action is executed track the conditional probabilities of post-conditions for the given pre-conditions. Note that the proposed methods can also be utilized by architectures that represent action sequences in terms of policies that are learned (e.g., through reinforcement learning) rather than action scripts or plans as long as the architecture also learns an explicit (approximation) τ of the MDP's $\mathcal{M}$ state transition model $\mathcal{T}_{\mathcal{M}}$. But note that because $\mathcal{T}_{\mathcal{M}}$ is defined for (complete) states in $S_{\mathcal{M}}$, it is possible (and likely) that the robot will not have encountered or gathered enough statistics of some states S^* (i.e., τ will likely be incomplete, especially with larger state spaces). In that case, performance estimates of executing actions in S^* will be off, both in terms of success/failure rates as well as duration estimates. The more fine-grained representations of pre- and post-conditions in action scripts utilized here allows robots to still adequately estimate the performance of actions in novel states S^* as long as the actions'

Authorized licensed use limited to: TUFTS UNIV. Downloaded on February 07,2023 at 21:12:05 UTC from IEEE Xplore. Restrictions apply.

preconditions are contained in S^*. This works because considering only the subset of propositions in S^* relevant for action execution as the pre-condition restricts the learned conditional probabilities to those aspects of the world affected by the action (i.e., it effectively induces an equivalence class of all world states that behave the same with respect to the execution of the action). For example, in the Fetch task our proposed algorithms will still be able to assess the robot's performance of fetching a large gear in an altered task environment that is missing all small gears. In contrast, a transition model based on world states would have to be trained on this (or any altered) environment in order to represent conditional probabilities for action success and failure as well as action durations.

VI. Conclusion

Robot self-assessment has the potential to drastically improve robot performance as well as human-robot interaction, because a robot can foresee issues and know when it is deviating from a plan. Instead of blindly executing a plan that will fail, a self-assessing robot can take precautions, stay vigilant on issues, and update humans throughout execution.

Our approach to self-assessment provides a robot with the ability to quickly estimate its expected probability of success, time-to-completion, and failure locations for a task. The robot then has the opportunity to interact with a human, modify its plan, or gain additional knowledge to improve task performance. The evaluation showed that the robot effectively predicted its performance prior to executing two tasks of varying complexity.

Though the robot could simulate its actions in advance to learn probability distributions, this would require the robot to update the distributions whenever it gained new experience. Upon completing an action, the robot would need to locate the instances of the action within other actions and then re-assess those actions. By dynamically simulating the actions when needed, it reduces the knowledge the robot needs to maintain. Moreover, the robot may generate a plan it has no prior experience for and will need to simulate the plan in order to assess its expected performance.

Not only does self-assessment have implications on task performance, but it can also play a critical role in human-robot interaction, because humans won't be able to fully comprehend the robot's abilities or limitations. We integrated the framework into a cognitive architecture and demonstrated how a human operator could instruct the robot to assess its expected performance. Again, note that the natural language details are not in the scope of this paper. Next steps will focus on how a robot can have dialogues with humans about task performance and provide explanations.

References

[1] P. Zhang, J. Wang, A. Farhadi, M. Hebert, and D. Parikh, "Predicting failures of vision systems," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2014, pp. 3566–3573.

[2] S. Daftry, S. Zeng, J. A. Bagnell, and M. Hebert, "Introspective perception: Learning to predict failures in vision systems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2016, pp. 1743–1750.

[3] H. Wu et al., "Learning robot introspection dynamics for error learning and prevention," 2017. [Online]. Available: http://130.243.105.49/Agora/IROS2017_Introspection/papers/IMRA-2017_paper_5.pdf

[4] C. B. Kuhn, M. Hofbauer, G. Petrovic, and E. Steinbach, "Introspective black box failure prediction for autonomous driving," in *Proc. IEEE Intell. Veh. Symp.*, 2020, pp. 1907–1913.

[5] Q. Yang, H. Chen, Z. Chen, and J. Su, "Introspective false negative prediction for black-box object detectors in autonomous driving," *Sensors*, vol. 21, no. 8, 2021, Art. no. 2819.

[6] A. Jauffret, C. Grand, N. Cuperlier, P. Gaussier, and P. Tarroux, "How can a robot evaluate its own behavior? A neural model for self-assessment," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, 2013, pp. 1–8.

[7] R. S. Andersen, O. Madsen, T. B. Moeslund, and H. B. Amor, "Projecting robot intentions into human environments," in *Proc. IEEE 25th Int. Symp. Robot Hum. Interactive Commun.*, 2016, pp. 294–301.

[8] E. A. Krause, P. Schermerhorn, and M. Scheutz, "Crossing boundaries: Multi-level introspection in a complex robotic architecture for automatic performance improvements," in *Proc. 26th AAAI Conf. Artif. Intell.*, 2012, pp. 214–220.

[9] M. Zillich, J. Prankl, T. Mörwald, and M. Vincze, "Knowing your limits-self-evaluation and prediction in object recognition," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2011, pp. 813–820.

[10] K. N. Kaipa, A. S. Kankanhalli-Nagendra, and S. K. Gupta, "Toward estimating task execution confidence for robotic bin-picking applications," in *Proc. AAAI Fall Symp. Ser.*, 2015, pp. 4–9.

[11] A. Jauffret, N. Cuperlier, P. Gaussier, and P. Tarroux, "From self-assessment to frustration, a small step toward autonomy in robotic navigation," *Front. Neurorobot.*, vol. 7, 2013, Art. no. 16.

[12] A. Gautam, J. W. Crandall, and M. A. Goodrich, "Self-assessment of proficiency of intelligent systems: Challenges and opportunities," in *Proc. Int. Conf. Appl. Hum. Factors Ergonom.*, 2020, pp. 108–113.

[13] T. Frasca, E. Krause, R. Thielstrom, and M. Scheutz, ""Can you do this?" Self-assessment dialogues with autonomous robots before, during, and after a mission," 2020, *arXiv:2005.01544*.

[14] D. Harel, *First-Order Dynamic Logic*. Berlin, Germany: Springer,1979.

[15] M. Scheutz, T. Williams, E. Krause, B. Oosterveld, V. Sarathy, and T. Frasca, "An overview of the distributed integrated cognition affect and reflection DIARC architecture," in *Cogn. Architectures*. Berlin, Germany: Springer, 2019, pp. 165–193.

Authorized licensed use limited to: TUFTS UNIV. Downloaded on February 07,2023 at 21:12:05 UTC from IEEE Xplore. Restrictions apply.