



Robot Proficiency Self-Assessment Using Assumption-Alignment Tracking

Xuan Cao , Alvika Gautam, Tim Whiting , Skyler Smith, Michael A. Goodrich, *Senior Member, IEEE*, and Jacob W. Crandall 

Abstract—While the design of autonomous robots often emphasizes developing proficient robots, another important attribute of autonomous robot systems is their ability to evaluate their own proficiency. A robot should be able to assess how well it can perform a task before, during, and after it has attempted the task. How can autonomous robots be designed to self-assess their behavior? This article presents the *assumption-alignment tracking* (AAT) method for designing autonomous robots that can effectively evaluate their own performance. In AAT, the robot a) tracks the veracity of assumptions made by the robot’s decision-making algorithms to measure how well these algorithms fit, or *align with*, its environment and hardware systems, and b) uses the measurement of alignment to assess the robot’s ability to succeed at a given task based on its past experiences. The efficacy of AAT is illustrated through three case studies: a simulated robot navigating in a maze-based (discrete time) Markov chain environment, a simulated robot navigating in a continuous environment, and a real-world robot arranging blocks of different shapes and colors in a specific order on a table. Results show that AAT is able to accurately predict robot performance and, hence, determine robot proficiency in real time.

Index Terms—Assumption-alignment tracking (AAT), autonomous robot system, robot proficiency self-assessment.

I. INTRODUCTION

THE design of autonomous robot systems has understandably emphasized the development of *proficient* robots—i.e., robots that can effectively carry out tasks in varying environments. Developing proficient robots is an ultimate goal, but an autonomous robot should also have the ability to identify and predict when it can and cannot successfully carry out a task.

Proficiency self-assessment is valuable for several reasons. First, safety dictates that autonomous systems should identify when they will fail, are failing, and have failed to accomplish a task [1], [2] and also be able to explain those failures [3], [4],

Manuscript received 4 November 2022; accepted 27 February 2023. This work was supported by the U.S. Office of Naval Research (ONR) under Grant #N00014-18-1-2503. This paper was recommended for publication by Associate Editor F. Amigoni and Editor W. Burgard upon evaluation of the reviewers’ comments. (*Corresponding author: Xuan Cao.*)

Xuan Cao, Tim Whiting, Skyler Smith, Michael A. Goodrich, and Jacob W. Crandall are with the Computer Science Department, Brigham Young University, Provo, UT 84602 USA (e-mail: caoxuan8872@gmail.com; tim@whittings.org; skyler.smith897@gmail.com; mike@cs.byu.edu; crandall@cs.byu.edu).

Alvika Gautam is with the Department of Mechanical Engineering, Texas A&M University, College Station, TX 77840 USA (e-mail: alvikag@tamu.edu).

Color versions of one or more figures in this article are available at <https://doi.org/10.1109/TRO.2023.3262187>.

Digital Object Identifier 10.1109/TRO.2023.3262187

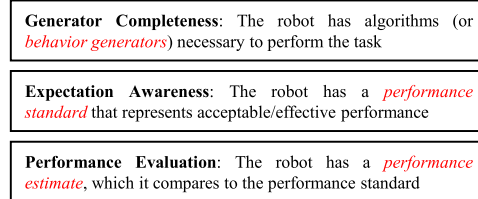


Fig. 1. Three components necessary for an autonomous robot to self-assess its ability to perform a task.

[5], [6]. Second, because autonomous robots typically operate in the context of a team, knowing one’s competence and limits can improve teaming and synergistic planning [7], [8]. Third, knowing one’s limits can be used to improve robot behavior by, for example, deciding when to continue learning and when to exploit one’s current knowledge [9], [10]. As such, we argue that autonomous robot design should simultaneously focus on both proficiency and self-evaluation of proficiency.

A few papers have directly addressed proficiency self-assessment [11], [12], [13] but each has limitations. The method proposed in [11] relies on prior knowledge that is not always available, while the methods used in [12], [13] are limited to specific platforms or environments. *Robotic introspection* is a research area similar to proficiency self-assessment but focuses more on identifying atypical operation modes [14] or input data [15], [16], [17], [18], [19]. Another related direction is *machine self-confidence*, which is a robot’s “self-trust in its functional abilities to accomplish assigned tasks” [20]. One general approach to machine self-confidence is *factorized machine self-confidence* (*FaM-Sec*), which characterizes the overall confidence of an autonomous system based on a set of factors that score different parts of the system’s decision-making process [20], [21], [22], [23], [24]. There is not yet a systematic method for identifying and accurately assessing all factors and then properly integrating them to assess the overall confidence of the robot system.

We propose that task-oriented proficiency self-assessment requires at least three evaluations (see Fig. 1). First, the robot must determine whether it has a set of decision-making algorithms (or *behavior generators*) that can be combined to perform the desired task [11]. Second, the robot must have a *performance standard*, which gives it an understanding of what constitutes desirable (or acceptable) performance on that task. Finally, the robot must estimate its performance and compare this estimate to the performance standard.

While each component in Fig. 1 is necessary for performing proficiency self-assessment, this article focuses on the third component: How can a robot effectively estimate its performance at any time during task execution? We then compare this performance estimate with the performance standard (which we assume is given) to assess whether the robot can perform the task. In particular, the approach we propose and analyze for estimating robot performance is based on the following view of proficiency self-assessment: *Proficiency self-assessment is awareness of how well one's "generators" (i.e., decision-making algorithms) interact and align with the environment(s), robot hardware, and task(s) under consideration.* Based on this view, we propose the *assumption-alignment tracking* (AAT) method that allows a robot to a) measure how well its generators align with its environment, hardware and tasks by tracking the veracity of the generators' assumptions, and b) estimate the robot's performance by adjusting its expected performance under normal circumstances according to the measurement of alignment (see Section III). We further propose a data-driven approach to implementing the AAT framework (see Section IV).

We evaluate AAT through three case studies: a simulated robot navigating in a maze-based (discrete time) Markov chain environment, a simulated robot navigating in a continuous environment, and a real-world robot arranging blocks of different shapes and colors in a specific order on a table (see Section V). Empirical results in a) scenarios where configurations remain the same throughout a task and b) scenarios where configurations change in the middle of a task demonstrate that AAT can yield interesting and accurate proficiency estimates (see Section VI). While AAT is shown to be useful for proficiency self-assessment, implementing AAT takes extra effort that could have been put into creating more proficient robot generators. Moreover, reducing the design effort to AAT could have complex effects on AAT efficacy. We discuss tradeoffs in how to allocate design effort in Section VII. Finally, we discuss key interesting observations from the experiments in Section VIII and present limitations of this work in Section IX.

This article evolved from the authors' previous work on the topic [25]. This version of this article extends the original work in the three ways: First, this article describes the methodology more systematically and provides a more generalized framework that can be applied to any discrete time state-transition system with an additive performance metric. Included in this formulation is a further description and analysis of different types of veracity checkers. Second, this article provides empirical results from two additional case studies that have different task domains and environments, including a real-world robot system. Finally, we evaluate the proposed method not only in scenarios with consistent configurations but also in scenarios in which the environment or robot hardware is altered during the mission.

II. RELATED WORK

This section reviews related literature and states how AAT relates to and differs from prior work.

A. Veracity and Reliability Assessment

The veracity assessment in AAT shares similarities with prior work. For example, Ramesh et al. [7] advocate for tracking

so-called *robot vitals*, which appear to be assessments of assumptions made about generator outputs. Similarly, Das et al. [4] used assessments of generator outputs to produce explanations to assist users in fault recovery of a robot system, and Béné and Doyen create viability tests that compare performance to predefined thresholds and use the result to estimate the resilience of an agent [26]. Finally, prior work has also discussed fault isolation to identify causes of task failure [27], wherein fault classes include "component faults" (hardware or software component failures during task) and "contextual faults" (faults related to functional system behaviors). Fault classes can be viewed as special types of alignment checkers that are restricted to post hoc assessment of failures.

Reliability [28] has been defined as the duration of time in which a robot meets performance standards under defined working conditions. Reliability metrics include mean time to failure, mean time to repair, and mean time between failures [29], [30]. Parameters needed to compute those metrics, such as failure time, failure-free time, and repair time can be estimated using fuzzy logic [31] or probability theory [32], [33]. Reliability is similar to AAT in that it evaluates actual performance relative to a performance standard, but restricts assessments of proficiency to *maintenance goal* [34]. AAT, by contrast, assesses proficiency using the probability of completing an *achievement-oriented* task or goal [34].

B. Proficiency Self-Assessment

Frasca et al. [11] introduced a general framework for robot self-assessment which allows a robot to determine what task it can accomplish and the probability it will accomplish it. Their work is that the estimated success probabilities for complex tasks rely on prior probabilities of atomic tasks. By contrast, AAT uses training data instead of explicit prior probabilities. Future work should consider how AAT could blend prior probabilities with observed data to improve assessment.

Dutta and Nelson [12] developed a learning-based method that enables a robot to measure how similar its current task is to its previous tasks and to estimate the completion time of the current task based on its previous experience. Similarly, Burghouts et al. [13] proposed a method that allows a robot to conduct self-assessment of competence by assessing whether the current environment is known followed by 1) asking a human for feedback about its competence if the environment is not known or 2) generalizing its competence from earlier experience if the environment is known. The idea of measuring similarity between tasks or assessing whether the current environment is known is similar to the veracity assessment in AAT, though the veracity assessment in AAT involves not only evaluations of environment but also evaluations of sensors, actuators, etc. Moreover, these prior methods [12], [13] only focus on a priori proficiency self-assessment, while AAT supports both a priori and in situ proficiency self-assessment.

C. Robotic Introspection

AAT is related to robotic introspection, which was first introduced by Aaron [14] as using data signatures that characterize robot operational state to differentiate between normal and

abnormal modes of operation. Grimmer et al. [15], [16] and Triebel et al. [17] adopted introspection as the capacity of a model to adjust the confidence of a particular prediction based on how representative its training data are of the corresponding test case. Daftry et al. [18] advocated for an introspective model independent from the underlying robot system. The model learns hidden latent representations from perceptual inputs with a spatio-temporal convolutional neural network and then uses a linear support vector machine that takes the latent representations as inputs and predicts how likely perceptual inputs would cause robot failure. Kuhn et al. [19] trained an introspective model to predict future disengagements of autonomous vehicles. The model uses state data describing the dynamics of vehicles as input and learns from the previous experience of the vehicle. Israelsen and Ahmed defined self-assessment as an introspective assurance for establishing user trust in an intelligent system [35]. Similar to AAT, each paper previously emphasizes the importance of identifying underlying assumptions, either explicitly or through data-driven learning methods. AAT differs from these papers in how evaluations of the assumptions are used to assess the probability of success in an achievement-oriented goal [34].

D. Machine Self-Confidence

Another notion similar to proficiency self-assessment is known as *machine self-confidence*, which is a robot's "self-trust in its functional abilities to accomplish assigned tasks" [20]. Several metrics have been proposed to measure machine self-confidence. Hutchins et al. [36] represented the self-confidence of an unmanned vehicle by measuring the uncertainties in its sensors and planners. Kuter and Miller [37] computed self-confidence by measuring the ability of an autonomous system to foresee contingencies that could threaten its performance, and to adapt its plan to circumvent those conditions. Sweet et al. [38] proposed three metrics for self-confidence in autonomous systems: success probability, plan robustness, and quality of meta-knowledge. Zagorecki et al. [39] considered autonomous systems that consist of multiple Bayesian network models. They proposed to use the *surprise index* [40] to determine which model better matches a case to be solved and to measure the confidence of an autonomous system. Kaipa et al. [41] considered the bin-picking problem where a robot is assigned to singulate multiple parts from a bin in a specific order. Each of these papers uses measures of sensor uncertainties, predicted plan disruptions, and data-driven diagnostics to assess self-confidence.

An integrated system of assessing self-confidence, known as *FaMSec*, characterizes the overall confidence of an autonomous system based on a set of factors that score different parts of the system's decision-making process [20], [21], [22], [23], [24]. In that work, five factors were proposed: command interpretation, model validity, solver quality, outcome assessment, and past performance [21], [22]. Methods for assessing the outcome assessment factor [21] and solver quality factor [23], [24] were developed. FaMSec was extended to assess and express uncertainties through generalized outcome assessments formulated in terms of task-relevant outcome semantics for UAV ISR applications [42]. The elements in FaMSec are similar to the process of identifying assumptions, defining alignment checkers, designing

progress checkers, and *training* these checkers using controlled experiments. Future work should evaluate how elements of FaM-Sec and AAT can be integrated. The self-confidence metrics in the prior two paragraphs are similar to the *progress checkers* used in this article. Indeed, self-confidence metrics could be used as progress checkers in future work. By contrast to self-confidence metrics, AAT includes direct assessments of how well explicitly identified assumptions are satisfied.

E. Metrics for Robot Proficiency Self-Assessment

Various metrics for robot proficiency self-assessment have been summarized in [43]. Some of those metrics are related to this work, including alignment of uncertainty and performance, mission progress, predicted versus actual completion time, and reliability. For each metric, we review its definition in [43], its related literature, and how it relates to this work.

Alignment of uncertainty and performance can be measured as the correlation between the variance and performance of the model's output, which was evaluated by Fitzgerald et al. [44]. Similarly, Fleming and Daw [45] correlated the model's uncertainty to the actual error incurred by the model's output. In this work, we evaluate AAT performance by computing the average predicted success/failure probability for success/failure trials [see Fig. 10(a), (d), and (g)].

Mission progress refers to the extent to which a task has been completed and reflects a robot's performance and proficiency. For sequential tasks, mission progress can be measured as the number or percentage of subtasks completed. In [46], [47], the estimation of mission progress involves reaching mileposts, satisfying preconditions or postconditions, deviations from scripts, and timing metrics. The *progress checkers* in the present article are application-specific implementations of mission progress. Future work should explore how additional progress checkers can be developed and applied.

Assessment metrics can also include the distribution of predictions and observations to recognize when a system is not functioning well enough. For example, an approach to measure the proficiency of a model using a Cramer Rao-like measure of the expected value of the second moment of a log likelihood function was studied in [48].

Predicted versus actual completion time indicates the accuracy in predicting the robot's task completion time. Similarly, Schneider et al. [49] measured the difference between the predicted and actual time for a failure. The difference between actual and predicted completion time is post hoc but is used in this article as the training target of the kNN algorithm in order to facilitate in situ assessments.

These metrics and methods can be classified as progress checkers (using the terminology in this article). AAT uses both alignment checkers and progress checkers, and future work should evaluate how these more sophisticated progress checkers could be combined with alignment checkers to better perform proficiency self-assessment. Of particular interest is exploring how general patterns of progress checkers can be distilled from these cited prior works.

III. METHODOLOGY AND FRAMEWORK

Estimating performance in arbitrary environments and scenarios is challenging because it is difficult to determine how

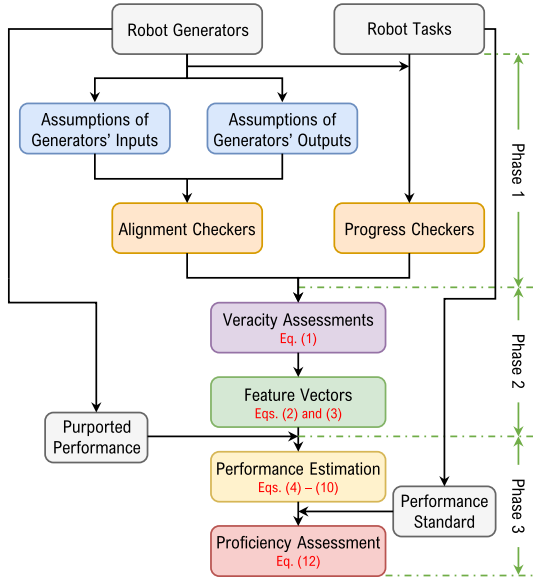


Fig. 2. Schematic of AAT for proficiency assessment.

environmental characteristics, the robot’s hardware, and the robot’s decision-making algorithms (or generators) will combine to impact the robot’s performance. This is difficult for post hoc estimates, but even more difficult for a priori and in situ estimates. Past work has conjectured that a robot’s performance is a function of the complexity of the environment in which the autonomous system operates (e.g., [50], [51]). Unfortunately, quantifying the effect of environment complexity on a robot’s task performance remains unsolved.

We assert that robot performance is sensitive to how well its generators align with its environment and hardware. This alignment can be determined via a set of metrics that do not require a direct assessment of the complexity of the environment, but rather is made by *tracking the veracity of the assumptions* upon which the robot’s generators rely. We call this AAT.

Fig. 2 shows the three phases of applying AAT to assessing robot proficiency. In phase 1, the robot designer identifies assumptions made in the construction of the robot’s generators, and creates a) *alignment checkers* that assess the veracity of these assumptions and b) *progress checkers* that explicitly estimate the robot’s progress in completing its tasks. In phase 2, the outcomes of these checkers over time form a time-indexed set of veracity assessments of the identified assumptions, which are then converted to feature vectors. In phase 3, feature vectors are used to model how to adjust the robot’s *purported performance* (expected performance under normal circumstances) to estimate the robot’s actual performance, which is compared to a performance standard to determine whether the robot is proficient or not. The rest of this section and the next section formalize the process.

A. Trajectory, Task, and Performance

Let $E = S \times A \times S$ be a discrete time state transition system that models the environment as a mathematical relation where S

denotes a suitable state space and A denotes a robot’s available actions. Elements of this relation are (present state, action, next state) triples.

A robot within the environment takes observations of the environment and generates actions, as described by the robot generator relation $R = S \times O \times A$. Elements of the robot relation are (state, observation, action) triples, where each state is associated with one observation and at least one action.

When the robot and environment relations are joined, a set of time-indexed state and observation *trajectories* are created with elements, respectively

$$\mathbf{s}_t = [s_0, s_1, \dots, s_t] \text{ and } \mathbf{o}_t = [o_0, o_1, \dots, o_t]$$

where, s_0 is an initial state and o_0 is an initial observation.

A *performance metric* can be either a *utility* to be maximized or a *cost* to be minimized that maps a time-indexed state trajectory to a real value. This work considers problems in which a single task is performed by the robot and in which there is a cost function defined for the robot metric

$$C : \{\mathbf{s}_t\} \mapsto \mathbb{R}.$$

This work further assumes that C is additive, such that

$$C(\mathbf{s}_t) = C([s_0, s_1, \dots, s_\tau]) + C([s_\tau, \dots, s_t])$$

where, $0 < \tau < t$. Future work should explore other performance metric structures.

In practice, the robot estimates its state trajectory based on its observation trajectory and then applies the cost function to the estimated state trajectory to predict its final cost. For simplicity, we explicitly define C as a mapping from an observation trajectory to a real value

$$C : \{\mathbf{o}_t\} \mapsto \mathbb{R}.$$

B. Assumptions and Veracity Assessments

As asserted in many No-Free-Lunch Theorems (e.g., [52]), all decision-making algorithms (or generators) for autonomous robot systems are based on assumptions or biases that dictate their performance [53]. When these assumptions and biases are satisfied in the real world, the robot’s behavior and its impact on the world satisfy the designer’s intentions. However, when the assumptions are not met, the robot’s behavior and its impact on the world are both less predictable and less likely to be satisfactory. Thus, *the ability to differentiate between conformity to and violation of these assumptions can provide rich insight into a robot’s performance.*

Fig. 2 illustrates two important forms of assumptions related to a robot’s generators: assumptions about generator inputs and generator outputs. Assumptions about generator inputs include assumptions about the robot’s sensors, the robot’s actuators, and the properties of the environment. For example, a mapping system for a robot performing a navigation task might assume that sensor readings have low variance and that the sensor can detect the objects in the environment. Assumptions about generator outputs relate to the properties of the outcomes of generator decisions. For example, a robot navigation system might be designed under the assumption that the robot’s mapping

system will compute consistent positions of stationary objects in the environment.

Once the assumptions made in the design of the robot generators are identified, the system designer creates *assumption checkers* that check *assumption veracity*. Assumption veracity assessments can begin before the robot starts to operate in the environment (i.e., a priori; these assessments are mainly limited to input assumptions) and then continue throughout the mission (i.e., in situ assessments). The resulting time series of assessments over each of the assumptions form the *alignment profile*, which is used to evaluate the robot's ability to perform the given task in the current environment.

Suppose that there are M assumptions made by the robot's generators. In AAT, at least one checker is created for each assumption. An *alignment checker* is a generator-specific function that evaluates the robot's observations to determine how well a generator's assumption is satisfied. Alignment checkers can produce boolean or real-valued assessments. For simplicity, assume that there is exactly one alignment checker per assumption, and denote the M alignment checkers as $\{v_1(\mathbf{o}_t), v_2(\mathbf{o}_t), \dots, v_M(\mathbf{o}_t)\}$. Note that even though technically \mathbf{o}_t is the input to each checker, many of the alignment checkers implemented in this work require only the current observation o_t , while others (such as those assessing stochastic properties) use only a limited number of the latest observations. This approach to alignment checking makes the veracity assessments sensitive to sudden changes during tasks.

Because not all assumptions can be easily checked and because some assumptions can easily be missed, we create a second set of checkers known as *progress checkers*. Progress checkers are assessments that explicitly estimate the robot's progress in completing a task. They aggregate many assumptions and hence are a sort of *catch all* checker. Such checkers could be, for example, how far a robot has traveled, the robot's average speed, or how many subtasks have been accomplished [46]. A progress checker is therefore a generator- and task-specific function that returns a real-valued estimate of progress. Suppose that there are K progress checkers, $\{v_{M+1}(\mathbf{o}_t), v_2(\mathbf{o}_t), \dots, v_{M+K}(\mathbf{o}_t)\}$.

The assessments made by alignment checkers and progress checkers are combined to form the *veracity assessment at time t*

$$\mathbf{v}(\mathbf{o}_t) = [v_1(\mathbf{o}_t), v_2(\mathbf{o}_t), \dots, v_{M+K}(\mathbf{o}_t)]. \quad (1)$$

Because some veracity assessments are noisy and benefit from temporal smoothing, $\mathbf{v}(\mathbf{o}_t)$ is better represented as a *feature vector*, at time t , $\mathbf{f}(\mathbf{o}_t) = [f_1(\mathbf{o}_t), f_2(\mathbf{o}_t), \dots, f_{M+K}(\mathbf{o}_t)]$ that includes any temporal blending that must occur. There are several ways to perform this temporal blending, and this article uses exponential smoothing, yielding

$$f_k(\mathbf{o}_t) = \lambda_k f_k(\mathbf{o}_{t-1}) + (1 - \lambda_k) v_k(\mathbf{o}_t) \quad (2)$$

where, $\lambda_k \in [0, 1]$. The initialization condition, $f_k(\mathbf{o}_{-1})$, is set to zero and is used for computing $f_k(\mathbf{o}_0)$. This article uses progress checkers that explicitly integrate multiple observations together. Therefore, the feature vectors for the progress checkers are $f_k(\mathbf{o}_t) = v_k(\mathbf{o}_t)$, corresponding to $\lambda_k = 0$. By contrast, this article uses $\lambda_k > 0$ for alignment checkers, which benefit from temporal smoothing. Evaluating $\lambda_k \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ in

the Markov chain study showed little sensitivity (discriminability root mean square error less than 2.14%) so $\lambda_k = 0.3$ is used in all evaluations.

Feature vectors give rise to the *alignment profile* at time t , which is the time series of veracity assessments encoded as feature vectors up to time t

$$\mathbf{F}(\mathbf{o}_t) = [\mathbf{f}^\top(\mathbf{o}_0), \mathbf{f}^\top(\mathbf{o}_1), \dots, \mathbf{f}^\top(\mathbf{o}_t)]^\top \quad (3)$$

where, the superscript \top denotes "transpose." The alignment profile gives insight into how well the robot's generators will interact with its environment and hardware systems. It can thus be used to estimate the robot's performance.

C. Estimating Performance

The estimate of the final performance is a function of the observations up to the current time t and projections of observations in the future. Let $\hat{C}([o_0, \dots, o_t, \dots])$, where the last ellipsis denotes future observations, denote this prediction of final task performance made at time t . Because the cost function C is additive, $\hat{C}([o_0, \dots, o_t, \dots])$ is estimated from two components: estimated performance so far on the mission up to time t , denoted by $\hat{C}([o_0, \dots, o_t])$, and the performance predicted in the future, denoted by $\hat{C}([o_t, \dots])$. Thus

$$\hat{C}([o_0, \dots, o_t, \dots]) = \hat{C}([o_0, \dots, o_t]) + \hat{C}([o_t, \dots]). \quad (4)$$

The second half of (4), specifying estimated future performance, is obtained from two parts: *purported performance* $\bar{C}([o_t, \dots])$ and a *scaling coefficient* η . *Purported performance* can be thought of as a type of expected performance if things go as planned from the current state. It is computed by the planner under the assumption that all the generators' assumptions hold. The *scaling coefficient* η depends on the current veracity assessment, $\mathbf{f}(\mathbf{o}_t)$, and is therefore expressed by the function $\eta(\mathbf{f}(\mathbf{o}_t))$. Intuitively, this scaling coefficient indicates how the cost is likely to change, usually at an increase, as assumptions are violated. In addition, as demonstrated in the case studies we present later in this article, purported performance estimates computed by the planner need not be perfect since scaling can simultaneously provide corrections to those estimates. Therefore, AAT estimates the final performance as

$$\hat{C}([o_0, \dots, o_t, \dots]) = \hat{C}([o_0, \dots, o_t]) + \bar{C}([o_t, \dots]) \cdot \eta(\mathbf{f}(\mathbf{o}_t)). \quad (5)$$

The next section presents an implementation of (5).

IV. AN IMPLEMENTATION OF THE AAT FRAMEWORK

This section presents an implementation of AAT, including design choices for the cost function C and the scaling coefficient η . First, we define C as the time-to-complete a task. Then, we approximate η based on the robot's past experiences, which consist of feature vectors contained in the alignment profiles of different past runs (experiences) as well as the known scaling coefficients associated with those runs. To do this, we implement a k-nearest neighbor (kNN) algorithm that computes the past feature vectors that are closest to the current feature vector.

These nearest-neighbor samples specify a probability distribution of the current scaling coefficient. Finally, the probability distribution of the current scaling coefficient is used to compute a probability distribution predicting the robot's performance, which can then be used to make an assessment of the robot's proficiency by comparing the predicted performance to the given performance standard. In Section V, we study the effectiveness of this implementation of AAT by applying it to three different systems.

A. Implementing the Cost Function C

For many tasks, including those considered in the three case studies we present subsequently, the cost metric is the time-to-complete the task. The cost up to the current time, t , is therefore the elapsed time

$$C(\mathbf{o}_t) = |\mathbf{o}_t| = t$$

where, $|\mathbf{o}_t|$ denotes the number of steps in the trace. Given this definition, (5) becomes

$$\hat{C}([o_0, \dots, o_t, \dots]) = t + \bar{C}([o_t, \dots]) \cdot \eta(\mathbf{f}(\mathbf{o}_t)). \quad (6)$$

B. Computing the Scaling Coefficient η

We compute the scaling coefficient η using a kNN algorithm (we justify the usage of kNN in Section VIII). This algorithm compares the current veracity assessment vector with feature vectors from the alignment profiles recorded in the training set. The training set is formed from past experiences in which the robot attempted the task in potentially different environments and under different conditions, some of which violate the assumptions made in the creation of the robot's generators.

Formally, the training set is formed from a set of trials in which each trial j yields an alignment profile $\mathbf{F}^j(\mathbf{o}_t)$ that contains a set of feature vectors $\{\mathbf{f}^j(\mathbf{o}_t)\}$. Associated with each of these feature vectors is a scaling coefficient, which denotes the robot's actual performance in the trial relative to the purported performance at that time in the trial. Let T^j denote the time it took the robot to perform the task in trial j and let $\bar{C}^j([o_t, \dots])$ denote the purported performance of the robot at time t in trial j . Then, the scaling coefficient of this sample is given by

$$\eta_t^j = \frac{T^j - t}{\bar{C}([o_t, \dots])}. \quad (7)$$

Training experiments, therefore, yield a set of time-indexed feature vectors, \mathbf{f}^j , along with an estimate of the scaling coefficient η^j . The combined vectors and scaling coefficients yield a training set X for the learning algorithm.

The kNN algorithm is used to estimate the scaling coefficient. The algorithm computes the set $N \subseteq X$ as the k samples in X with the feature vectors that are nearest to the current feature vector $\mathbf{f}(\mathbf{o}_t)$. A weighted $L1$ distance (e.g., the Manhattan distance) is used as the distance metric to compute this set. Given two feature vectors, \mathbf{f}^i and \mathbf{f}^j , the weighted $L1$ distance is given by

$$\|\mathbf{D}, \mathbf{f}^i, \mathbf{f}^j\|_1 = \sum_{n=1}^{M+K} D_n \cdot |f_n^i - f_n^j| \quad (8)$$

where, $\mathbf{D} = [D_1, D_2, \dots, D_{M+K}]$ is a vector used to weight the relative strength of the dimensions of the feature vector. Each nearest neighbor $n \in N$ then provides an estimate of the scaling coefficient, η^n .

To form a probability distribution over the scaling coefficients η , each $n \in N$ is assigned a raw weight, denoted by w_{raw}^n , that is inversely proportional to the distance between the currently measured feature vector $\mathbf{f}(\mathbf{o}_t)$ and the feature vector of the neighbor, \mathbf{f}^n . From these raw weights, a normalized weight is computed as

$$w_{\text{norm}}^n = w_{\text{raw}}^n / \sum_{i=1}^k w_{\text{raw}}^i. \quad (9)$$

These normalized weights, combined with the scaling estimates η^n , form a probability distribution to define the scaling coefficient η . In turn, the scaling estimates η^n are used in (6) to produce an estimate (made by sample n) of final performance

$$\hat{C}^n([o_0, \dots, o_t, \dots]) = t + \bar{C}([o_t, \dots]) \cdot \eta^n. \quad (10)$$

Equations (9) and (10) then provide a probability distribution over the robot's final estimated performance on the task.

The experiments in the next section consider three distance functions: $\mathbf{D}_{\text{alignment}}$, $\mathbf{D}_{\text{progress}}$, and \mathbf{D}_{all} given by

$$\mathbf{D}_{\text{all}} = \mathbf{D}_{\text{alignment}} + \mathbf{D}_{\text{progress}}. \quad (11)$$

$\mathbf{D}_{\text{alignment}}$ includes nonzero weights for alignment checkers while zero weights for progress checkers. On the other hand, $\mathbf{D}_{\text{progress}}$ includes nonzero weights for progress checkers while zero weights for alignment checkers. The specific three distance functions and raw weights w_{raw}^n used in the three case studies are described in the next section.

C. Assessing Proficiency Requires a Performance Standard

The process of proficiency self-assessment described in Fig. 1 requires not only the performance estimate, but also a performance standard. This section considers the latter. Since the case studies use time-to-completion of an achievement-oriented task [34] as the performance metric, the performance standard is a predefined time bound T_{bound} , which is used to determine whether robot performance is acceptable. Define the goal function, denoted by G , that maps cost C to a boolean value

$$G(C) = \begin{cases} \text{true} & \text{if } C \leq T_{\text{bound}} \\ \text{false} & \text{otherwise} \end{cases}$$

where, $G = \text{true}$ means the robot's performance is acceptable. Note that G can be either measured for completed trials or estimated for ongoing trials. In other words, G is a random variable with binary outcomes, $\{\text{true}, \text{false}\}$.

Each nearest neighbor $n \in N$ provides a performance estimate \hat{C}^n along with a normalized weight w_{norm}^n using (9) and (10). Applying G to each \hat{C}^n tells whether each nearest neighbor predicts success or failure. Thus, summing up the normalized weights of the nearest neighbors that predict success yields the predicted success probability

$$P(G = \text{true}) = \sum_{n=1}^k w_{\text{norm}}^n, \text{ if } G(\hat{C}^n) = \text{true}. \quad (12)$$

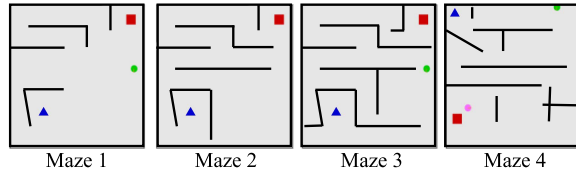


Fig. 3. Four worlds (or mazes) used in the navigation case study. The blue (triangle) robot is tasked with getting to its charger (the red square). Black line segments and other robots (green and pink circles) are obstacles in the environment.

The predicted failure probability is $1 - P(G = \text{true})$.

The *proficiency assessor* is a binary classifier that takes as input the success probability in (12), compares it to a subjective threshold θ , and indicates that the system will succeed if $P(G = \text{true}) \geq \theta$. Varying θ changes the true positive and false positive rates of the resulting classifier, yielding the receiver-operator characteristic (ROC) curves, which are useful for assessing discriminability.

V. CASE STUDIES

We applied AAT to three case studies: a navigation task, a maze-based discrete time Markov chain, and a Sawyer robot manipulating blocks on a table. The case studies were chosen because they have different task domains and environments. Consequently, the case studies provide evidence that a straightforward and simple implementation of AAT can produce reasonably effective estimates of proficiency self-assessment across many task domains.

Each case study implements study-specific elements of AAT. However, in accordance with the process of AAT described in the previous sections, each study consists of the following steps. First, the assumptions made in the design and implementation of the robot's generator(s) are identified. Second, veracity checkers for the identified assumptions are implemented. Third, experiments are performed with variations in environment and system configurations to populate a dataset, which is divided into training and test set. Fourth, case-specific parameters for the kNN algorithm are chosen and used to predict the task success probability of the robot system from the training set. The resulting predictor is then evaluated on the test set. The remainder of this section describes these elements for each case study.

A. Navigation Task

1) *Task Domain*: The robot is a simulated robot that can spin in place in either direction or move forward. The robot is equipped with two sensors: a camera that looks down on the world from above and a sensor that detects whether or not the robot is on a charging pad. The robot's task is to navigate to its charger within a certain amount of time.

In this case study, we consider the four world environments shown in Fig. 3. These simulation environments include the robot (shown as a blue triangle) and its charger (shown as a red square). All other entities shown in the figures, including black line segments and pink and green robots (shown as circles),

are obstacles through which the blue robot cannot pass. The four worlds shown in Fig. 3 are designed to represent different difficulty levels, determined by the initial distance between robot and charger as well as the number and placement of obstacles. Mazes 1–2 were selected to generate training data, while Mazes 3–4 were used to generate test data.

Given that the robot must travel different distances to reach its charger in each world, we set a different performance standard for each world. The performance standards in Mazes 1–4 are set such that the robot should reach the charger within 60, 150, 300, and 220 s, respectively.

2) *Generators and Assumptions*: The simulated robot is equipped with three different decision-making algorithms (or generators) to perform the navigation task: 1) a mapper, 2) a path planner, and 3) a controller. The *mapper* takes as input the camera image and creates a map of the environment by detecting and localizing itself and its charger in the world. It also creates an obstacle map of the world. A new map is created every time a new camera image is received. Mapper assumptions are:

- 1) the camera produces up-to-date images;
- 2) the camera is in the expected location, is oriented downward, and has the assumed view-angle;
- 3) the camera sees color according to specification;
- 4) the camera has low noise and distortion;
- 5) the robot is blue and the charger red, and no other objects in the environment are those colors;
- 6) the robot and charger are visible in the camera image;
- 7) all obstacles are also visible and are not white in color.

The assumed output of the mapper is a consistent and accurate map of the world.

The *planner* takes as input the map, created by the mapper every time a new camera image is received, to plan a path from the robot to the charger using RRT* [54]. In addition to assuming the map created by the mapper is correct, the planner assumes:

- 1) the robot is of the assumed size and shape;
- 2) the world is stationary (other than robot movement);
- 3) there is a path to the charger that can be found within 1500 iterations of RRT*;
- 4) the open space in the world has uniform cost (i.e., the robot drives as easily through one open space as another).

The assumed planner output is a planned path with a consistent path length.

The *controller* takes as input the planned path and outputs commands to the robot's actuators, which move the robot along the chosen path to the charger. In addition to assuming the path selected by the planner is feasible, the controller assumes:

- 1) the robot's actuators are engaged (and react to the controller's commands);
- 2) the robot spins at the expected speed;
- 3) the robot moves straight when going forward;
- 4) the robot moves at the expected speed.

The expected output of the controller are wheel movements that move the robot in the expected manner through the world. Together with the expected output of the other two generators, the generators are assumed to cause the robot to approach the charger at an expected rate.

3) *Alignment and Progress Checkers*: Based on the identification of these assumptions, Fig. 4 shows the alignment

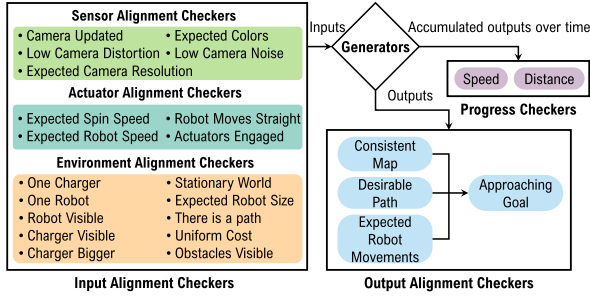


Fig. 4. List of alignment and progress checkers implemented in the navigation case study.

checkers that were implemented in this study to monitor the veracity of these assumptions. Note that the experiments did not include an alignment checker that verified that the camera was located and oriented as expected. Progress checkers were also created to monitor two parameters that correlate to the robot’s performance: 1) the average speed with which the robot has navigated so far and 2) the proportion of the distance the robot has covered towards its goal (the charger) from its initial position.

4) *Study-Specific Details:* Implementing AAT requires specifications of study-specific values and mechanisms related to (8)–(10), including purported performance $\bar{C}([o_t, \dots])$, raw weights (w_{raw}^n), and distance functions.

Purported performance. For this navigation task, purported performance is based on the amount of time we expect the robot to take to traverse the path computed by the planner under normal circumstances. The path computed (using RRT*) by the planner consists of a sequence of segments in the world. For each segment, the robot must first orient (spin action) in the direction of segment, and then traverse the segment (straight action). Given how fast we observed the robot spin and move under normal circumstances, we compute the purported performance of the robot as

$$\bar{C}([o_t, \dots]) = 4|\mathbb{P}| + \frac{1.2 \cdot \text{dist}(\mathbb{P})}{v_{\text{max}}} \quad (13)$$

where, \mathbb{P} is a set of segments that define the planned path from the robot’s current position to its charger, $|\mathbb{P}|$ is the number of segments in the path, $\text{dist}(\mathbb{P})$ is the length of the path \mathbb{P} , and v_{max} is the robot’s assumed maximum speed. The first portion of (13) (i.e., $4|\mathbb{P}|$) indicates that it takes the robot approximately four seconds on an average to orient to the direction of the next segment after it has navigated a segment. The second half of this equation specifies the average time required to traverse the segments of the path when assumptions are met. Because the robot needs to occasionally reorient (by turning in place) as it traverses a segment, it does not always travel at full speed. Thus, the coefficient of 1.2 is added (determined experimentally under normal conditions) to account for the time spent reorienting as it traverses a segment.

Raw weight. In this case study, the raw weight for each nearest neighbor, w_{raw}^n , is given by

$$w_{\text{raw}}^n = \max(0, \alpha - \|\mathbf{D}, \mathbf{f}(o_t), \mathbf{f}^n\|_1) + \beta \quad (14)$$

TABLE I
COMBINATIONS OF ASSUMPTION VIOLATIONS IN THE STUDY

Combination of Variations	Range of Parameters
Camera noise-Robot bias	Noise: 0 to 0.3; Bias: -2 to 2
Camera noise-Robot speed	Noise: 0 to 0.3; Speed: 1 to 11m/s
Camera distortion-Robot bias	Distortion: -2 to 2; Bias: -2 to 2
Camera distortion-Robot speed	Distortion: -2 to 2; Speed: 1 to 11m/s

where, $\alpha = 5$ and $\beta = 0.5$ are subjectively chosen. (14) is subjectively chosen to guarantee that the raw weight is positive and correlates to distance negatively. Pilot evaluations, not reported in this article, indicate that its parameters, α and β , can be changed without substantially affecting AAT efficacy (e.g., different values of α and β are used for the block manipulation case study). Moreover, (14) can be replaced with (15), which is a more common way to compute raw weight.

Distance functions. For \mathbf{D}_{all} , the coefficient for each alignment checkers is set to 1, while the coefficients for the progress checkers (average speed and proportion of traveled distance) are set to 2 and 3, respectively. These coefficients are not thoroughly optimized, but are subjectively set to adequately balance the impact of alignment and progress assessments in this study; the other case studies use the same weights. $\mathbf{D}_{\text{alignment}}$ and $\mathbf{D}_{\text{progress}}$ can be derived using (11).

5) *Training and Evaluation:* To evaluate the ability of the robot to perform proficiency self-assessment, the simulator is configured to allow a human to act as a *foil* to the robot. The authors, acting as the foil, varied these scenarios by modifying the robot’s camera sensor and actuators in two ways each: the noise and distortion of the camera image, and the speed and bias of the robot’s wheels (negative bias causes the robot to drift right when moving straight, while positive bias causes the robot to drift left). Table I describes how these variations were randomized and combined together throughout the scenarios. When the magnitude of variations is large, these variations produce assumption violations (i.e., the generators were not specifically designed to handle these scenarios), and hence the robot is more likely to fail in the task.

The combinations of variations were applied evenly in simulations across the four mazes (see Fig. 3), yielding 41 simulations in each world. Simulations conducted in Mazes 1–2 were used for training data, while simulations conducted in Mazes 3–4 were used to test the ability of AAT to discriminate between successful scenarios and unsuccessful scenarios. In these scenarios, variations were kept consistent for the entire time the robot performed the task.

Many real world scenarios are prone to sudden changes in the environment and system capabilities. For example, the robot may suddenly begin to fail in the middle of a mission if its sensors or actuators get damaged or its battery begins to fail. Thus, to further test the efficacy of AAT, we conducted three additional simulation runs where a sudden change was introduced about 60 s after the task started. The same AAT predictor (trained in the previously mentioned training scenarios) was used to predict the task outcome for these additional simulation runs. Table II lists the configurations for these three sudden-change scenarios for this case study.

TABLE II
CONFIGURATION DETAILS FOR THE SIMULATIONS WITH A SUDDEN CHANGE IN THE MIDDLE

Configuration before	Configuration after
Normal (all assumptions met)	Camera noise altered to 0.25
Camera noise was 0.25	Normal (all assumptions met)
Normal (all assumptions met)	Robot bias altered to -4.5

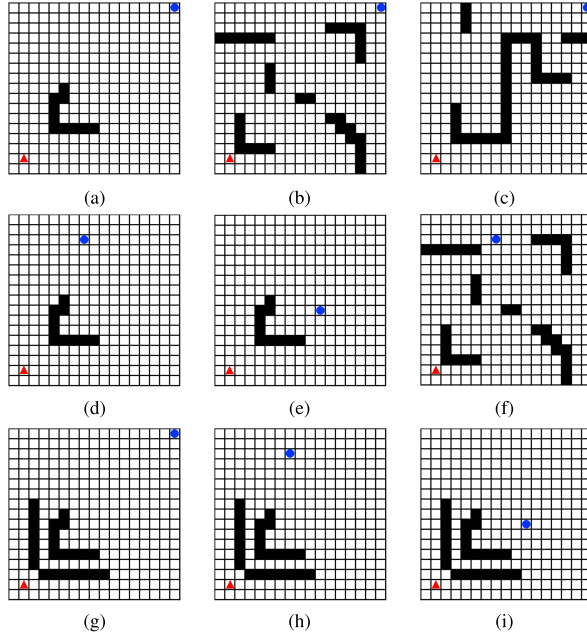


Fig. 5. State space used in the maze-based Markov chain study. White cells: valid states. Black squares: walls. Blue circles: goal states. Red triangles: starting states.

B. Maze-Based Discrete Time Markov Chain

1) *Task Domain*: A simulated robot in a maze-based discrete time Markov chain world (see Fig. 5) attempts to reach a goal state 1) within a specific number of time steps and 2) with fewer than a specific number of collisions with walls. The discrete time Markov chain components are as follows.

- 1) *State space*: In the maze world, white cells are valid states, black squares are walls, and the blue circle is the goal state. The world is surrounded by implicit walls.
- 2) *Action space*: The robot's action space is moving in the cardinal directions, $\{\text{up}, \text{down}, \text{left}, \text{right}\}$.
- 3) *Current state*: The state where the robot currently is.
- 4) *Transition*: The robot transitions from the current state to the next state indicated by its action (e.g., moving up if the action is up) with probability ρ , and transits to one of the other three adjacent states with probability $(1 - \rho)/3$. The robot remains in the same state if the transition would cause the robot to hit a wall.
- 5) *Reward*: The robot gets a state-related reward when a transition happens without a collision. If the robot hits a wall, it receives a negative reward as a punishment for the collision.

- 6) *Goal state*: a specified state in the world; e.g., the blue cell in Fig. 5.

The robot is equipped with the following sensors that assist the robot to complete its task.

- 1) A simulated GPS that returns the robot's current state.
- 2) An obstacle sensor that indicates which adjacent states are open and which are obstacles.
- 3) A collision sensor that indicates whether the robot hit a wall when it tried to move.
- 4) A direction sensor that reports the direction the robot moved.
- 5) A simulated motor sensor that reports the actual action (not the intended action) taken by the robot.
- 6) A goal sensor that indicates whether the robot is in the goal state or not.
- 7) A reward sensor that reports the actual reward or punishment that the world gives to the robot when it acts.

The sensors only return correct values with probability 0.99, and the following information is returned when they give incorrect values.

- 1) The GPS returns a location randomly chosen from the robot adjacent states.
- 2) The obstacle sensor returns the false status of the robot adjacent states; the correctness for each state is independent.
- 3) The collision sensor returns the false outcome.
- 4) The direction sensor returns a false direction randomly chosen from all potential false directions.
- 5) The motor sensor returns an action that is not in the action space.
- 6) The goal sensor returns the false outcome.
- 7) The reward sensor returns the correct reward plus or minus 1.

2) *Generator and Assumptions*: The robot does not directly know any of the components of the *actual* world described in the previous section. Instead it assumes a specific configuration of the world, which may or may not match the actual world. The robot generates its policy using the value iteration algorithm (see, for example, [55]) under the assumption that the real world's state space, action space, transition probability, and rewards match the specified configuration. The value iteration algorithm is applied to the specified configuration rather than the configuration in which the robot actually operates. When the robot is in a state not represented in its policy (due to its assumed state space being different than the real state space), it takes an action randomly chosen from its assumed action space. If the intended action is excluded in the real action space, the robot takes an action randomly chosen from the real action space.

3) *Alignment and Progress Checkers*: Since value iteration is executed given the robot's assumptions about the world, alignment and progress checkers are needed; see Fig. 6. The following *Input* alignment checkers were implemented.

- 1) *CurrentState* checker: whether the state deduced from GPS information matches with that deduced from the motion information.
- 2) *StateSpace1* checker: whether the validity of the current state matches with the assumption.

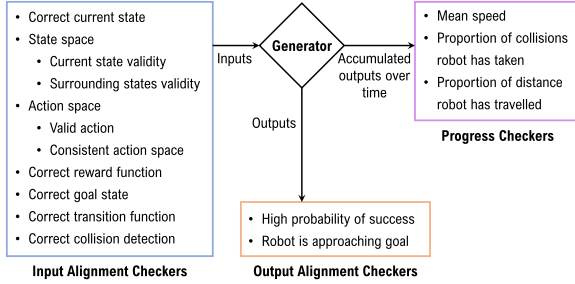


Fig. 6. Alignment and progress checkers in the maze-based discrete time Markov chain case study.

- 3) `StateSpace2` checker: whether the validity of the surrounding states matches with the assumption.
- 4) `ActionSpace1` checker: whether the intended action is valid.
- 5) `ActionSpace2` checker: whether the action space is consistent among the state space.
- 6) `Reward` checker: whether the reward function matches with the assumption.
- 7) `GoalState` checker: whether the goal state matches with the assumption.
- 8) `TransitionFunction` checker: whether the transition function matches the assumed transition patterns.
- 9) `Collision` checker: whether detected collisions are correct.

Moreover, two *output* alignment checkers are implemented to check the output of the generator: 1) *Success* checker: whether the success probability estimated by the robot exceeds 0.5; 2) *Approach* checker: whether the task completion time estimated by the robot keeps decreasing.

- Three different progress checkers are also created as follows.
- 1) `RobotMeanSpeed` checker: the robot’s mean speed, given by the Manhattan distance traveled from the start position divided by the moves taken.
 - 2) `CollisionProportion` checker: actual number of collisions the robot has experienced divided by the number of collisions allowed.
 - 3) `DistanceProportion` checker: the Manhattan distance the robot has traveled from the start position, divided by the Manhattan distance from the start position to the goal position.

4) *Study-Specific Details: Purported Performance*. The purported performance is estimated based on the algorithm described in [56].

Raw weight. The raw weight in this case study is computed as follows. If all the distances are nonzero, w_{raw}^n is given by

$$w_{\text{raw}}^n = 1/\|\mathbf{D}, \mathbf{f}(\mathbf{o}_t), \mathbf{f}^n\|_1. \quad (15)$$

If some of the distances are zero, then w_{raw}^n is assigned one for zero-distance neighbors and zero for nonzero-distance neighbors.

Distance functions. For \mathbf{D}_{all} , the coefficients for all alignment checkers are set to 1, while the coefficients for the progress

TABLE III
REPRESENTATIVE VARIATIONS CONDUCTED IN THE MAZE-BASED DISCRETE TIME MARKOV CHAIN STUDY WHEN THE ACTUAL WORLD WAS WORLD 1. SIMILAR KINDS OF VARIATIONS IN CONFIGURATIONS WERE MADE FOR OTHER ACTUAL WORLDS. FOR THE ACTION SET: U=UP, D=DOWN, R=RIGHT, L=LEFT. THE REWARD FUNCTION IS REPRESENTED BY A TUPLE (VALID STATE REWARD, COLLISION REWARD, GOAL REWARD)

World Index	State Space	Action Set	Trans. Prob.	Reward
1	Fig. 5(a)	{u, r}	0.7	(0, -1, 2)
2	Fig. 5(a)	{u, l, r}	0.7	(0, -1, 2)
3	Fig. 5(a)	{u, r}	0.5	(0, -1, 2)
4	Fig. 5(a)	{u, r}	0.7	(0, -3, 2)
5	Fig. 5(c)	{u, r}	0.7	(0, -1, 2)
6	Fig. 5(d)	{u, d, l, r}	0.7	(0, -1, 2)
7	Fig. 5(a)	{u, d, l, r}	0.5	(0, -1, 2)
8	Fig. 5(b)	{u, r}	0.7	(0, -3, 2)

checkers (average speed, proportion of traveled distance, proportion of taken collisions) are set to 1, 5, and 1, respectively. $\mathbf{D}_{\text{alignment}}$ and $\mathbf{D}_{\text{progress}}$ are from (11).

5) *Training and Evaluation*: We collect data from simulations in three different simulated (actual) worlds with only slight differences between each other. The first actual world is configured as World 1 in Table III. The second actual world differs from World 1 in having a higher transition probability (0.95) while the third actual world differs from World 1 in having more walls near the robot start position [see Fig. 5(g)].

For each actual world, we consider two sets of assumed worlds (i.e., the world the robot initially assumes it is in). The first set includes a world identical to the actual world and another ten worlds with only one component varied from the actual world. While the second set contains ten worlds with two components varied from the actual world. Table III lists a few representatives assumed worlds for the first actual world. Note how:

- 1) World 1 is identical to the actual world;
- 2) Worlds 2–5 each have one component varied from World 1;
- 3) Worlds 6–8 each have two components varied from World 1.

Similar assumed worlds are used for the other two actual worlds.

Since the assumed worlds in the second set differ from the actual world more than the assumed worlds in the first set, we train the kNN model on data from the first set and test the model on data from the second set, so that we can check the capability of the model to generalize from easier scenarios and predict well in harder scenarios. We conduct 50 simulations for each assumed world.

As in the other case studies, in addition to evaluating the ability of AAT to predict success and failure in worlds with static conditions, we also analyze its effectiveness when conditions in the world suddenly change. Thus, we consider a scenario in which a sudden change is introduced about 15 time steps into the simulation. In this scenario, the actual world is initially configured as World 1 (see Table III) for the first 15 time steps. While after 15 time steps, the state space is suddenly altered to Fig. 5(g) (i.e., more walls are added near the robot’s start state). Meanwhile, the robot assumes that it is in World 1 during the entire simulation.

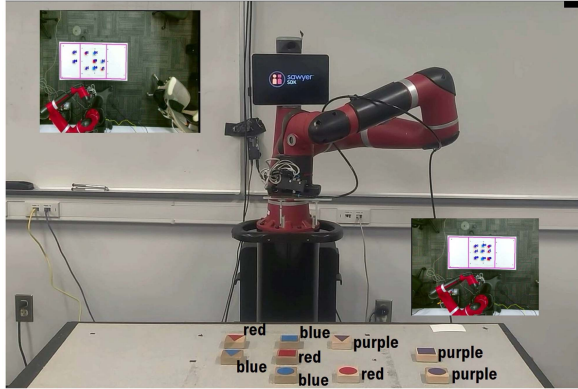


Fig. 7. Sawyer robot was tasked with organizing blocks on a table in the block-manipulation case study.

C. Block Manipulation by a Sawyer Robot

1) *Task Domain*: A Sawyer Robot [57] is tasked with organizing blocks, each of which has a different color and shape, on a table. It has a 7 DOF arm for manipulation and a pneumatic gripper attached to the arm for grasping. The robot uses a Kinect2 camera [58] mounted on the ceiling to perceive the environment from a bird's eye view.

Fig. 7 shows the task setup consisting of the robot and a table. The table has nine blocks of three different colors and shapes. The robot's task is to arrange these blocks in the center of the table in a desired arrangement within a certain amount of time. The top left inset figure shows the environment as perceived and expected by the robot. Any other entities on the table other than the blocks of the three shapes and colors are perceived as foreign objects. The bottom right inset figure shows the desired arrangement of blocks at the end of the task.

Task difficulty is controlled in part by varying the initial configuration of blocks on the table. Based on the initial positions of the blocks, it can require a different number of total swaps to complete the task, where one swap is defined as one complete pick and place operation. Thus, at a given time for each visible block, the calculated number of swaps can be either: a) 0 : If the block is already at goal position, b) 1 : If the block is not at the goal position but the goal position is free, or c) 2 : If the block is not at the goal position and the goal position is occupied by another block. We subjectively set the acceptable performance standard for this task at 300 s (i.e., the robot succeeds if it sets up the table in less than 300 s), regardless of the number of swaps required to properly arrange the blocks on the table.

2) *Generators and Assumptions*: The Sawyer robot uses three different generators to perform the table setup task: 1) a mapper, 2) a path planner, and 3) a controller. The mapper takes as input the camera image and the point cloud (produced by the distance sensor on the Kinect) and creates a map of the environment by detecting the table and the blocks and localizing the blocks in the environment with respect to the robot base. Mapper assumptions are as follows.

- 1) The camera produces up-to-date images.
- 2) The camera sees the colors according to specification.

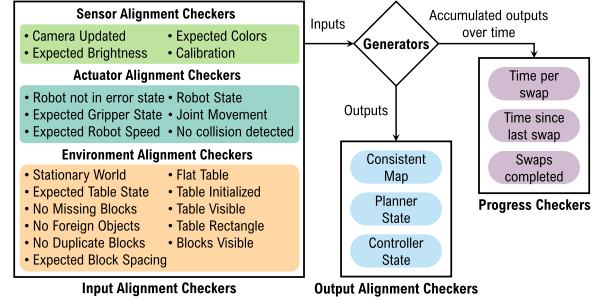


Fig. 8. Alignment and Progress checkers for the block manipulation case study.

- 3) Image brightness of the camera image is according to specification.
- 4) The table is visible and is a flat rectangle.
- 5) There are nine blocks visible on the table of a combination of three colors (red, blue, purple) and three shapes (square, triangle and circle), and there are no missing or duplicate blocks and no foreign objects present on the table.
- 6) The blocks are being consistently detected and have sufficient spacing between them. The assumed output of the mapper is a consistent and accurate map of the table.

The colors and shapes of blocks were detected using color and edge detection techniques using OpenCV [59].

The planner takes as input the map, created by the mapper, and consists of two subcomponents: the high-level and the low-level planner. The high-level planner combines a history of the block attempts with the map and goal specification to plan which block to move next. Given the selected block and the map, the low level planner plans a path for the arm to move the gripper from the current position to the current block location (pick), and then (once grasped) to move the block to the desired position on the table (place). The low level planner uses MoveIt [60] for motion planning. In addition to assuming that the map provided by the mapper is correct, the planner assumes that the 1) world is stationary (other than the robot movement and the block moved by the robot) and that 2) the planner output is reasonable given the task (i.e., there exists a trajectory that can be executed within a reasonable time-bound given the planner selected from the MoveIt library).

The controller takes as input the planned path and outputs commands to the robot's arm and gripper, moving the robot arm along the chosen path to pick and place selected blocks. In addition to assuming the path selected by the planner is desirable, the controller assumes:

- 1) the robot's arm moves at the expected speed within the set bounds;
- 2) the gripper is working properly;
- 3) the robot joint movements are expected;
- 4) up-to-date robot state is available, with robot not being in an error state;
- 5) no collision is detected during trajectory execution. The expected output of the controller is joint movements to move the robot's arm.

3) *Alignment and Progress Checkers*: Fig. 8 summarizes both the veracity and progress checkers we implemented for this case study. Alignment checkers were implemented for each assumption with two exceptions. Due to the difficulty in detecting in real time if the robot’s camera is not calibrated properly, we did not implement a camera calibration checker for this study. In addition, we did not implement an alignment checker to evaluate expected planner output. Future work should explore how such alignment checkers could be implemented to check both of these assumptions. In addition to the alignment checkers, we implemented three progress checkers, which monitored the 1) mean speed of the robot (computed as the running average of time taken per swap), 2) the time elapsed since the last swap was complete, and 3) the number of completed swaps divided by the total number of required swaps identified at the start of the task.

4) *Study-Specific Details: Purported performance*: The purported performance for the robot is given by

$$\bar{C}([o_t, \dots]) = \text{SwapsRemaining} \cdot T_0 \quad (16)$$

where, SwapsRemaining is the remaining number of swap operations at time t and T_0 is the average time for the robot to complete one swap under nominal circumstances. T_0 was precalculated empirically to be approximately nine seconds. *Raw weight*: The raw weight for this case study is computed using (14), but with values $\alpha = 3, \beta = 0.2$.

Distance Functions. The weighting vector for all checkers \mathbf{D}_{all} is the 1-vector of size $M + K$, $[1, \dots, 1] \in \mathbb{R}^{M+K}$, where M and K are the numbers of alignment and progress checkers, respectively. Therefore, the weighting vectors for alignment and progress checkers, $\mathbf{D}_{\text{alignment}}$ and $\mathbf{D}_{\text{progress}}$, are the 1-vectors with size M and K , respectively.

5) *Training and Evaluation*: For training data, we conducted 30 trial runs with the Sawyer robot setting up the table, each with a different initial table configuration. In ten of these trial runs, all conditions were normal (e.g., lighting conditions, block placement, etc., match robot assumptions, though noisy veracity checkers still sometimes reported violations). In the other 20 training runs, we varied aspects of the robot’s camera (e.g., hue, saturation, exposure, brightness) to impact its vision. Many of these alterations caused violations of the assumptions made by the robot’s mapper generator, causing the robot to either not detect blocks or misclassify them.

As in the previous case studies, we evaluate the resulting predictor (trained on the stated training set) in two conditions: static scenarios and sudden-change scenarios. For the static test scenarios, we conducted an additional 30 trial runs (ten in nominal conditions, 20 with variations in camera parameters), each with a different initial table configuration.

Two sudden-change scenarios were used. In the first scenario, the robot operated under nominal conditions for the first 54 s. At that time, we suddenly altered the hue and brightness on the robot’s camera. Both the nominal initial conditions and the conditions after the change in this scenario are representative of the conditions of some trial runs in the robot’s training data. In the second scenario, we tricked the robot for the first 60 s of the run by moving any block the robot reached for before it could grasp it. After 60 s, we stopped doing this (conditions returned to

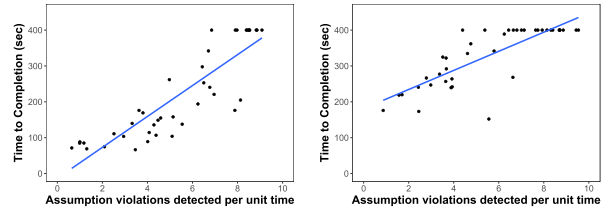


Fig. 9. Average assumption violations detected per unit time compared with task completion time in Mazes 2 (left) and 3 (right) in the navigation study. Similar trends existed in Mazes 1 and 4. Lines depict linear fits to the data for visualization purposes, but a linear relationship is not implied.

normal). The condition in the first 60 s of this scenario (moving the block from the robot) is not represented in the training data.

All runs in this study are terminated after either successful task completion or after 300 s, whichever comes sooner.

VI. RESULTS

This section first demonstrates the correlation between assumption veracity and robot performance, and then evaluates AAT when sudden-changes occur during task execution.

A. Correlation Between Veracity and Robot Performance

AAT relies on the hypothesis that the ability to evaluate the veracity of generator assumptions can provide rich insight into a robot’s performance. Since this hypothesis is intuitive, we only use the data from the navigation case study to test it, for simplicity. Fig. 9 shows the correlation between robot performance (measured as time-to-completion, where higher time indicates worse performance) and the number of detected assumption violations per unit time for Maze 2 and 3 in the navigation study. In each world, task performance was negatively correlated with the number of violations in assumptions. As the number of detected violations in assumptions per unit time increases, completion time also tends to increase. Pearson correlations confirm statistically significant ($p < 0.001$) and strong correlations between the number of assumption violations and completion time in each world, with $r = 0.778$ in Maze 1, $r = 0.874$ in Maze 2, $r = 0.802$ in Maze 3, and $r = 0.890$ in Maze 4 (results for Maze 1 and 4 are not shown in the figure). Note that simulations were automatically terminated after 400 s, meaning that completion times marked as 400 s would have been higher had the simulation been run to completion. These results indicate that tracking the veracity of assumptions can be useful for understanding robot performance.

B. Evaluating AAT Given a Consistent Configuration

The ability of AAT to discriminate between success and failure during task execution across all three case studies is summarized in Fig. 10. Fig. 10(a)–(c) shows results for the navigation case study, Fig. 10(d)–(f) shows results for the maze-based discrete Markov chain, and Fig. 10(g)–(i) shows results for block manipulation by a Sawyer robot. The figure shows similar trends across all three case studies.

Fig. 10(a), (d), and (g) shows that when both alignment and progress checkers are used (i.e., we use the distance function

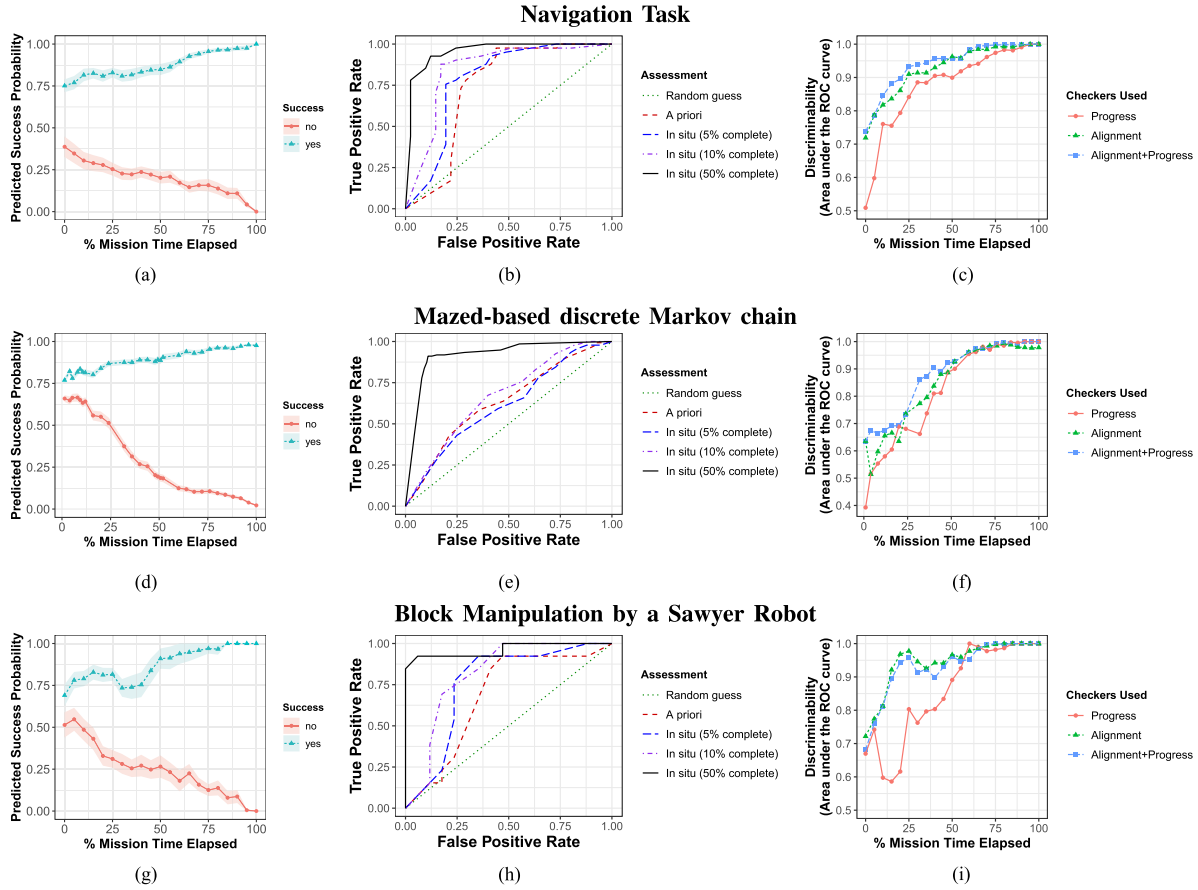


Fig. 10. AAT effectively discriminates between success and failure in scenarios with a consistent configuration. (a)–(c) navigation task; (d)–(f) maze-based discrete time Markov chain; (g)–(i) block manipulation by a Sawyer robot. (a), (d), (g) The average predicted probability of success over time (using the weighting vector \mathbf{D}_{all}) for cases in which the robot completed and did not complete the mission on time. Error ribbons show the standard error of the mean. (b), (e), (h) ROC curves showing the ability of the performance estimator to determine whether the robot’s performance meets an acceptable threshold at various points in time (using the weighting vector \mathbf{D}_{all}). (c), (f), (i) The discriminability of the predictor (as measured by the area under the ROC curve—0.5 indicates no discriminability) over time using the three different weighting vectors.

\mathbf{D}_{all}), AAT effectively differentiates between scenarios where the robot succeeds and scenarios where the robot does not succeed. In instances in which the robot fails to complete the task on time, the algorithm predicts substantially lower probability of success, on average, than scenarios in which the robot eventually succeeds in the task. These differences in predicted probability of success begin in the early stages of the mission, suggesting that the predictor not only makes effective in situ assessments, but it also can make effective a priori assessments of proficiency at the onset of the mission.

Fig. 10(b), (e), and (h) shows ROC curves for the binary proficiency assessor classifier at various times throughout the mission. In these figures, the percentage of mission elapsed is calculated according to the respective performance standard, where the time corresponding to the performance standard is equivalent to 100% mission time elapsed. Line types (and colors) indicate elapsed mission percentage. The lower left corner of the ROC curve corresponds to a success criterion of $\theta = 1$ and the upper right corner of the ROC curve corresponds to $\theta = 0$ (see

Section IV-C). These figures show high discriminability in all case studies by the time 50% of mission time has elapsed.

However, high discriminability comes much earlier, on average, in the navigation and block manipulation case studies than in the discrete maze study. In the former two studies, the robots’ camera sensors allow them to observe the entire environment at once, rather than only receiving limited local information about its current state (as is the case in the discrete maze). This larger amount of information allows the robot to make more extensive and comprehensive evaluations about its assumptions at each time step. For example, the robot can perform assessments about whether its camera is returning images that meet assumed specifications and whether or not the entire environment meets assumed specifications (with respect to stationarity, recognition of objects, etc.). For this reason, the robots in the navigation and block-manipulation studies are often able to detect many violations of generator assumptions long before the robot’s generators begin to fail. By contrast, the robot in the discrete maze-based simulations gets to sense only one state of the environment at

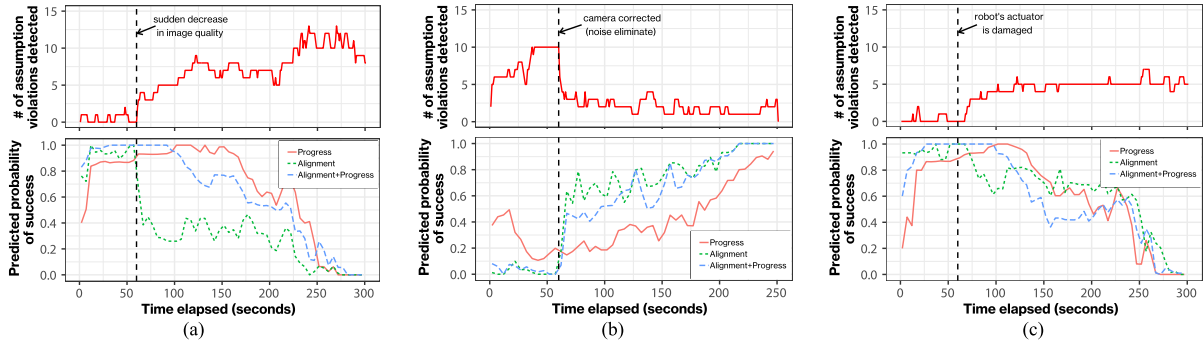


Fig. 11. Number of violations detected and the predicted probability of success over time for sudden-change scenarios in the navigation study using three different weighting vectors (see legends). (a) Scenario in which everything is normal for the first 60 s. After that, camera noise begins. (b) Scenario in which there is a large amount of noise in the camera image for the first 60 s. After that, camera noise is corrected so that all assumptions are met. (c) Scenario in which, for the first 60 s, everything is normal. After that, the robot’s wheel is damaged (bias = -4.0) causing it to drift. The extent of the bias produces conditions that are not represented in the training data, hence predictions do not correct quickly.

a time. It therefore must move more before it is able to reason about whether assumptions made by the generators are or are not met.

Finally, Fig. 10(c), (f), and (i) compares the effectiveness of the three weighting vectors ($D_{\text{alignment}}$, D_{progress} , and D_{all}) in the three studies by plotting the area under the ROC curve over time. In these plots, a discriminability of 0.5 indicates that the predictor cannot differentiate at all between success and failure, and a discriminability of 1.0 indicates that the predictor can fully differentiate between success and failure. While all predictors eventually produce high discriminability in all three case studies, predictors that use alignment checkers produce higher discriminability early in the mission. The distinction between the effectiveness of alignment checkers and progress checkers in the first half of a mission is particularly stark in the navigation and block-manipulation studies.

These results indicate that alignment checkers enable accurate assessments of robot proficiency using AAT early in a mission. On the other hand, predictors based on progress checkers are typically unable to discriminate effectively between success and failure as soon.

C. Evaluating AAT in Sudden-Change Scenarios

The results in the previous section indicate that alignment checkers are typically more effective than progress checkers early in the mission. To more fully understand the strengths and weaknesses of alignment and progress checkers, we consider scenarios in which conditions change during the mission. We present results for these *sudden-change* scenarios for each case study separately given the use of only alignment checkers, only progress checkers, and both kinds of checkers.

1) *Navigation Task*: The results of the three sudden-change scenarios described in Section V-A5 are shown in Fig. 11. These figures show both the number of assumption violations detected at each time step (top) as well as the predicted probability of success by each predictor (predictors vary by which veracity checkers are used). In the first scenario [see Fig. 11(a)], conditions are normal (i.e., they meet the generators’ assumptions)

for the first 60 s, resulting in few assumption violations being detected by the alignment checkers, and each of the three predictors predicts a high probability of success. At 60 s, camera noise is introduced at a level that is within the bounds of samples from the training data (noise was 0.25; see Table I), leading to the robot failing to accomplish the task within the allotted time. As soon as camera quality is changed, the robot immediately starts detecting assumption violations [see top panel of Fig. 11(a)]. As a result, the alignment-only predictor immediately predicts a substantially reduced probability of success. However, the progress-only predictor is biased by the fact that the robot initially performed very well and, consequently, takes a long time to adapt its predictions after the change. The combined predictor adapts slower than alignment-only but faster than progress-only.

Results of a second scenario, shown in Fig. 11(b), tell a similar story. In this scenario, the camera image quality is poor during the first 60 s, but is corrected after 60 s. As a result, the robot succeeds in the task. Once the camera noise is corrected, the alignment checkers immediately detect fewer assumption violations. In addition, the alignment-only and combined predictors almost immediately estimate a higher probability of success (the increase for the combined predictor is lower due to the impact from progress checkers), which gradually increases as the scenario goes on. However, the progress-only predictor takes a long time to predict success after the sudden change given that the robot’s performance was so poor during the first 60 s of the mission.

The results of these first two sudden-change scenarios highlight strength of alignment checkers and a weakness of progress checkers. Consistent with what we observed in the last section, alignment checkers allow AAT to quickly detect whether conditions are likely to produce task success when conditions are representative of its past experiences. By contrast, progress checkers, who are based on the robot’s progress on the task so far, are often unable to adapt quickly.

A third sudden-change scenario, however, identifies limitations of our implementation of AAT based only on alignment checkers. In this scenario, depicted in Fig. 11(c), all generator assumptions are met for the first 60 s (as in the first scenario). However, at 60 s, the robot’s motors are damaged (i.e., robot

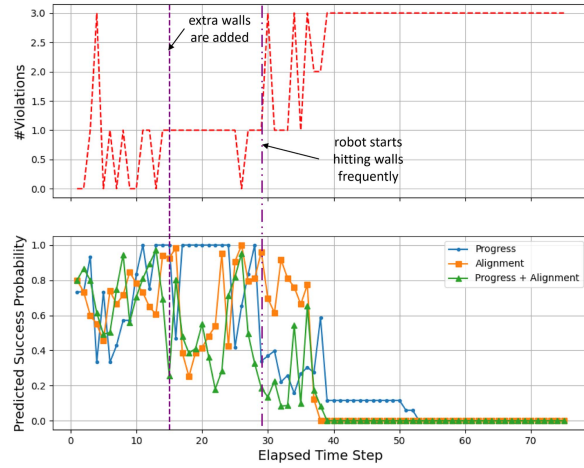


Fig. 12. Assumption violations detected and predicted probability of success for a sudden-change scenario in the discrete maze study. For the first 15 time steps, the environment is configured as in Fig. 5(a). At time-step 15, more walls are added near the robots start state [see Fig. 5(g)]. The robot’s generators assume the initial world configuration at all times.

bias is set to -4.5), which causes the robot to move in circles when it attempts to go straight. While variations in robot bias were part of the robot’s training set, this variation is outside the range of the training data (the variation is more severe; see Table I). As such, after the change, the robot identifies the assumption violations, but predicts that it will still be able to complete the task in time. Only after substantial time does its prediction of success begin to decrease for all predictors. This result indicates that, for predictions based on alignment checkers to be accurate, conditions must be represented in the robot’s training data. Future work should explore how to augment AAT so that it considers whether conditions are within its training experiences.

2) *Maze-Based Discrete-Time Markov Chain*: In the single sudden-change change scenario we consider in the maze-based discrete-time Markov chain environment, the robot believes that it is in the world shown in Fig. 5(a). This assumption is correct for the first 15 moves, and hence relatively few violations are detected (see Fig. 12). At move 15, the environment is suddenly altered to the world shown in Fig. 5(c) (i.e., walls are added to the environment). The change is correctly identified by one of the alignment checkers at time step 16, specifically the `StateSpace2` checker (see Section V-B2). Interestingly, the number of total assumption violations shown in the top panel remains the same, which is because another alignment checker (the `Approach` assumption checker) turns off.

The bottom panel of Fig. 12 shows the predictions made with the three different weighting vectors. As expected, the alignment-only predictor estimates a decreased probability of success once the state-space change is identified. However, that decrease lasts for only two time steps. After time step 18, the prediction of the alignment predictor rebounds since, even though there are extra walls adjacent to the robot, the robot still manages to keep approaching the goal state, which inhibits and finally overtakes the effect of the state space assumption

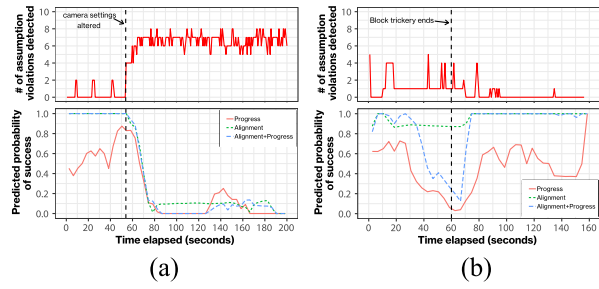


Fig. 13. Assumption violations detected and predicted probability of success for sudden-change scenarios in the block manipulation study using three different weighting vectors (see legends). (a) Scenario in which, for the first 54 s, everything is normal. After that, camera settings are modified resulting in noise in the camera image. (b) Scenario in which, for the first 60 s, the block that the robot was seeking was moved every time the robot tried to grasp it (a condition that was not represented in the training data). After 60 s, the trickery ended (conditions became normal).

violation. After time step 29, the robot starts hitting a wall frequently. However, since the set of alignment checkers did not include a checker that evaluated the assumption that the robot should not hit a wall; the alignment-only predictor is not able to respond to the frequent collisions until time step 36.

On the other hand the progress-only predictor predicts success until about time step 28, since it is not aware of the change and the robot still keeps approaching the goal state. After time step 29 when the robot starts hitting a wall frequently, the `Collision` progress checker (which counts collisions) starts accumulating, leading to the progress predictor dramatically decreasing its prediction. The combined predictor responds to both the sudden change and the frequent collisions, making it the best predictor for this trial. The result for this case study not only strengthens the claim that alignment-only predictors are more sensitive to sudden change than progress-only predictors, but also indicates the importance of covering as many assumptions as possible.

3) *Block Manipulation by a Sawyer Robot*: The results of the two sudden-change scenarios with Sawyer are consistent with those of the other two case studies (see Fig. 13). In the scenario corresponding to Fig. 13(a), conditions were normal for the first 54 s, after which the hue and brightness on the camera sensor were changed so that they no longer met generator assumptions. This led to the robot failing to finish arranging the blocks on the table. Prior to the change, the alignment-only predictor predicted that the robot would complete the task with near certainty and the progress-only predictor was less certain. After the change, all predictors quickly decreased their predictions of success. The rapid response is possible because the altered conditions were represented in the training data.

In the second scenario, depicted in Fig. 13(b), a person moved the block whenever the robot attempted to pick it up during the first 60 s of task execution. After 60 s, this trickery ended and conditions became normal, which led to the robot completing the task in time. Although the robot would have been unable to successfully complete the task had the trickery continued, the alignment-only predictor still predicted a probability of success for the first 60 s for two reasons. First, the initial conditions were outside the robot’s training data. Second, a key assumption

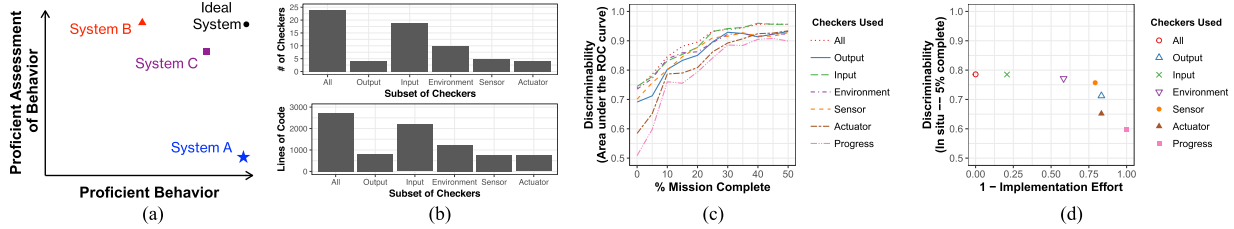


Fig. 14. Illustration of the tradeoff between implementation effort and discriminability of AAT. (a) Hypothetical tradeoff for designing robots that have both proficient behaviors and are proficient at assessing their behavior. (b) Implementation effort to create various subsets of assumption checkers as measured by number of checkers (top) and lines of computer code (bottom). Since progress checkers required minimal effort to create in this case study, implementation effort for progress checkers is not shown. (c) Discriminability of AAT as a function of mission time based on the area under the ROC curve for various subsets of assumption checkers. (d) Tradeoff between implementation effort and discriminability after 5% of the mission was complete. $1 - \text{Implementation Effort}$ is computed as one minus the percentage of checkers used. An ideal outcome would be in the upper right-hand corner—which would indicate perfect discriminability achieved without any implementation effort.

checker, one designed to detect a nonstationary world, did not operate according to specification. Thus, the alignment checkers were not able to identify assumption violations and the resulting alignment-only predictor inaccurately predicted success. The progress-only predictor performed better because the trickier impacted robot performance resulting in reattempts and/or failures to pick up the block(s). Thus, both the progress-only and the combined predictors figured out that the robot would fail at the task with high probability by about 30–40 s into the mission. When the human stopped moving the blocks away from the robot, the robot successfully completed the task less than 100 s later but the progress-only predictor was slow to detect the changed circumstances, while the combined checker adapted its predictions more quickly.

This scenario highlights the potential benefits of progress checkers. Progress checkers, though often slow to diagnose success and failure conditions (because they need accumulated data), have an important role in that they can successfully predict success or failure when alignment predictors fail. This is important because alignment-only predictors will fail if an assumption is missed, if an alignment checker fails, or if the training data insufficiently covers a failure condition. Thus, an effective combination of alignment and progress checkers can be useful in performing proficiency self-assessment with AAT.

VII. DESIGN TRADEOFFS

A. Proficient Behavior Versus Proficient Assessment of Behavior

The case studies demonstrate that AAT can effectively perform proficiency self-assessment. However, it took more effort to implement AAT in these systems than it took to create the robot generators. For example, we implemented the generators for the navigation case study using about 1100 lines of computer code, while our implementation of AAT (including assessing input and output assumptions and defining the prediction function F) required over 2700 lines of code. Although lines of code do not perfectly represent effort; the number of lines of code in this case study coincides with our observation that it took more effort to implement AAT than to implement the generators themselves.

This subjective observation about effort highlights a potential tradeoff in designing autonomous robot systems: Should system designers focus more on developing proficient autonomous behavior or on creating proficient assessments of behavior? This tradeoff is visualized in Fig. 14(a), which illustrates several hypothetical systems including an ideal system that is both proficient and capable of self-assessing proficiency. System A represents a system in which designers spent all of their time developing proficient robot behaviors and ignored the problem of proficiency self-assessment. Such a system would be acceptable in scenarios in which either the robot never fails or when it is unnecessary to identify failures. However, in situations in which failing to identify failures has large implications, System B (in which system designers focus more extensively on developing proficiency self-assessment capabilities) might be a better choice. System C illustrates a potential *middle ground* in which the robot has neither ideal behavior generation nor ideal behavior assessment, but design effort is allocated to produce reasonable proficiency in both behavior generation and behavior assessment.

B. Implementation Effort Versus Efficacy of AAT

In AAT, one way to save the design effort to proficient assessment of behavior is limiting the number of assumptions that are tracked, which we argue has complex effects on AAT efficacy. On the one hand, appropriate *simplifications* of checkers would not hurt AAT efficacy too badly, since some checkers correlate with others. On the other hand, checker *redundancy* would help to compensate for false outputs of unstable checkers, therefore making AAT more robust.

1) *Checker Simplification:* We observed that veracity assessments among assumption checkers tended to correlate with each other. For example, in the navigation case study, changes in the veracity of assumptions made about the robot's sensors and actuators subjectively correlated with (and likely caused) assumption violations about generator outputs. In this example, violated assumptions about generator output are symptoms of violated input assumptions. In addition, we observed that assessments of some input assumptions seemed to correlate with each other. For example, in the navigation case study, violating

assumptions about the amount of sensor noise often caused the robot to believe that its actuators were not behaving according to assumptions. These anecdotal observations give credence to the idea that not all assumptions need to be tracked for the robot to self-assess its proficiency.

We explored that idea by evaluating how well AAT estimated robot performance in the navigation case study given various subsets of checkers identified in Fig. 4. Since these subsets require different amounts of effort to implement, as measured both by the useful (though imperfect) metrics of lines of code and number of alignment checkers [see Fig. 14(b)], it is informative to evaluate how different subsets impact the ability of the robot to assess its performance.

Fig. 14(c) uses the area under the ROC curve to compare the ability of each subset of checkers to accurately discriminate between success and failure over the initial stages of the robot’s mission. The figure shows that all subsets of checkers differentiate between success and failure substantially better than random guesses even at the start of the task. After 10% of the mission is completed, all subsets of checkers produce high discriminability, though with some small variations. However, the subsets of *output* and *actuator* alignment checkers (see Fig. 4) initially produced lower discriminability than the other subsets of checkers, suggesting that assessment accuracy is affected by which checker subset is chosen but that some subsets can be very accurate. These results indicate that not all generator assumptions need to be tracked in order for the robot to do a reasonably good job of assessing its own proficiency, but that the subset of checkers chosen is important.

The tradeoff between the implementation effort and discriminability of AAT in the navigation case study is plotted in Fig. 14(d). Ideally, the system would produce full discriminability at no effort to the system creator, which would produce a point in the upper right-hand side corner. In practice, the set of *sensor* alignment checkers (see Fig. 4) seems to provide a good tradeoff in producing high discriminability with relatively low implementation effort in the scenarios tested.

2) *Checker Redundancy*: When some checkers are not robust or require extra implementation effort to be robust, redundancy in checkers can help improve the overall stability of AAT. For example, in the block-manipulation case study, a robust implementation of the `ExpectedBrightness` and `ExpectedTableState` alignment checkers would require considerable effort. Even if these alignment checkers yield incorrect veracity assessment, some robust and easier-to-implement checkers, such as the `BlocksVisible` and `NoMissingBlocks` checkers, will likely yield accurate results that could compensate for those errors. Fig. 15 illustrates that situation in a specific robot trial in the block-manipulation case study. In the first 20 s of the trial, nominal conditions prevailed. After 20 s, there were slight and unintentional changes in lighting, causing a few alignment checkers to indicate assumption violations, particularly `ExpectedBrightness` and `ExpectedTableState`. Since complementary alignment checkers (`NoMissingBlocks` and `BlocksVisible`) did not indicate assumption failures and since training data included such cases, the predictors still indicated likely success. Indeed, the robot completed the task in just under a minute.

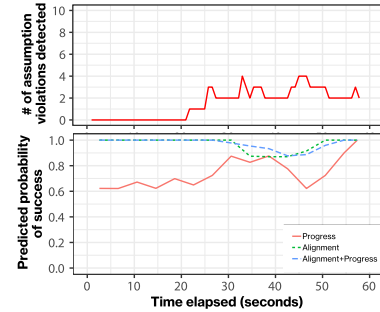


Fig. 15. AAT assessments in a single run of the block-manipulation task under normal conditions.

VIII. DISCUSSION

In our implementations, we used the kNN algorithm to make proficiency assessments from past experiences since kNN is reasonably effective given limited data, such as the scenarios we consider in this article. Furthermore, the solutions of the kNN algorithm are easily interpreted when explanations of proficiency self-assessment are required. That said, other, potentially more powerful, machine-learning algorithms could be chosen to learn η in future work.

Results in Section VI demonstrates that AAT can produce reasonably accurate assessment of proficiency across many task domains. This provides some evidence that AAT appears to generalize better than the methods presented in [12], [13]. AAT is purely data-driven and does not require a priori probabilities of the primitive tasks needed in [11]. Finally, FaMSec [20], [21], [22], [23], [24] uses only progress checkers, while AAT uses both alignment and progress checkers.

The three case studies demonstrate that *alignment* and *progress* checkers have different strengths and weaknesses. It is not always clear how violations in assumptions detected by the alignment checkers will impact robot proficiency. The case studies also suggest that alignment assessments tend to respond more quickly to potential issues that arise, which can lead to earlier detection of potential failures than assessments made by progress checkers. On the other hand, the case studies suggest that a successful system might require only a small number of progress checkers since they effectually make aggregate assessments of the behavior of multiple generators. However, the progress checkers used in this article do not provide detailed knowledge for the reasons for failure or success, nor do they tend to detect potential issues as quickly as alignment checkers. Importantly, results in Section VI-C3 suggest that progress checkers act as a *catch all* that can compensate for missing assumptions, alignment checker failures, and failure conditions uncovered in training data. Therefore, proficiency self-assessment benefits from using both forms of checkers. Future work should address best balance design effort in creating these two forms of checkers.

Observe that the result in Section VII-B does not suggest a specific “optimal” tradeoff choice for implementing AAT, such as only implementing sensor-based alignment checkers. Different

systems and scenarios will likely produce different results with respect to these tradeoffs. However, these results demonstrate that identifying effective subsets of checkers to implement can substantially reduce the effort required to implement AAT while keeping the efficacy of AAT high. Moreover, we emphasize that the tradeoff between time spent generating quality behaviors and performing quality assessment can be misleading. The metaphor of writing test cases for computer code is applicable. Frequently, doing unit testing in software engineering produces better code in less time. We hypothesize that something similar will likely occur with creating alignment checkers in parallel with creating behavior generators. Explicitly identifying and tracking assumptions made in the implementation of decision-making algorithms can help to produce more effective robot behavior.

IX. LIMITATIONS

First, AAT requires the system designer to have a deep understanding of the robot system to be designed and to put in extra implementation effort. This could be limiting if the designer is integrating a set of off-the-shelf algorithms in the robot system. Moreover, it can be difficult or even infeasible to identify all assumptions. We think that using straightforward and easy-to-implement progress checkers that aggregate many assumptions could ease this issue. Moreover, we hypothesize that many checkers could be designed to be reusable across tasks and robot systems. If checkers can be designed to be easily tuned or modified, the implementation effort and difficulty of implementing AAT would be reduced. Many off-the-shelf robotic systems use common tools and libraries, such as ROS as a middleware, the MoveIt library for planning and manipulation, and the ROS navigation stack for navigation applications. If these common tools were labeled with the list of assumptions that they make or paired with appropriate alignment checkers then generalizability to new tasks and robots could be facilitated. In addition, systems with similar generators or applications have similar assumptions, which make it easier to reuse checkers across applications. Second, the two simulated robot systems both involve simplifications that are not very realistic in a real-world case. Such simulation-reality gaps could make veracity checking more challenging for real-world applications of AAT. Indeed, we notice that it is more difficult to implement checkers for the block manipulation study than the two simulated studies. As discussed previously, we hypothesize that keeping a certain level of checker redundancy could ease such issues. We also hypothesize that some existing algorithms for robot perception could be adopted to implement real-world checkers, such as algorithms described in [61], [62], [63]. Nevertheless, we acknowledge that general application of the AAT framework may be challenging in large, real-world systems.

Third, the three case studies have relatively simple tasks with achievement-oriented goals [34]. Applications of AAT to complex tasks such as autonomous driving or long-duration autonomy could be challenging. When a complex task can be decomposed into simple subtasks, then it is conceivable that the composition of AAT components designed for each subtask could yield an AAT system for the complex task.

Fourth, the proposed AAT framework assumes single-robot systems. We envision two potential ways of adapting AAT to collaborative robot teams. One way is to implement AAT for each robot in the team and then create mechanisms that combine proficiency assessment of each robot to assess the team's proficiency. The other is identifying assumptions and implementing checkers for the whole team.

Finally, the proposed AAT framework assumes additive cost functions. For nonadditive cost functions, the framework could potentially be adapted by adjusting (4)–(5).

X. SUMMARY AND FUTURE WORK

This article formalizes a method for designing robots that perform proficiency self-assessment. This method, called AAT, is developed from the perspective that proficiency self-assessment is awareness of how one's generators (i.e., decision-making algorithms) interact and align with the environment(s), robot hardware, and task(s) under consideration. In AAT, the robot continually monitors the veracity of input and output assumptions made in the construction of its generators, and then use these assessments to estimate the robot's ability to perform the task. Three case studies (a simulated robot navigating in a discrete maze environment, a simulated robot navigating in a continuous environment, and a real robot arranging blocks on a table) demonstrate that AAT can perform informative proficiency self-assessment often even before a task has been started. However, results also show that the quality of AAT's performance predictions do vary based on the coverage and quality of assumption checkers that are created, as well as how representative training data is to the current scenario. Results also demonstrate opportunities and challenges for integrating explicit evaluations of assumptions with evaluations of proficiency that are based on robot progress on its task.

Future work is needed to better establish AAT's usefulness as a systematic approach to proficiency self-assessment. This includes designing more general and reusable checkers, applying AAT to more complex robot systems (including collaborative robot teams with more complex tasks and nonadditive cost functions), better understanding the design tradeoffs between generator quality and alignment checker coverage, and using assumption tracking to develop explanations [64] about proficiency assessments. Further work is also needed to establish benchmarks for proficiency self-assessment [65].

REFERENCES

- [1] R. Dearden, T. Willeke, R. Simmons, V. Verma, F. Hutter, and S. Thrun, "Real-time fault detection and situational awareness for rovers: Report on the Mars technology program task," in *Proc. IEEE Aerosp. Conf. Proc.*, 2004, vol. 2, pp. 826–840.
- [2] 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.
- [3] S. Honig and T. Oron-Gilad, "Understanding and resolving failures in human-robot interaction: Literature review and model development," *Front. Psychol.*, vol. 9, 2018, Art. no. 861.
- [4] D. Das, S. Banerjee, and S. Chernova, "Explainable AI for robot failures: Generating explanations that improve user assistance in fault recovery," in *Proc. IEEE/ACM Intl. Conf. Hum.-Robot Interact.*, 2021, pp. 351–360.

- [5] Z. Han, D. Giger, J. Allspaw, M. S. Lee, H. Admoni, and H. A. Yanco, "Building the foundation of robot explanation generation using behavior trees," *J. Hum.-Robot Interact.*, vol. 10, no. 3, pp. 1–31, 2021.
- [6] A. Alvanpour, S. K. Das, C. K. Robinson, O. Nasraoui, and D. Popa, "Robot failure mode prediction with explainable machine learning," in *Proc. IEEE Int. Conf. Automat. Sci. Eng.*, 2020, pp. 61–66.
- [7] A. Ramesh, M. Chiou, and R. Stolkin, "Robot vitals and robot health: An intuitive approach to quantifying and communicating predicted robot performance degradation in human-robot teams," in *Proc. Companion ACM/IEEE Int. Conf. Hum.-Robot Interact.*, 2021, pp. 303–307.
- [8] D. Schreckenghost, T. Fong, T. Milam, E. Pacis, and H. Utz, "Real-time assessment of robot performance during remote exploration operations," in *Proc. IEEE Aerosp. Conf.*, 2009, pp. 1–13.
- [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] 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.
- [11] 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," in *Proc. HRI Workshop Assessing, Explaining, Conveying Robot Proficiency Hum.-Robot Teaming*, 2020, pp. 1–6.
- [12] A. Dutta, P. Dasgupta, J. Baca, and C. Nelson, "Towards autonomously predicting and learning a robot's efficiency in performing tasks," in *Proc. IEEE/WIC/ACM Int. Joint Conf. Web Intell. Intell. Agent Technol.*, 2013, vol. 3, pp. 92–95.
- [13] G. J. Burghouts, A. Huizing, and M. A. Neerinx, "Robotic self-assessment of competence," in *Proc. HRI Workshop Assessing, Explaining, Conveying Robot Proficiency Hum.-Robot Teaming*, 2020, pp. 1–7.
- [14] A. C. Morris, "Robotic introspection for exploration and mapping of subterranean environments," Ph.D. dissertation, Carnegie Mellon Univ., 2007.
- [15] H. Grimmert, R. Paul, R. Triebel, and I. Posner, "Knowing when we don't know: Introspective classification for mission-critical decision making," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2013, pp. 4531–4538.
- [16] H. Grimmert, R. Triebel, R. Paul, and I. Posner, "Introspective classification for robot perception," *Int. J. Robot. Res.*, vol. 35, no. 7, pp. 743–762, 2016.
- [17] R. Triebel, H. Grimmert, R. Paul, and I. Posner, "Driven learning for driving: How introspection improves semantic mapping," in *Proc. Robot. Res.*, Springer, 2016, pp. 449–465.
- [18] 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.
- [19] 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.
- [20] B. W. Israelsen and N. Ahmed, "A factor-based framework for decision-making competency self-assessment," in *Proc. AAAI SSS-22 Symp. Closing Assessment Loop: Communicating Proficiency Intent Hum.-Robot Teaming*, 2022, pp. 1–3.
- [21] M. Aitken, "Assured human-autonomy interaction through machine self-confidence," Master's thesis, Univ. Colorado, Boulder, USA, 2016.
- [22] M. Aitken, N. Ahmed, D. Lawrence, B. Argrow, and E. Frew, "Assurances and machine self-confidence for enhanced trust in autonomous systems," in *Proc. RSS Workshop Social Trust Auton. Syst.*, 2016, pp. 1–3.
- [23] B. Israelsen, N. Ahmed, E. Frew, D. Lawrence, and B. Argrow, "Machine self-confidence in autonomous systems via meta-analysis of decision processes," in *Proc. Int. Conf. Appl. Hum. Factors and Ergonom.*, Springer, 2019, pp. 213–223.
- [24] B. W. Israelsen, "Algorithmic assurances and self-assessment of competency boundaries in autonomous systems," Ph.D. dissertation, Univ. Colorado, Boulder, USA, 2019.
- [25] A. Gautam, T. Whiting, X. Cao, M. A. Goodrich, and J. W. Crandall, "A method for designing autonomous robots that know their limits," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2022, pp. 121–127.
- [26] C. Béné and L. Doyen, "From resistance to transformation: A generic metric of resilience through viability," *Earth's Future*, vol. 6, no. 7, pp. 979–996, 2018.
- [27] S. Banerjee and S. Chernova, "Fault diagnosis in robot task execution," in *Proc. AAAI Spring Symp. Ser.*, 2019, pp. 1–8.
- [28] B. Dhillon, A. Fashandi, and K. Liu, "Robot systems reliability and safety: A review," *J. Qual. Maintenance Eng.*, vol. 8, no. 3, pp. 170–212, 2002.
- [29] A. Kampa, "The review of reliability factors related to industrial robots," *Robot. Automat. Eng. J.*, vol. 3, pp. 555–624, 2018.
- [30] G. Golda, A. Kampa, and I. Paprocka, "Analysis of human operators and industrial robots performance and reliability," *Manage. Prod. Eng. Rev.*, vol. 9, pp. 24–33, 2018.
- [31] S. Gürel, E. Körpeoğlu, and M. S. Aktürk, "An anticipative scheduling approach with controllable processing times," *Comput. Operations Res.*, vol. 37, no. 6, pp. 1002–1013, 2010.
- [32] I. Paprocka, W. M. Kempa, K. Kalinowski, and C. Grabowik, "A production scheduling model with maintenance," in *Advanced Materials Research*, vol. 1036. Stafa-Zurich, Switzerland: Trans. Tech. Publ., 2014, pp. 885–890.
- [33] W. M. Kempa, I. Paprocka, K. Kalinowski, and C. Grabowik, "Estimation of reliability characteristics in a production scheduling model with failures and time-changing parameters described by gamma and exponential distributions," in *Advanced Materials Research*, vol. 837. Stafa-Zurich, Switzerland: Trans. Tech. Publ., 2014, pp. 116–121.
- [34] M. B. Van Riemsdijk, M. Dastani, and M. Winikoff, "Goals in agent systems: A unifying framework," in *Proc. 7th Int. Joint Conf. Auton. Agents Multiagent Syst.*, 2008, vol. 2, pp. 702–709.
- [35] B. W. Israelsen and N. R. Ahmed, "'dave... I can assure you... that it's going to be all right...' a definition, case for, and survey of algorithmic assurances in human-autonomy trust relationships," *ACM Comput. Surv.*, vol. 51, no. 6, 2019, Art. no. 113.
- [36] A. R. Hutchins, M. Cummings, M. Draper, and T. Hughes, "Representing autonomous systems' self-confidence through competency boundaries," in *Proc. Hum. Factors Ergonom. Soc. Annu. Meeting*, Los Angeles, CA, USA: SAGE Publications, 2015, vol. 59, pp. 279–283.
- [37] U. Kuter and C. Miller, "Computational mechanisms to support reporting of self confidence of automated/autonomous systems," in *Proc. AAAI Fall Symp. Ser.*, 2015, pp. 18–21.
- [38] N. Sweet, N. R. Ahmed, U. Kuter, and C. Miller, "Towards self-confidence in autonomous systems," in *Proc. AIAA Infotech, Aerosp.*, 2016, Art. no. 1651.
- [39] A. Zagorecki, M. Kozniowski, and M. Druzdzel, "An approximation of surprise index as a measure of confidence," in *Proc. AAAI Fall Symp. Ser.*, 2015, pp. 39–41.
- [40] J. Habbema, "Models for diagnosis and detection of combinations of diseases," in *Decision Making and Medical Care*, F. T. De Dombal and F. Gremy, Eds. New York, NY, USA: North-Holland Publishing Company, 1976, pp. 399–411.
- [41] 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.
- [42] N. Conlon et al., "Generalizing competency self-assessment for autonomous vehicles using deep reinforcement learning," in *Proc. AIAA SCITECH Forum*, 2022, Art. no. 2496.
- [43] A. Norton et al., "Metrics for robot proficiency self-assessment and communication of proficiency in human-robot teams," *J. Hum.-Robot Interact.*, vol. 11, no. 3, pp. 1–38, 2022.
- [44] T. Fitzgerald, E. Short, A. Goel, and A. Thomaz, "Human-guided trajectory adaptation for tool transfer," in *Proc. 18th Int. Conf. Auton. Agents MultiAgent Syst.*, 2019, pp. 1350–1358.
- [45] S. M. Fleming and N. D. Daw, "Self-evaluation of decision-making: A general Bayesian framework for metacognitive computation," *Psychol. Rev.*, vol. 124, no. 1, pp. 91–114, 2017.
- [46] D. Schreckenghost, T. Milam, and T. Fong, "Measuring performance in real time during remote human-robot operations with adjustable autonomy," *IEEE Intell. Syst.*, vol. 25, no. 05, pp. 36–45, Sep./Oct. 2010.
- [47] D. L. Schreckenghost, T. Milam, and T. Fong, "Techniques and tools for summarizing performance of robots operating remotely," in *Proc. 14th Int. Conf. Space Operations*, 2016, Art. no. 2310.
- [48] P. M. Djuric and P. Closas, "On self-assessment of proficiency of autonomous systems," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2019, pp. 5072–5076.
- [49] C. Schneider, A. Barker, and S. Dobson, "Evaluating unsupervised fault detection in self-healing systems using stochastic primitives," *EAI Endorsed Trans. Self-Adaptive Syst.*, vol. 15, no. 1, pp. 1–15, 2015.
- [50] H. Huang, E. Messina, and J. Albus, "Autonomy level specification for intelligent autonomous vehicles: Interim progress report," in *Proc. Perform. Metrics Intell. Syst. Workshop*, Gathersburg, MA, USA, 2003, pp. 16–18.

- [51] J. W. Crandall, M. A. Goodrich, D. R. Olsen Jr, and C. W. Nielsen, "Validating human-robot interaction schemes in multi-tasking environments," *IEEE Trans. Syst., Man, Cybern.*, vol. 35, no. 4, pp. 438–449, Jul. 2005.
- [52] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, Apr. 1997.
- [53] L. S. Fletcher et al., *The MIT–Cornell Collision and Why It Happened*. Berlin, Germany: Springer, 2009, pp. 509–548.
- [54] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," in *Proc. Robot. Sci. Syst.*, 2010, pp. 34–41.
- [55] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Hoboken, NJ, USA: Prentice Hall Press, 2009.
- [56] X. Cao, "Computing agent competency in first order Markov processes," Master's thesis, Brigham Young Univ., Provo, USA 2021.
- [57] "Rethink robotics," 2019. [Online]. Available: <https://www.rethinkrobotics.com/sawyer>
- [58] T. Wiedemeyer, "IAI Kinect2," 2014–2015, Accessed: Jun. 12, 2015. [Online]. Available: https://github.com/code-iai/iai_kinect2
- [59] G. Bradski, "The OpenCV library," *Dr Dobbs's J. Softw. Tools*, vol. 25, no. 11, pp. 120–123, 2000.
- [60] I. A. Sucas and S. Chitta, "MoveIt motion planning framework," 2013. Accessed: Nov. 24, 2020. [Online]. Available: <http://moveit.ros.org>
- [61] D. M. Rosen, L. Carlone, A. S. Bandeira, and J. J. Leonard, "SE-Sync: A certifiably correct algorithm for synchronization over the special euclidean group," *Int. J. Robot. Res.*, vol. 38, no. 2–3, pp. 95–125, 2019.
- [62] N. Akai, L. Y. Morales, T. Hirayama, and H. Murase, "Misalignment recognition using Markov random fields with fully connected latent variables for detecting localization failures," *IEEE Robot. Automat. Lett.*, vol. 4, no. 4, pp. 3955–3962, Oct. 2019.
- [63] D. Adolfsson, M. Castellano-Quero, M. Magnusson, A. J. Lilienthal, and H. Andreasson, "CorAL: Introspection for robust radar and lidar perception in diverse environments using differential entropy," *Robot. Auton. Syst.*, vol. 155, 2022, Art. no. 104136.
- [64] T. Chakraborti, S. Sreedharan, and S. Kambhampati, "The emerging landscape of explainable automated planning and decision-making," in *Proc. 29th Int. Joint Conf. Artif. Intell.*, 2021, pp. 4803–4811.
- [65] G. E. Mullins, P. G. Stankiewicz, R. C. Hawthorne, and S. K. Gupta, "Adaptive generation of challenging scenarios for testing and evaluation of autonomous vehicles," *J. Syst. Softw.*, vol. 137, pp. 197–215, 2018.



Tim Whiting is working toward the Ph.D. degree in computer science with Brigham Young University, Provo, UT, USA.

His interests range from programming language design and reasoning to intelligent algorithm design and user experience. He is especially interested in how programmers reason about their code and how their mental models fit with their intentions. He cares about making code resilient and robust to real world environments and about using good abstractions to make code reusable, modular, and efficient.



Skyler Smith is working toward the computer science degree with an emphasis in software engineering.

He is currently a Software Engineer Intern with Proofpoint, Sunnyvale, CA, USA. He is a senior at Brigham Young University, Provo, UT, USA. His research interests include computer systems design and software architecture.



Michael A. Goodrich (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical and computer engineering from Brigham Young University, Provo, UT, USA, in 1992, 1995, and 1996, respectively.

He is currently a Professor of computer science with Computer Science Department, Brigham Young University, where he directs the Human-Centered Machine Intelligence Lab. His research interests include human–robot interaction, multiagent systems, and artificial intelligence.



Xuan Cao received the B.S. degree in energy and environmental engineering and the Ph.D. degree in engineering thermophysics from Zhejiang University, Hangzhou, China, in 2013 and 2018, respectively. He is currently working toward the second Ph.D. degree in computer science with Brigham Young University, Provo, UT, USA.

His research focuses on robot proficiency self-assessment. He is also interested in machine learning and artificial intelligence.



Jacob W. Crandall received the B.S., M.S., and Ph.D. degrees in computer science from Brigham Young University, Provo, UT, USA, in 2001, 2003, and 2006, respectively.

He is currently a Professor with Computer Science Department, Brigham Young University, where he directs the Laboratory for Interactive Machines. His research interests include human–machine cooperation, multiagent systems, robotics, and artificial intelligence.



Alviqa Gautam received the Ph.D. degree in computer science from IIIT-Delhi, New Delhi, India, in 2019.

She is a Senior Research Engineer with TEES, Texas A&M University, College Station, TX, USA. Her research interests include risk and pedestrian intent-aware autonomous vehicles path planning and efficient communication for safe operations. Although her work is grounded in unmanned autonomous vehicles, it also interfaces with topics across human–robot interaction and cooperation.

Prior to Texas AM, her research as a postdoctoral scholar at BYU, Provo, UT, USA, focused on enabling dexterous robots to self-assess, and communicate their proficiency at a task, and designing machines that can effectively cooperate with people.