Dynamic Simplex: Balancing Safety and Performance in Autonomous Cyber Physical Systems

Baiting Luo
Vanderbilt University
USA
baiting.luo@vanderbilt.edu

Shreyas Ramakrishna
Vanderbilt University
USA
shreyas.ramakrishna@vanderbilt.edu

Ava Pettet
Vanderbilt University
USA
ava.pettet@vanderbilt.edu

Christopher Kuhn
Technical University of Munich
Germany
christopher.kuhn@tum.de

Gabor Karsai
Vanderbilt University
USA
gabor.karsai@vanderbilt.edu

Ayan Mukhopadhyay
Vanderbilt University
USA
ayan.mukhopadhyay@vanderbilt.edu

ABSTRACT

Learning Enabled Components (LEC) have greatly assisted cyber-physical systems in achieving higher levels of autonomy. However, LEC's susceptibility to dynamic and uncertain operating conditions is a critical challenge for the safety of these systems. Redundant controller architectures have been widely adopted for safety assurance in such contexts. These architectures augment LEC "performant" controllers that are difficult to verify with "safety" controllers and the decision logic to switch between them. While these architectures ensure safety, we point out two limitations. First, they are trained offline to learn a conservative policy of *always* selecting a controller that maintains the system's safety, which limits the system's adaptability to dynamic and non-stationary environments. Second, they do not support reverse switching from the safety controller to the performant controller, even when the threat to safety is no longer present. To address these limitations, we propose a dynamic simplex strategy with an online controller switching logic that allows two-way switching. We consider switching as a sequential decision-making problem and model it as a semi-Markov decision process. We leverage a combination of a myopic selector using surrogate models (for the forward switch) and a non-myopic planner (for the reverse switch) to balance safety and performance. We evaluate this approach using an autonomous vehicle case study in the CARLA simulator using different driving conditions, locations, and component failures. We show that the proposed approach results in fewer collisions and higher performance than state-of-the-art alternatives.

ACM Reference Format:
Baiting Luo, Shreyas Ramakrishna, Ava Pettet, Christopher Kuhn, Gabor Karsai, and Ayan Mukhopadhyay. 2023. Dynamic Simplex: Balancing Safety and Performance in Autonomous Cyber Physical Systems. In *ACM/IEEE 14th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2022) (ICCPS '23), May 9–12, 2023, San Antonio, TX, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3576841.3585934

ICCPS '23, May 9–12, 2023, San Antonio, TX, USA

ACM ISBN 979-8-4007-0036-1/23/05…$15.00
https://doi.org/10.1145/3576841.3585934

1 INTRODUCTION

Autonomous cyber-physical systems (CPS) are an important component of many applications in the fields of medicine, aviation, and the automotive industry. Such systems are often equipped with LEC that are trained using machine learning (ML) methods [9]. A critical challenge for these systems is making safe and efficient decisions under unanticipated system faults and dynamically changing operating conditions [34]. However, recent incidents involving autonomous vehicles (AVs) from automotive companies such as Waymo, Tesla, Uber, and Cruise [33] illustrate the complexity of this decision-making process. Indeed, the National Highway Traffic Safety Administration (NHTSA) released a summary report that highlighted 392 crashes involving AVs in the United States between June 2021 and May 2022 [24].

A common mechanism for dealing with failures and ensuring safety, especially in CPS, is the usage of *controller-redundant* architectures, e.g., the simplex architecture [31] and controller sandboxing [2]. Such architectures typically augment CPS that use a high-performing but unverifiable controller (the *performant* controller) with a verified controller (the *safety* controller) [1]. A decision logic, often uses a verification-based approach trained with a safety-based utility function or a simple set of domain rules, triggers a switch from the performant controller to the safety controller under unsafe operating conditions or system faults. Verification-based approaches like linear matrix inequality [32], reachability analysis [2], and safety certificates [29] have also been explored for this decision logic. Such techniques have been widely (and successfully) used in practice, e.g., unmanned aerial vehicles [35], remote-controlled cars [8], and industrial infrastructures [22].

While these approaches have shown promising results, there are two major limitations. First, the decision logic is generally trained offline. While offline training provides the advantage of invoking the policy almost instantaneously when making decisions, such policies can often become stale in non-stationary and dynamic conditions [15, 25]. Second, the decision logic is usually designed to *only perform a one-way switch*, i.e., when the system under consideration detects an imminent threat to safety, the logic dictates a switch from the performant controller to the safety controller. Once such a switch is made, the control remains in the safety mode

forever (barring some exceptions that perform the reverse switch based on system stability [10, 18]). However, once the threat no longer exists, using the safety controller could delay or ignore the system's mission-critical objectives [18].

However, the reverse switch , i.e., transitioning back to the performant controller from the safety controller, is highly non-trivial for several reasons. First, in real-world CPS, safety is paramount, and a myopic switch could be detrimental to the overall health of the system and related entities, including humans. As a result, it is imperative that a careful and non-myopic evaluation is done on the evolution of the system and possible exogenous factors before switching back to the performant controller. Second, these exogenous factors could be dynamic [26]; such variation makes it necessary that the logic is equipped to perform planning with the most up-to-date information available at hand (e.g., through online planning). However, online approaches are slow (compared to their offline counterparts), which inhibits their usage in practice. Third, although the reverse switch is crucial for improving the system's performance, frequent back-and-forth switching among the controllers can be detrimental to stability and performance. While these factors make designing the reverse switching logic challenging, one advantage compared to the forward switching logic is that the system's safety is not sensitive to the computation time, allowing us to explore non-myopic, principled, albeit more computationally intensive algorithmic methods.

In this paper, we present a principled hybrid approach to address the challenges of balancing the safety and performance objectives in autonomous CPS. We make the following contributions: **1)** We present the dynamic simplex strategy (DS) that is online and allows for two-way switching while avoiding frequent back-and-forth arbitration. **2)** We formulate the decision-making problem as a semi-Markov decision process [17]. While we perform the forward switch (i.e., from the performant to the safety mode) myopically to prioritize safety, the decision for reverse switching to the performant controller is performed in a non-myopic manner to find a promising action by using Monte Carlo tree search (MCTS). We present a combination of online heuristic search and domain-based safety rules for switching. **3)** To monitor the need for switching, we use a set of runtime data-driven safety monitors that collectively indicate the system's imminent risk by keeping track of system faults and uncertainties in the operating environment. **4)** We evaluate the proposed approach extensively through multiple autonomous vehicle studies in simulated urban environments using the CARLA simulator [11] and demonstrate that the proposed approach leads to fewer infractions and higher performance than state-of-the-art alternatives.

2 PROBLEM SETUP AND MODEL

2.1 Problem Setup

Consider an autonomous CPS with both performance and safety objectives. We use the example of an autonomous vehicle for this paper. The goal of the system designer is to enable control logic that determines operational parameters such as speed and steering angle. Instead of directly *acting* on the parameters, such systems are equipped with controller(s) that affect operational parameters. The decision-maker must therefore select a controller, which in turn, selects the parameters. Typically, a performant controller, C^p, is used to ensure that the autonomous CPS focuses on its performance objectives (e.g., minimize the travel time) [12]. Performant controllers are often designed using ML-based approaches and trained on data that closely mimics the operating conditions. Such data typically involves information from sensors like cameras, radar, and lidar to compute high-level trajectories or low-level control actions for the system. Formally, we denote a data point representing an operating condition as a *scene*. For example, a scene could be a collection of scalar values denoting precipitation and the location of the vehicle over a few seconds. We assume that each scene is associated with a set of features $w \in \mathbb{R}^m$ that includes structural (spatial) features w^s (e.g., the type of road, road curvature, and the presence of road signs) and temporal features w^q (e.g., weather conditions) characterizing the operating conditions.[1] Typically, the performant controller is trained on data from a large number of such scenes.

In trying to achieve the system's performance objectives, the performant controller may neglect the safety objectives [18]. As a result, these systems are also equipped with several runtime monitors, a safety controller C^s, and a decision logic for safety assurance. The monitors raise alarms based on identifying different operational hazards (e.g., out-of-distribution (OOD) data for LEC) and system hazards (e.g., sensor failure), which can be either critical or non-critical. When a hazard is detected, the decision logic switches the system's control from the performant controller to the safety controller, which focuses exclusively on ensuring safety, e.g., the safety controller might reduce the system's speed, intervene through braking, or alert the driver for manual intervention. Given this setting, our goal is to design an approach that balances the safety and performance objectives of the CPS.

2.2 Problem Formulation

We refer to the autonomous CPS and the environmental conditions (i.e., w) as our *system* of interest. We begin with the assumption that the decision-maker knows the spatial features w^s *a priori* for the finite set of scenes the vehicle will travel through. We assume that the future temporal features w^q are unknown to the vehicle. In practice, information pertaining to the spatial features such as the curvature of the road can be retrieved easily. Note that this assumption is not critical (or important) for our formulation or solution approach; it is merely based on our domain of interest.

We consider the evolution of the system in continuous time. The dynamics of decision-making are governed by the following events: (a) when the spatial parameters w^s change (e.g., the vehicle enters a new stretch on the road with curvature), (b) when the temporal parameters w^q change (e.g., the weather changes), (c) when a component of the system fails, (d) when the traffic density changes, or (e) when the runtime monitor state changes. When an event occurs, the decision-maker must take an action, i.e., choose between operating in the performant mode or the safety mode. Note that the time between the events is governed by some exogenous distribution that is not necessarily memoryless; for example, the change of the spatial parameters depends on the speed of the car. To

[1] while we define features for autonomous vehicles, such attributes can capture arbitrary operating conditions relevant to any autonomous CPS.

capture the non-memoryless transitions and the continuous-time evolution of the system, we model our decision-making problem as a semi-Markov decision process (SMDP) [17].

An SMDP can be represented by a tuple $\{\mathcal{S}, \mathcal{A}, T, R, \tau\}$, where $\mathcal{S}$ is a finite set of states, A is a finite set of actions that can be performed in a state, T is the state-action transition model, R is the reward function, and τ is the temporal distribution over state transitions. We describe each component of the SMDP below.

State We denote the finite set of states by $\mathcal{S}$. Formally, we represent the state $s_t \in \mathcal{S}$ by the tuple $(v_t, w_t^s, w_t^q, d_t, C_t, \Phi_t, \Psi_t, \omega_t)$, where v_t is the velocity of the vehicle, w_t^s and w_t^q are the structural and temporal scene features, d_t is the traffic density, $C_t \in \{C^p, C^s\}$ is the controller driving the system, $\Phi_t = \{\phi_t^1, \ldots, \phi_t^n\}$ is the failure state of the n components (e.g., sensors), Ψ_t is the runtime monitor state (e.g., OOD detector), and ω_t is a counter that keeps track of the number of switches that have been performed until the current time, with all of the variables being observed at time t. We assume that $\phi_t^i \in \{0, 1\}\ \forall i, t$.

Actions We denote the set of all actions by $\mathcal{A}$. An action in our setting is restricted to selecting a controller. In this paper, we restrict our attention to two controllers—a safety controller and a performant controller. Our action space is therefore simplified to the binary choice of whether or not to switch the controllers. In principle, our problem formulation (and the solution approach) can accommodate an arbitrary number of controllers as part of the action space.

Transitions: The evolution of our system model is governed by several stochastic processes. First, the spatial parameter is governed by the track on which the system operates (known *a priori*) and the system's speed, which in turn is a function of the controller in use. Second, the weather conditions, traffic density, sensor failures and runtime monitor states are governed by exogenous distributions. As our solution approach is based on exploring possible trajectories under the effect of the actions, we only need access to a set of generative models for simulating the transitions [26]. We describe the specific models we use for such parameters in the evaluation section. The only deterministic update to a state under an action is that of the counter ω, which is incremented by 1 every time the decision logic performs a switch between the controllers.

Reward Function: Rewards in an SMDP consist of a lump sum immediate reward upon taking an action and/or a continuous-time reward as the system evolves [17]. We model reward as the sum of immediate rewards that capture both performance and safety objectives. Formally, the reward for an action $a \in \mathcal{A}$ in state $s_t \in \mathcal{S}$ includes a performance score $\lambda^p(s_t, a)$ and a safety score $\lambda^f(s_t, a)$. In our implementation, we model λ^p as the chosen controller's (i.e., the action a's) average speed and λ^f as the controller's likelihood of collision given the state s_t. We estimate both terms by using surrogate models trained through historical data (we describe the exact estimation process in Section 3.1). The instantaneous reward is calculated as the weighted sum of the two scores:

$$R(s_t, a) = \alpha_1 \cdot \lambda^p(s_t, a) - \alpha_2 \cdot \lambda^f(s_t, a) \tag{1}$$

where α_1 and α_2 are hyperparameters.

While such a function is sufficient to capture safety and performance objectives, practical constraints require that we prevent frequent back-and-forth switching among the controllers. Therefore, we include a third term called cost of switching λ^c that adds a penalty based on the number of previously performed switches leading up to the current state. For an arbitrary state s_t, we use ω_t to compute this penalty (recall that ω_t tracks the number of controller switches). Specifically, $\lambda^c = 0$, if the ω_t variable is 0 or 1; otherwise $\lambda^c = \omega_t / m_s$, in which m_s is the maximum number of switches that can happen during the planning horizon (i.e. future scenes considered during panning). However, we point out that the cost of switching frequently should not be used to limit the forward switch (from the performant to the safety mode) as it can compromise the safety of the autonomous CPS. Therefore, we calculate the reward for the forward switch according to Eq. (1) and use the penalty term only for computing rewards for the reverse switch as shown below:

$$R(s_t, a) = \alpha_1 \cdot \lambda^p(s_t, a) - \alpha_2 \cdot \lambda^f(s_t, a) - \alpha_3 \cdot \lambda^c(s_t, a) \tag{2}$$

Based on the above SMDP, given a state, our goal is to choose actions based on a utility function (e.g., expected discounted reward). We describe the exact criteria and our approach below.

3 DYNAMIC SIMPLEX STRATEGY

A schematic diagram of our approach is shown in Fig. 1. We switch between the controllers based on the following criteria: for the forward switch (i.e., from the performant to the safety controller), we take the action that maximizes the myopic one-step reward (based on Eq. (1)), which ensures that any imminent threats to safety are thwarted based on historical data. Furthermore, our method assumes some safety verification protocols will be given to provide conditions for switching into the safety controller (e.g., conditions provided by the proof of safety for the safety controller) when it is applied to real-world applications. Naturally, the decision-maker cannot be entirely myopic about action selection; the performance score λ^p and the safety score λ^f capture some non-myopic effects of taking an action by leveraging a surrogate model trained using historical data. However, note that the reverse switch occurs *after* the control logic had previously decided to switch to the safety controller; this decision must have resulted from an imminent threat to the safety of the CPS. Hence, to ensure that the CPS can safely switch back to the performant mode, we do non-myopic planning and take the action that maximizes the expected discounted cumulative reward. These criteria essentially form the core of our dynamic simplex strategy. To actuate the strategy, we use the following components: 1) a myopic action selector that uses given safety verification protocols and a complementary neural network-based surrogate model trained with historical data and ; 2) a non-myopic planner based on an approximate heuristic search algorithm to perform the reverse switch; and 3) a set of runtime monitors to monitor changes in environmental parameters and sensor faults.

We describe the proposed strategy briefly here. When an event occurs, the control logic first checks which controller is driving the system. If the performant controller is operating the system, the switcher activates the myopic action selector based on the trained surrogate model (or a verification protocol) to get the performance and safety scores of both controllers. No switching is performed if the performant controller has a higher reward. Otherwise, a forward switch is performed. Next, if the safety controller is operating,

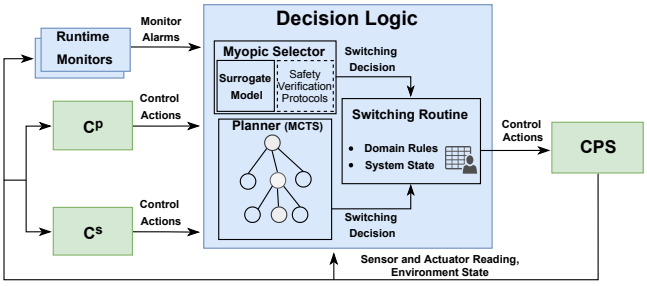

Figure 1: Overview of the proposed dynamic simplex strategy. The blocks in blue are designed through the solution approach. The green blocks represent generic autonomous CPS components and controllers.

the switcher activates the planner, which uses a set of generative models to simulate possible future trajectories and picks an action that maximizes the expected discounted cumulative reward. To balance exploration and exploitation in the future trajectories, we use MCTS. In our setting, if a new event occurs while a decision is being computed, the computation is immediately terminated, and decision-making is initiated with the new state. We describe each decision-making component below.

3.1 Myopic Action Selector

The role of the myopic action selector is to decide whether to switch from the performant controller to the safety controller, i.e., the forward switch. A number of varied works have proposed different potential methods to trigger the decision logic, e.g., safety verification methods such as reachability analysis, which computes a set of states reachable within some number of time steps and then checks if this reachable set contains states outside the system's safe region [28]; and OOD detection methods, which detect the data that are not similar to the data used for training [36].

However, the existing safety verification methods are subject to limited efficiency and accuracy for safety-critical systems that are time-sensitive and operate in dynamic environments [20]. Further, computing the exact reachable set for most nonlinear systems is a complex problem [14], and deployment of LEC aggravates the complexity of the problem. However, safety verification methods for neural network control systems are important components to ensure the safety of complex CPS. The myopic selector we propose (Fig. 1) can easily accommodate outputs of safety verification protocols and complement arbitrary surrogate models into decision-making procedure. This integration is orthogonal to the main contribution we make; in this paper, we show how a computationally cheap and safety-focused forward switch can be complemented with a data-driven non-myopic reverse switch to balance safety and performance.

More specifically, the goal of the model used for the forward switch is to take as input the current state s_t and return an aggregated score that captures the immediate threat to the safety of the system, coupled with some performance objective. In this work, the score is composed of the collision likelihood of both the controllers and a measure of performance based on normalized average speed (see Eq. (1)). Specifically, we train a deep neural network (DNN) using historical data of the performant and safety controllers pertaining to operations in similar environments to estimate these parameters to show the proposed technique works well by only using a surrogate model. This data is curated from a large number of prior simulations, and the simulated observations are stored in a tabular format (we describe the exact curation process in Section 4.1). Furthermore, the safety of the system can be further guaranteed as long as a safety verification protocol is given. Finally, the myopic action selector chooses the best action based on Eq. (1).

3.2 Non-Myopic Planner

The decision to switch from the safety controller to the performant controller is more involved than the forward switch, and has not been addressed in prior work, partly due to its complexity. Given a state s_t such that $C_t = C^s$, we use MCTS to evaluate whether or not to switch the controller to C^p. MCTS is a heuristic search algorithm for sequential decision-making [3]. It enables online planning in a way that can incorporate changes in environmental parameters at decision-time; also, it is an anytime algorithm, which enables the system-designer to constrain the computation time based on domain-specific requirements.

MCTS operates by iteratively building a search tree that represents future trajectories. The tree nodes represent the states, and the edges represent the actions that mark the state transitions. The search begins with a root node that denotes the current state. In each simulation, a child node is recursively selected until a leaf node is reached. Unless this leaf node denotes the end of the planning horizon, an action is taken in this state node, and the tree is expanded. To estimate the value of an action from a node, the algorithm simulates a "rollout" from the child node to the end of the planning horizon with a computationally cheap default policy. Using the algorithm requires three components: (1) generative models to simulate the future states, (2) a tree policy to navigate the tree search, and (3) a default policy to estimate the value of an action. We describe these components below.

Generative Models: We use a set of generative models to sample future trajectories. First, we use a model for sampling future weather parameters, conditioned on the current weather. We learn the weather model using the historical data gathered from simulations. Second, we use a model for sampling average traffic density conditioned on current traffic density, also learned based on simulated data. Third, we model sensor failures by gathering historical data about the sensors and their susceptibility to adverse weather conditions and lighting levels. While some of these failures (e.g., bright images) are rectified over time, other failures (e.g., broken lens) persist until replacement. Fourth, we model the runtime monitor alarms by learning the duration and the arrival time of such states using historical data. Finally, we use the surrogate model and any given safety verification protocol discussed in the previous section to identify the safety score and the performance score of the controllers. While we use different distributions and neural networks for learning the generative models, the proposed framework is agnostic to the model used. For the sake of brevity, we present a detailed description of the generative models in Appendix A.3.

Tree Policy: We perform action selection within the tree using the standard Upper Confidence bound for Trees (UCT) algorithm [19].

Default Policy: To simulate rollouts, it is common to use a computationally cheap default policy [26]. Our default policy is random, i.e., we randomly select between making a switch or staying with the current controller in use.

Tree Search: Each search begins by initializing the current state as the tree's root node. The core idea behind the algorithm is that the search tree over possible future trajectories is explored asymmetrically and iteratively, with the search being biased toward promising action trajectories. During each simulation, the tree policy (explained above) is used to select which leaf node is expanded, after which the default rollout policy quickly estimates the new node's value.

State Transitions: Here, we describe how we perform the state transitions inside the tree. Although the temporal parameters (e.g., weather) evolve in continuous time, we discretize time and assume that such parameters are queried every τ^q units of time, e.g., our system can query the current weather every 20 seconds. In principle, each parameter can be queried at different frequencies depending on the domain of interest; in such a case, let τ^q denote the discrete time period after which the parameter with the highest frequency is queried. Each state transition caused by other events will evolve t_q towards τ^q, which denotes the transition of temporal parameters in our case. Once new temporal parameters are sampled, t_q is reset to τ^q.

At a given state s_t (at time t), the state of the system consists of the location, which can be defined by the structural parameters of the scene (i.e., w_t^s) and the velocity of the vehicle v_t. Depending on the action taken, v_t can change. It is then trivial to compute the time taken by the vehicle to the next location (i.e., structural component of scene) as we assume that the structural features are known *a priori* (see Section 2). We denote this arrival time by t_e. The time t_e can then be compared with τ^q to populate the structural features of the next state, i.e., determine if the vehicle has moved to the next location. The other temporal parameters, the future average traffic density d across a scene, the arrival time of sensor failures, and the arrival time of the next runtime monitor state are sampled according to the generative models. The earliest arriving event leads to the transition of the state. Finally, we compute the reward for a state-action pair by querying a surrogate model (denoted by G), which is also used to retrieve the velocity of the vehicle inside the tree (for future states given an action). We present the complete search algorithm algorithm in Algorithm 1.

Algorithm 1: Monte Carlo Tree Search (MCTS)

Input: current state s_t, generative models M, surrogate model G, query time τ^q, $t_q = \tau^q$, number of tree simulations I_n, parameter controlling tree exploration c_{uct}
Output: Policy π

function MCTS(s_t)
  initialize s_t as the root node
  for $m = 1, \cdots, I_n$ **do**
    Tree_Search(s_t)
  end for
  $\pi(a|s_t) \longleftarrow \frac{N(s_t,a)}{N(s_t)}$ ▷ switching policy
  return π
end function

function TREE_SEARCH(s_t) ▷ recursive tree search
  if s is terminal **then**
    $r \longleftarrow R(s_t, a)$
    return r
  else if s not visited **then**
    $r \longleftarrow rollout(s_t)$ ▷ rollout
    return r
  end if
  $a \longleftarrow \text{argmax}_{a \in A}\, UCB(s_t, a, c_{uct})$
  $\hat{v} \sim G(s_t, a)$ ▷ sample new velocity
  $\hat{t}_o \sim M(s_t, a)$ ▷ sample the duration of Ψ_t
  t_e = **distance**$(w_t^s, \hat{v})$ ▷ estimate the time to arrive in the next structural scene
  $\hat{t}_f \sim M(t, min(t_e, \hat{t}_o, t_q))$ ▷ sample if sensor failure will happen before any other event and its arrival time

  if $\hat{t}_f$ exist **then**
    $s^*_{t=t+\hat{t}_f} \sim M(s_t)$ ▷ Sample new Φ and d
    $t_q \longleftarrow t_q - \hat{t}_f$
  else if $\hat{t}_o < t_e$ and $\hat{t}_o < t_q$ **then**
    $s^*_{t=t+\hat{t}_o} \sim M(s_t)$ ▷ Sample new Ψ and d
    $t_q \longleftarrow t_q - \hat{t}_o$
  else if $t_e < t_q$ **then**
    $s^*_{t=t+t_e} \sim M(s_t)$ ▷ Sample new w^s and d
    $t_q \longleftarrow t_q - t_e$
  else if $t_e = t_q$ **then**
    $s^*_{t=t+t_q} \sim M(s_t)$ ▷ Sample new w^s, w^q, and d
    $t_q \longleftarrow \tau^q$
  else
    $s^*_{t=t+t_q} \sim M(s_t)$ ▷ Sample new w^q and d
    $t_q \longleftarrow \tau^q$
  end if
  save t_q
  $r \longleftarrow R(s_t, a) + \gamma Tree\_Search(s_t^*)$ ▷ perform tree search
  $N(s_t, a) \longleftarrow N(s_t, a) + 1$
  $Q(s_t, a) \longleftarrow Q(s_t, a) + \frac{r - Q(s_t,a)}{N(s_t,a)}$
  $N(s_t) \longleftarrow N(s_t) + 1$
  return r
end function

3.3 Runtime Monitors

To ensure the safety of the autonomous CPS, it is imperative that we keep track of the relevant state and environmental parameters. It is especially essential to track sensor faults which can lead to catastrophic accidents [13]. Several automotive standards, such as the ISO 26262 [16], categorize faults as: (a) permanent faults, which persist until removed or repaired (e.g., a broken camera), and (b) intermittent faults, which eventually go away on their own (e.g., occlusion). Both types of faults must be monitored and incorporated into decision-making.

In this paper, we deal with perception-based autonomous CPS. Therefore, we consider faults in the three cameras used in the autonomous vehicle in our case study (discussed in the experiments section). While many types of faults can be associated with a digital camera, we focus on the common fault of occlusion [5]. We train a monitor using prior data and only use inference on the trained models at decision time. Specifically, based on prior work for occlusion detection in AV [13], we train a model to detect continuous blobs of black image pixels. We mask an image to find connected black pixels in it and then color these pixels as white. Then, we calculate the percentage of white pixels in the image and use an exogenous threshold on the resulting value to detect occlusion. We describe the parameter values in the evaluation section 4.1.

We also implement the real-time OOD detector introduced by Cai and Koutsoukos [4] as one of our runtime monitors. We train a variational autoencoder (VAE) and utilize the reconstruction error for anomaly detection. Given an input, the trained decoder is used

to sample independent and identically distributed (IID) samples from the latent space; then, the reconstruction error is used as a nonconformity measure within inductive conformal anomaly detection (ICAD). Given a sample, if the p-value computed by ICAD is smaller than a threshold, this test sample is hypothesized to be an OOD example. Finally, the computed p-values are used to construct the martingale, which the stateful detector uses to classify an input as an OOD example.

3.4 Switching Routine and Domain Rules

Finally, we acknowledge that data-driven decision-making must be coupled with a switching routine that uses appropriate domain rules and the system's state information to ensure that the decision can be smoothly implemented. This is necessary because the performant and the safety controllers operate at different performance levels (both controllers have different achievable high speeds), and performing an instantaneous controller transition can impact the system's stability. For example, in case of a forward switch, the system might be operating at a higher speed with the performant controller, which the safety controller cannot handle. Therefore, instead of an instantaneous controller transition, we consider the system's physical state and domain-specific rules to determine a feasible transition. Note that a forward switch, i.e., switching to the safety mode, must be performed irrespective of other factors to ensure the system's safety. As a result, for the forward switch, we design the decision logic to trigger a change in the control action (e.g., decrease speed). However, in the case of the reverse switch, we design the decision logic to trigger a change in the control action as well as domain rules. Intuitively, we switch from the safety mode to the performant mode within specific areas of the feature space to ensure safe transitions. In our setting, we enable the switch on the main roads, overpasses, and freeways; however, we disable reverse switching on intersections, lane changes, and roundabouts. We present the ablation study of domain rules in Appendix A.2.

4 EXPERIMENTS

We evaluate the proposed dynamic simplex strategy on an AV example in CARLA simulation [11]. We create 10 tracks in two urban areas (towns) of the simulator. Fig. 7 in Appendix A.1 illustrates a snapshot of these tracks with different weather conditions. In total, we have 50 road segments with various structural scene features. We consider the road type, road curvature, presence of traffic signs, and traffic density as structural (spatial) features (w^s). We consider cloudiness, precipitation, and precipitation deposit parameters as temporal features (w^q). To vary weather, we randomly sample a change in the neighborhood of the current parameters to avoid sudden (drastic) fluctuations in weather.

4.1 Setup

AV setup: We show the system block diagram of the AV in Appendix A.1. It is primarily driven by a high-performant ML-based controller called as learning by cheating (LBC) [6], which uses a DNN for navigation. The controller uses six sensors, including three forward-looking cameras, an inertial measurement unit (IMU), a global positioning system (GPS), and a speedometer. For the safety controller, the system uses an autopilot controller, which is the safety controller in our setup. The controller uses the GPS, the speedometer, and the semantic segmentation camera sensor to compute the control action. We describe the exact operation of both the controllers in Appendix A.1. All experiments were run on a desktop computer with AMD Ryzen Threadripper 16-Core Processor, 4 NVIDIA Titan XP GPUs, and 128 GiB RAM.

Data for Surrogate Model: We run 111800 simulations using each controller in the 50 road segments across the CARLA towns, which amounts to 164 hours of driving, for generating data for the surrogate models. We randomly vary weather parameters, traffic density, and introduce camera faults (image blur and camera occlusion) for one or more of the available cameras in the simulations.

Baselines: We evaluate the performance of dynamic simplex strategy (DS) against the following baseline controller configurations: (1) LBC controller [6], (2) autopilot (AP) controller [11], and (3) traditional simplex architecture [31] with an offline decision logic for forward switching (*SA*). The forward switching is performed based on the controller's historical performances, and (4) traditional simplex architecture with reverse switching (*GS*). For simplex configurations, both forward and reverse switching are performed with a myopic policy that maximizes the one-step reward (based on Eq. (1)).

Runtime Monitors: We assume our system is equipped with runtime monitors to detect the changes in location and weather conditions (it is trivial to design such monitors in practice). For sensor failures, we design a monitor that can detect occlusion in the AV as described in Section 3. Specifically, if white pixels are larger than 10% (we set the threshold based on cross-validation), we consider an image to be occluded, otherwise, we mark the camera to not have occlusion. We obtain an F1-score of 98% on the validation images. For detecting OOD inputs, we implement a real-time OOD detector as described in Section 3. We use similar parameters as in prior work [4]; we use a VAE to generate 10 new examples for each given input, the martingale is then computed with the sequence of p-values computed by ICAD given the 10 new examples. The threshold for stateful detector is set to 100 to detect when the martingale becomes consistently large and if the input is an OOD example.

Hyperparameters: For parameters of MCTS, the tree depth is set according to the simulation duration needed to finish the next three structural scenes. We set the number of MCTS simulations per decision to 500, the exploration parameter c_{uct} to $\sqrt{2}$, and the discount factor γ to 0.9. Finally, for the reward calculations, we choose the weights $\alpha_1 = 1$, $\alpha_2 = 1$, and $\alpha_3 = 0.5$ based on manual tuning. For tuning the hyperparameter, we use a subset of the data for validation, fix α_2 at 1 (the weight on safety), and then vary the other parameters. We select the best weights based on a combination of travel times and safety score (we define these metrics below).

4.2 Results

We run each controller 30 times around each track. For each run, we start the initial scene with a random weather condition. We begin by evaluating the controllers without sensor failures. Then, we inject failures at random and evaluate their effects.

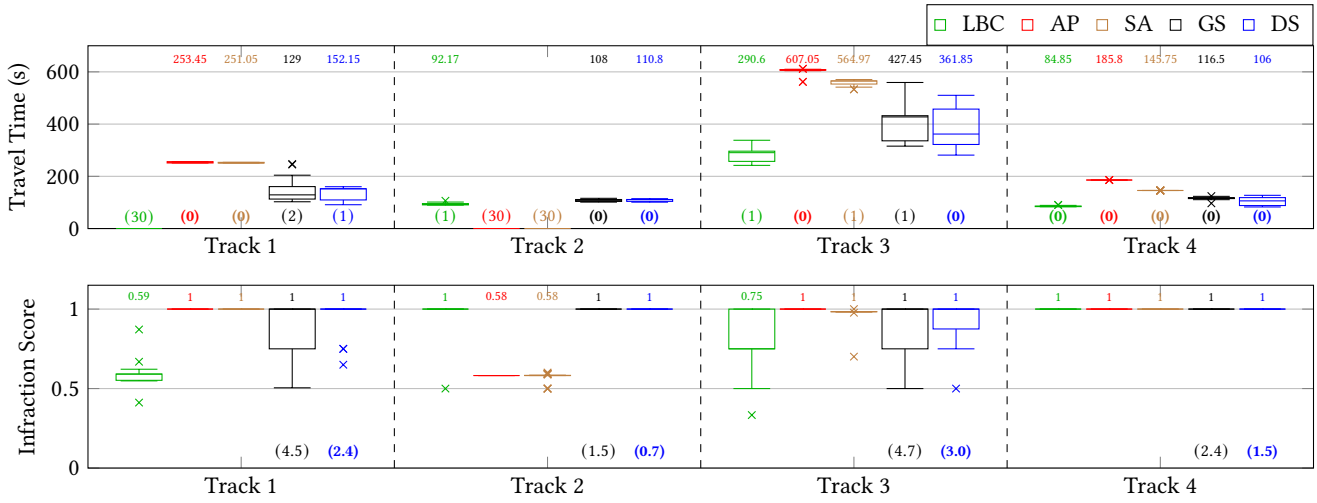

Figure 2: Top: Travel times of the different controller configurations across the 4 tracks (lower is better). We show the number of times a controller failed to complete a track in parenthesis below each box, highlighting the least failures in bold. We also show the median of the distributions at the top of the graph. We observe *DS* is the only configuration that provides competitive performance without sacrificing safety. (Bottom): We show the infraction score (higher is better) for all controllers across all tracks. We observe that *DS* consistently achieves a median score of 1, the highest possible safety score. We also show the mean of reverse switches performed by *GS* and *DS* across all tracks below each box. We observe that *DS* performs significantly fewer switches than *GS*.

AV operation with no sensor failure: We begin by evaluating the performance of the controller configurations across all the tracks *without any sensor failures*. As our objective is to maximize the system's performance while maintaining safety, we show results in terms of time taken to complete the track and the number of infractions (e.g., collisions). We begin by investigating the travel times taken by the controllers in Fig. 2 (top row).

We first observe that the LBC controller (the performant controller in isolation) has the best (i.e., lowest) travel times for all tracks aside from track 1. However, this performance comes at the cost of a highly unreliable safety performance (i.e., high variance in infraction scores, described below). The LBC also fails to complete track 1 even once in the 30 runs due to catastrophic failures. We observe that the AP controller can be erratic; while it (in isolation) often prioritizes safety resulting in track completion (with long travel times), it also leads to several failures, e.g., in track 2. We also observe that **the proposed approach (*DS*) results in the fewest infractions while achieving competitive travel times**. Note that longer travel times shown by *DS* are also a result of prioritizing safety, which we describe next.

In order to evaluate safety, we compute an *infraction score* as $0.5 \cdot RC + 0.25 \cdot col_v + 0.25 \cdot col_o$, where RC is the percentage of the route that was completed by the vehicle, and col_v and col_o denote the presence of collisions with other vehicles and other objects (e.g., a wall) respectively. We design the score such that a **higher score is better**, i.e., if any infraction is observed, the resulting parameter in the score is set to 0; otherwise, it is set to 1. We observe that the traditional simplex configuration *SA* is the safest among all the simplex configurations. We also observe that the *DS* configuration shows comparable performance with a median infraction score of 1.0 across all tracks. Analyzing the results on performance and safety, **we observe that the proposed *DS* outperforms other approaches in terms of balancing safety and performance**. For exmaple, in track 3, while LBC shows lower travel times and only fails to complete the track once, it also has a low median infraction score of 0.75, which indicates a significant number of collisions with the other vehicles or objects. Finally, we also observe that *DS* performs much fewer switches than *GS*, indicating our method can approach the nearly optimal time to switch to achieve performance objective without sacrificing safety.

AV operation with permanent sensor failure: To explicitly evaluate how the controllers perform under sensor failures, we randomly inject faults during our evaluation. Specifically, we simulate a center camera occlusion for the AV at random times and persist the failure once it occurs. We show the results in Fig. 3. We observe that such permanent camera occlusion severely affects the LBC controller, causing it to collide in all cases. On the other hand, the AP controller is unaffected by the occlusion. The proposed approach (*DS*) significantly outperforms *SA* and AP in travel time across all tracks but also achieves a median infraction score of 1 with low variance, thereby considerably improving the other simplex configurations. Note that *DS* has to perform slightly more reverse switches than *GS* to achieve performance objective without sacrificing infraction score on Track 2 and Track 4.

AV operation with intermittent sensor failure: As highlighted in the Section 4 in the main text, we also simulate intermittent failures from which an autonomous CPS can recover. For example, for an AV, an intermittent failure can be caused by high precipitation or strong light affecting the camera (e.g., sunlight). In our experiments, we use an exponential growth function to simulate the likelihood of occlusion conditioned on weather and location, i.e., the sunnier or heavier the precipitation is, it is more likely to cause a temporary occlusion. We also ensure that such failures are dependent on the structural features of the state, e.g.,

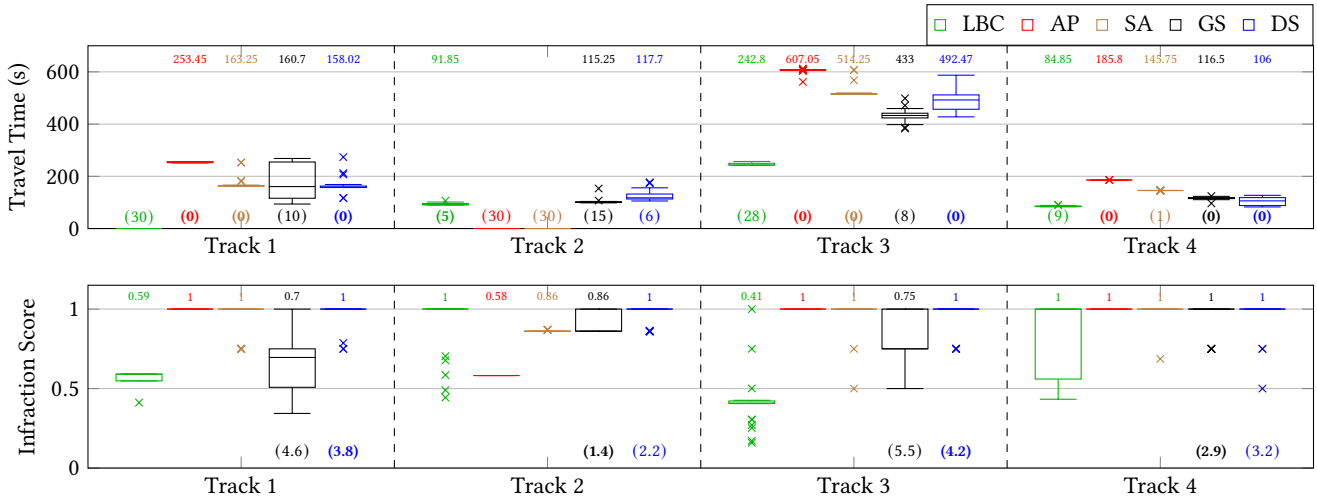

Figure 3: Top: Travel times for all controllers with permanent camera occlusion on all 4 tracks (lower is better). We show the number of times a controller failed to complete a track in parenthesis below each box, highlighting the least failures in bold. We also show the median of the distributions. Bottom: The infraction score (higher is better) for all controllers with permanent camera occlusion on all 4 tracks. We observe that DS consistently achieves a median score of 1, the highest possible safety score. We also show the mean of reverse switches performed by *GS* and *DS* across all tracks below each box.

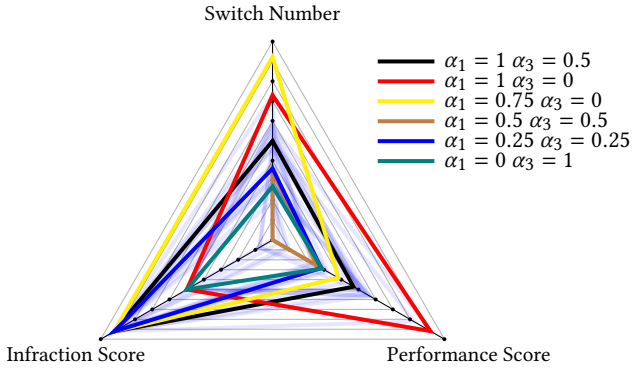

Figure 4: Sensitivity analysis radar chart. We collect data by fixing α_1 and α_2 and vary α_3 from 0 to 1; then, we fix α_2 and α_3 and vary α_1 from 0 to 1. The travel time (lower is better) is reformed as the performance score (higher is better) in the graph. For comprehensibility, we plot 6 out of 25 results with different colors and the rest of the other results in blue with low opacity. Generally, we do not consider tuning α_2 as it can harm the system's safety.

sunny conditions or precipitation is unlikely to cause occlusion if the vehicle is operating in a tunnel.

We present the results with intermittent sensor failure in Fig. 5. The intermittent failures do not affect the controllers as much as the permanent failures (Fig. 3). We observe that the *GS* controller fails to complete the tracks significantly more times than the other controllers. All the other four controllers have similar performance as they have in Fig. 2 with regard to the ability to complete the tracks, and the *DS* controller shows a lower median travel time while performing equally in terms of safety than *SA*. Though *GS* offers much lower median travel time than *DS* on Track2 and Track 4, it achieves so by significantly sacrificing the safety and ability to complete the tracks.

Sensitivity Analysis: Recall that in Eq. (1) and Eq. (2), α_1, α_2, and α_3 denote the weights to trade off the objectives of performance, safety, and avoiding frequent switching, respectively. We perform sensitivity analysis to analyze the effects of these weights by keeping α_2 fixed at 1 and varying α_1 and α_3. In Fig. 4, we show how the performance of the proposed *DS* controller without sensor failures is affected by the weights. We observe that increasing the value of α_3 from 0 to 1 gradually reduces the number of switches. We also observe that small values of α_1 generally lead to a lower performance score, which indicates longer travel times. Furthermore, we find that setting α_1 to 1 and α_3 to 0 results in the best performance score but comes with the sacrifice of infraction score, which implies the occurrence of failures or collisions. Finally, constraining the performance score and the number of switches by setting α_1 to 0 and α_3 to 1 (thereby penalizing switches heavily) sacrifices performance significantly, emphasizing the need of performing principled switching to balance safety and performance.

Computation Time: We present the detailed results of computation time in Appendix A.4. We observe that 500 MCST iterations give the best average travel time without harming the system's safety, with an average computation time of 0.94 seconds, which is feasible for the reverse switch.

5 RELATED WORK

The conventional decision logic used in simplex architectures is based mainly on verification techniques such as linear matrix inequality [32] and reachability analysis [2]. For example, Johnson, Taylor T *et al.* [18] present a real-time reachability algorithm that uses the offline LMI results with online reachability analysis. A zero-level set of barrier certificates [29] that separates the unsafe region

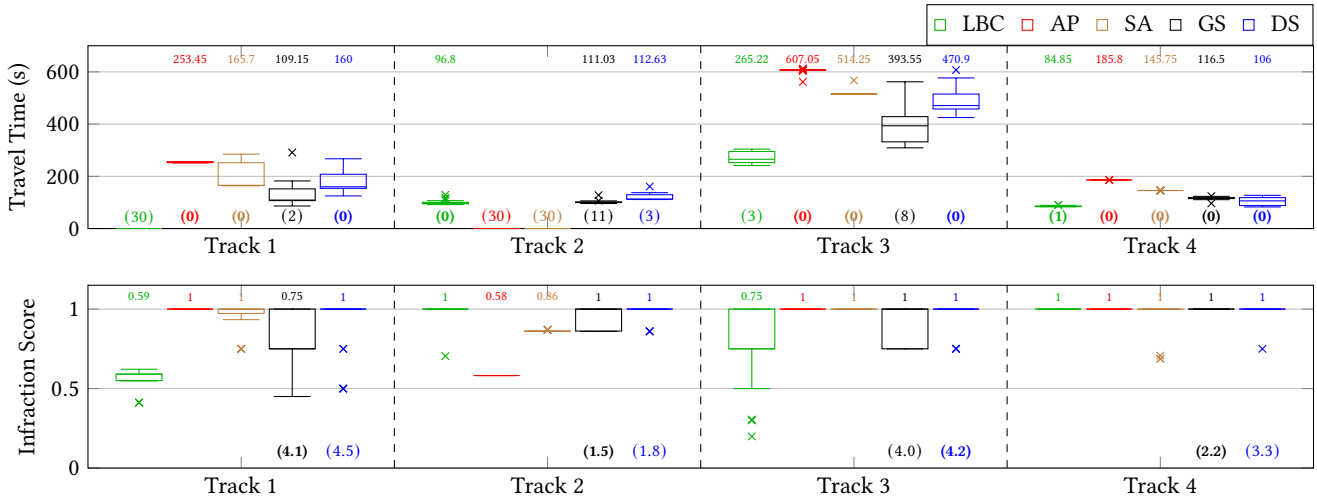

Figure 5: Travel times and safety scores for the AV with intermittent camera occlusion on all 4 tracks. We show the number of times a controller failed to complete a track in parenthesis below each box, highlighting the least failures in bold. We also show the median of the distributions. *DS* achieves the best performance among the controllers that do not fail. We also observe that *DS* consistently has a better performance than *SA* and less variance of infraction score than *GS*, indicating that *DS* maintains a good balance between performance and safety. The mean of reverse switches performed by *GS* and *DS* across all tracks is also shown below each box in the graph of infraction score.

from all the possible system trajectories is presented as a decision logic for hybrid systems. While these approaches have worked well, they require an abstract system model, which is challenging to design for these complex autonomous systems with black-box ML components. There have also been other non-verification approaches that do not require the abstract model. For example, Phan, Dung *et al.* [27] used a compositional proof technique called Assume-Guarantee contracts for switching between the controllers. A rule-based approach using historical performances of the controllers is designed for an unmanned aerial vehicle [35]. Such approaches are promising, they do not include the ability to perform the reverse switch, which is crucial for improving the system's performance.

Although previous works have proposed various promising strategies for switching control to the safety controller, relatively little work has been done to investigate the decision logic for switching back to the performant controller [10, 35]. Desai et al. [10] discuss a general framework for reverse switch that uses reachability analysis to check if the system is safe in the near future, irrespective of the controller that the system is running with. However, this decision logic results in a conservative reverse switch as the switch only happens when all reachable states of both controllers are in the safe set. Recently, many learning-based driving decision models have also been proposed [7, 28, 30]. For example, the Neural Simplex Architecture [28] is proposed to retrain and adapt the performant controller online and performs the reverse switch by leveraging reachability analysis. However, this architecture demands that both controllers are run in parallel and the neural network is updated online, significantly increasing the system's computation burden. Also, reachability analysis generally assumes an accurate system model is available; however, reachability analysis for learning-based controllers is still limited to feed-forward ReLU-based networks [21]. On the other hand, our approach uses an *anytime* algorithm MCTS to find a (near) optimal decision by exploring possible future trajectories in an asymmetric manner. Also, as the set of generative models used by the online heuristic search can be easily updated [23], the proposed approach can seamlessly adapt to any exogenous changes in the environment.

6 DISCUSSION

Finally, we conclude with a discussion about how safety and performance can be balanced in a complex cyber-physical system through dynamic switching between controllers. We hypothesize that the forward switch, which is critical for ensuring the safety of the system, must be: a) computationally cheap in terms of taking time to detect the threat to the system and making the decision for the switch; and b) must place utmost emphasis on the safety of the system. The reverse switch, on the other hand, has different constraints. First, it can afford relatively higher latency, i.e., the system can operate in the *safe mode* while the decision for the reverse switch is computed. Second, the reverse switch must be non-myopic. Note that it is critical for the system to avoid facing the very same threats to safety that had caused the forward switch in the first place; as a result, careful consideration of the evolution of the system, conditional on the environmental parameters, must be performed to ensure that the reverse switch is safe. While the proposed data-driven approach in the paper is not verifiable, the Monte Carlo search is guaranteed to converge (given enough computational time) to the optimal action given the underlying Markov decision process. This observation further points out the need to accurately represent the Markovian process. An important consideration in the reverse switch is also the dynamic nature of the environment, i.e., the non-myopic decision-making approach must be equipped to consider any abrupt changes in the environmental

parameters. This consideration is the major driver for the usage of an online search based approach in the proposed method, as opposed to a *policy* trained offline (e.g., by using reinforced learning or dynamic programming) and then invoked instantaneously during execution.

7 CONCLUSION

We present the dynamic simplex strategy for controller selection in controller-redundant CPS. Our approach provides principled approaches for both forward and reverse switching. The approach balances safety and performance by leveraging a combination of a myopic action selector based on a surrogate model and an online non-myopic planner based on MCTS (for reverse switching). Our experimental evaluations using multiple AV examples in the CARLA simulator show that our approach with only a surrogate model can outperform other state-of-the-art alternatives in improving the system's performance without compromising safety under different environmental conditions and component failures.

Acknowledgment: This work was supported by the DARPA Assured Autonomy project and Air Force Research Laboratory. Any opinions, findings, and conclusions expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA or AFRL.

REFERENCES

[1] Stanley Bak, Taylor T Johnson, Marco Caccamo, and Lui Sha. 2014. Real-time reachability for verified simplex design. In *Real-Time Systems Symposium (RTSS), 2014 IEEE*. IEEE, 138–148.

[2] Stanley Bak, Karthik Manamcheri, Sayan Mitra, and Marco Caccamo. 2011. Sandboxing controllers for cyber-physical systems. In *International Conference on Cyber-Physical Systems*. 3–12.

[3] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games* 4, 1 (2012), 1–43.

[4] Feiyang Cai and Xenofon Koutsoukos. 2020. Real-time Out-of-distribution Detection in Learning-Enabled Cyber-Physical Systems. In *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*. 174–183.

[5] Andrea Ceccarelli and Francesco Secci. 2022. RGB cameras failures and their effects in autonomous driving applications. *IEEE Transactions on Dependable and Secure Computing* (2022).

[6] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. 2020. Learning by cheating. In *Conference on Robot Learning*. PMLR, 66–75.

[7] Jianyu Chen, Bodi Yuan, and Masayoshi Tomizuka. 2019. Model-free Deep Reinforcement Learning for Urban Autonomous Driving. In *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019, Auckland, New Zealand, October 27-30, 2019*. 2765–2771.

[8] Tanya L Crenshaw, Elsa Gunter, Craig L Robinson, Lui Sha, and PR Kumar. 2007. The simplex reference model: Limiting fault-propagation due to unreliable components in cyber-physical system architectures. In *International Real-Time Systems Symposium*. 400–412.

[9] Guido Dartmann, Houbing Song, and Anke Schmeink. 2019. *Big data analytics for cyber-physical systems: machine learning for the internet of things*. Elsevier.

[10] Ankush Desai, Shromona Ghosh, Sanjit A. Seshia, Natarajan Shankar, and Ashish Tiwari. 2019. SOTER: A Runtime Assurance Framework for Programming Safe Robotics Systems. *49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (2019), 138–150.

[11] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An open urban driving simulator. *arXiv:1711.03938* (2017).

[12] Parham Gohari, Franck Djeumou, Abraham P Vinod, and Ufuk Topcu. 2020. Blending controllers via multi-objective bandits. *arXiv preprint arXiv:2007.15755* (2020).

[13] Charles Hartsell, Shreyas Ramakrishna, Abhishek Dubey, Daniel Stojcsics, Nagabhushan Mahadevan, and Gabor Karsai. 2021. ReSonAte: A Runtime Risk Assessment Framework for Autonomous Systems. In *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 118–129.

[14] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. 1998. What's Decidable about Hybrid Automata? *J. Comput. Syst. Sci.* 57, 1 (1998), 94–124.

[15] Carl-Johan Hoel, Katherine Driggs-Campbell, Krister Wolff, Leo Laine, and Mykel J Kochenderfer. 2019. Combining planning and deep reinforcement learning in tactical decision making for autonomous driving. *IEEE Transactions on Intelligent Vehicles* 5, 2 (2019), 294–305.

[16] [Online] International Organization for Standardization. 2021. ISO 26262. https://www.iso.org/standard/43464.html

[17] Jacques Janssen. 2013. *Semi-Markov models: theory and applications*. Springer Science & Business Media.

[18] Taylor T Johnson, Stanley Bak, Marco Caccamo, and Lui Sha. 2016. Real-time reachability for verified simplex design. *ACM Transactions on Embedded Computing Systems* 15, 2 (2016), 1–27.

[19] Levente Kocsis and Csaba Szepesvári. 2006. Bandit Based Monte-Carlo Planning. In *17th European Conference on Machine Learning (Lecture Notes in Computer Science, Vol. 4212)*, Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou (Eds.). 282–293.

[20] Xiangguo Liu, Chao Huang, Yixuan Wang, Bowen Zheng, and Qi Zhu. 2022. Physics-Aware Safety-Assured Design of Hierarchical Neural Network based Planner. In *13th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2022, Milano, Italy, May 4-6, 2022*. IEEE, 137–146.

[21] Alessio Lomuscio and Lalit Maganti. 2017. An approach to reachability analysis for feed-forward ReLU neural networks. *CoRR* abs/1706.07351 (2017). http://arxiv.org/abs/1706.07351

[22] Sibin Mohan, Stanley Bak, Emiliano Betti, Heechul Yun, Lui Sha, and Marco Caccamo. 2013. S3A: secure system simplex architecture for enhanced security and robustness of cyber-physical systems. In *International Conference on High Confidence Networked Systems*, Linda Bushnell, Larry Rohrbough, Saurabh Amin, and Xenofon D. Koutsoukos (Eds.). 65–74.

[23] Ayan Mukhopadhyay, Geoffrey Pettet, Chinmaya Samal, Abhishek Dubey, and Yevgeniy Vorobeychik. 2019. An online decision-theoretic pipeline for responder dispatch. In *ACM/IEEE International Conference on Cyber-Physical Systems*. 185–196.

[24] [Online] National Highway Traffic Safety Administration. 2022. Summary Report: Standing General Order on Crash Reporting for Level 2 Advanced Driver Assistance Systems. https://www.nhtsa.gov/sites/nhtsa.gov/files/2022-06/ADAS-L2-SGO-Report-June-2022.pdf

[25] Geoffrey Pettet, Ayan Mukhopadhyay, and Abhishek Dubey. 2022. Decision Making in Non-Stationary Environments with Policy-Augmented Monte Carlo Tree Search. *arXiv preprint arXiv:2202.13003* (2022).

[26] Geoffrey Pettet, Ayan Mukhopadhyay, Mykel J Kochenderfer, and Abhishek Dubey. 2021. Hierarchical planning for resource allocation in emergency response systems. In *International Conference on Cyber-Physical Systems*. 155–166.

[27] Dung Phan, Junxing Yang, Matthew Clark, Radu Grosu, John D. Schierman, Scott A. Smolka, and Scott D. Stoller. 2017. A Component-Based Simplex Architecture for High-Assurance Cyber-Physical Systems. In *17th International Conference on Application of Concurrency to System Design, ACSD 2017, Zaragoza, Spain, June 25-30, 2017*. 49–58.

[28] Dung T Phan, Radu Grosu, Nils Jansen, Nicola Paoletti, Scott A Smolka, and Scott D Stoller. 2020. Neural simplex architecture. In *NASA Formal Methods Symposium*. 97–114.

[29] Stephen Prajna and Ali Jadbabaie. 2004. Safety verification of hybrid systems using barrier certificates. In *International Workshop on Hybrid Systems: Computation and Control*. Springer, 477–492.

[30] Shreyas Ramakrishna, Charles Hartsell, Matthew P. Burruss, Gabor Karsai, and Abhishek Dubey. 2020. Dynamic-weighted simplex strategy for learning enabled cyber physical systems. *J. Syst. Archit.* 111 (2020), 101760.

[31] Danbing Seto, Bruce Krogh, Lui Sha, and Alongkrit Chutinan. 1998. The Simplex architecture for safe online control system upgrades. In *The 1998 American Control Conference. ACC (IEEE Cat. No. 98CH36207)*, Vol. 6. IEEE, 3504–3508.

[32] Danbing Seto and Lui Sha. 1999. *A case study on analytical analysis of the inverted pendulum real-time control system*. Technical Report. Carnegie Mellon University.

[33] [Online] State of California Department of Motor Vehicles. 2022. Autonomous Vehicle Collision Reports. https://www.dmv.ca.gov/portal/vehicle-industry-services/autonomous-vehicles/autonomous-vehicle-collision-reports/

[34] Kush R Varshney and Homa Alemzadeh. 2017. On the safety of machine learning: Cyber-physical systems, decision sciences, and data products. *Big Data* 5, 3 (2017), 246–255.

[35] Prasanth Vivekanandan, Gonzalo Garcia, Heechul Yun, and Shawn Keshmiri. 2016. A simplex architecture for intelligent and safe unmanned aerial vehicles. In *International Conference on Embedded and Real-Time Computing Systems and Applications*. 69–75.

[36] Yahan Yang, Ramneet Kaur, Souradeep Dutta, and Insup Lee. 2022. Interpretable Detection of Distribution Shifts in Learning Enabled Cyber-Physical Systems. In *13th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2022, Milano, Italy, May 4-6, 2022*. IEEE, 225–235.

A TECHNICAL APPENDIX

Notation	Description
w_t	A sequence of scenes characterized by transitions in structural features w_t^s or in temporal features w_t^q at time step t
s_t	State tuple $< v, w_t^s, w_t^q, C_t, \phi_t, \omega_t >$ at time step t
a	The action a which can be taken
v_t	Velocity of the vehicle
C_t	The controller $C_t \in \{C^P, C^S\}$ driving the system at time step t
d_t	The traffic density at time step t
ϕ_t	The failure state of n components
ψ_t	The runtime monitor state at time step t
ω_t	A counter that keeps track the number of switches has performed
$R(s,a)$	The scoring function which returns the reward given state s and action a
$\lambda(s,a)$	$\lambda \in \{\lambda^P, \lambda^f, \lambda^c\}$ which returns performance score, the safety score, or the cost of switch.
m_s	The maximum number of switches may happen during the planning horizon.
τ^q	The discrete time period after which the parameter with the highest frequency is queried
M	A set of generative models.
G	Surrogate model.
t_e	The estimated time that the system may take to arrive at next closest structural scene.

A.1 Controllers

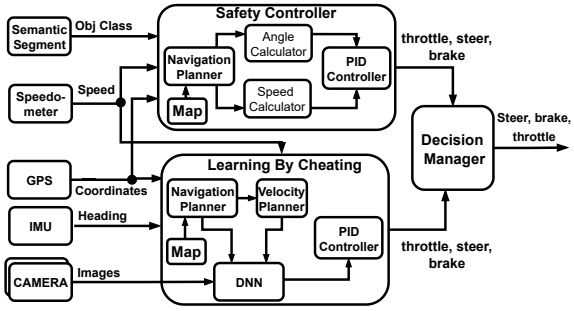

Figure 6: System model of the example AV in CARLA simulation.

Performant Contoller: It uses a navigation planner that takes the waypoint information from the simulator and divides them into smaller position targets. Next, it uses the GPS and IMU sensors to get the vehicle's current position. It feeds this along with the position targets into a velocity planner that computes the desired speed. The desired speed and camera images are fed into a DNN, which predicts the trajectory angle and the target speed. These predictions and the current speed is sent to PID controllers to compute the throttle, brake, and steer control signals.

Autopilot Controller: The controller uses the GPS, the speedometer, and the semantic segmentation camera sensor to compute the control actions as follows. First, it uses the navigation planner to get the preset waypoints and divides them into smaller position targets using the priority information (e.g., position of traffic signs) from the simulator. The position targets, current position, and speed (from GPS and IMU sensors) are sent to angle and speed calculator functions to calculate the trajectory and the desired speed. These values are sent to different PID controllers to compute the throttle, brake, and steer control signals. Finally, the control actions from the two controllers are forwarded to a decision manager with the logic discussed in Section 3.

Controller Operation: In addition to these rules, we also use a warm-up phase during which warms up the controller selected by the decision-maker before bringing it online to operate the system. The warm-up is required because we do not run both controllers in parallel to avoid computational costs and save energy, i.e., when one controller is operating, the other controller is idle and is unaware of the system's current state. If the decision-maker decides to switch, the routine starts to run the idle controller in shadow mode to slowly update it with the system's current state. After the warm-up phase is completed, the selected controller's actions are taken predictions start being used for operating the system.

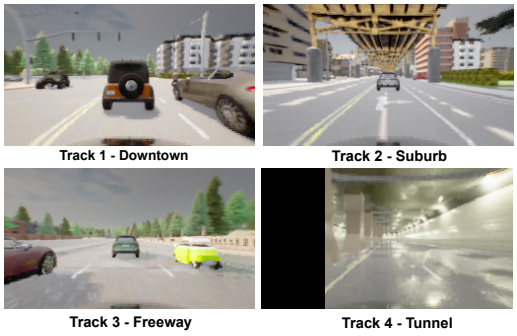

Figure 7: Samples of the tracks as captured by the center camera attached to the AV. The occlusion in the track 4 image is because of a camera failure.

Track Description: Track 1 is around downtown with high traffic density and has several traffic signs in most scenes. Track 2 is around the suburb with an overpass and typically has a low traffic density. Track 3 is around a long freeway with low traffic density for all the scenes. Finally, Track 4 runs through a tunnel and then enters a city with traffic lights and medium traffic density.

A.2 Ablation Study

To understand the effects of the non-myopic planner and the domain rules, we conducted an ablation study. We show the results in Fig. 8. D-Myopic and ND-Myopic indicate the configurations that use a myopic selector with domain rules and a myopic selector without domain rules for reverse switches, respectively. D-Nonmyopic and ND-Nonmyopic indicate the configurations that use a non-myopic planner with domain rules and a non-myopic planner without domain rules for reverse switching, respectively. We observe that coupling domain rules with different reverse-switching configurations have a minor impact on the system's travel times and infraction scores on Track 1, Track 2, and Track 4. However, both D-Nonmyopic and ND-Nonmyopic achieve a smaller variance of infraction score on Track 1, shorter travel time on Track 4, and fewer reverse switches across all tracks than D-Myopic and ND-Myopic, demonstrating that reverse switching with nonmyopic

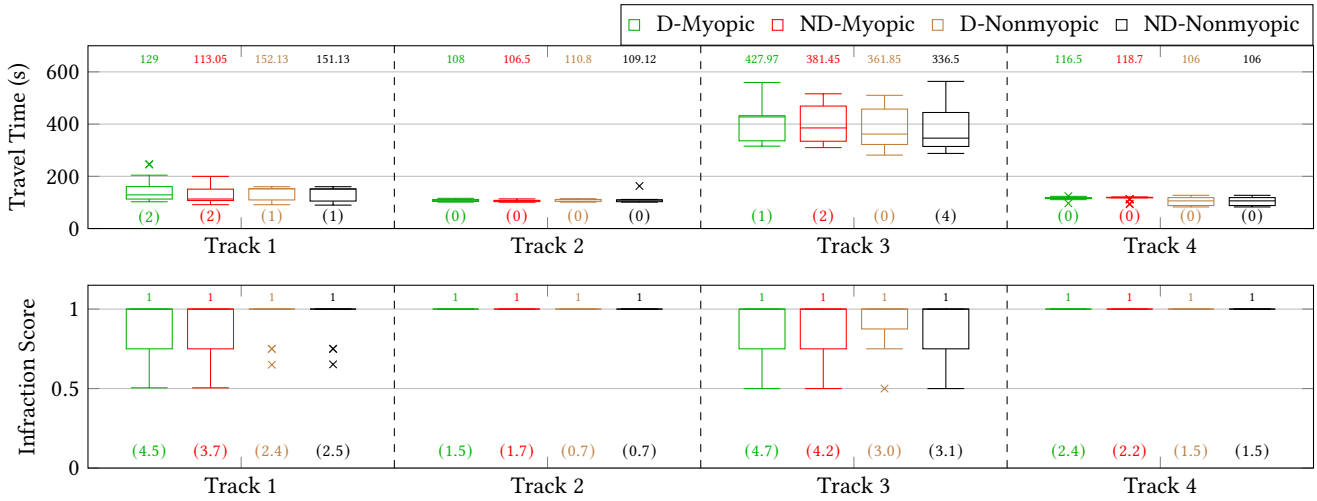

Figure 8: Top: Travel times of the different controller configurations across the 4 tracks (lower is better). We show the number of times a controller failed to complete a track in parenthesis below each box. We also show the median of the distributions at the top of the graph. (Bottom): We show the infraction score (higher is better) for all controller configurations across all tracks. We also show the mean of reverse switches performed by different configurations below each box.

Table 1: Computation Time of Non-Myopic Planner

MCTS Iterations	Average Computation Time (seconds)	Average Travel Time (seconds)	Average Infraction Score	Average Number of Switches
100	**0.38**	234.68	0.95	6.25
500	0.94	**196.84**	**1**	5.42
1000	1.64	232.43	**1**	5.00
2000	3.00	230.58	**1**	**3.33**

planner plays an essential role in improving the system's performance and stability. We also observe that removing domain rules from reverse-switching configurations leads to more failures on Track 3. We hypothesize that this behavior is driven by Track 3 being a long freeway with a low traffic density; therefore, both the performant controller and the safety controller can operate with higher velocities than they can operate on the other three tracks, which are similar in terms of velocities achieved by the controllers. On Track 3, switching without domain rules (e.g., reducing speed for switching) can have a detrimental effect on the system's stability.

A.3 Generative Model

We use different distributions and an artificial neural network as our generative models. First, we sample temporal features and traffic density with random distribution during data collection; therefore, we model the transition of temporal features and traffic density as so. Second, we model the distribution of the permanent sensor failure as a Weibull distribution. We learn the distribution parameters by maximizing the likelihood of sensor failure data. Third, to model the intermittent sensor failure rate, we use an exponential growth function to simulate the likelihood of occlusion conditioned on weather and location, i.e., the sunnier or heavier the precipitation is, the more likely it is to cause a temporary occlusion. We also ensure that such failures depend on the state's structural features, e.g., sunny conditions or precipitation is unlikely to cause occlusion if the vehicle is operating in a tunnel. Finally, to model the duration and the arrival time of runtime monitor alarms, we aggregate historical alarm data and compute the average duration and inter-arrival time of these alarms conditioned on different temporal features, structural features, and traffic density. Then we use these historical data to train an artificial neural network as our generative model.

A.4 Computation Time

Table 1 shows the average computation times, travel time, infraction score, and the average number of switches taken by the decision logic of our proposed *DS* configurations. Note that the average computation time highly correlates to the computational capacity of the hardware. Recall that the forward switching of the configuration is performed based on inference by a model trained using historical data. As a result, it is extremely fast, as required in practice for ensuring safety. The *DS* uses the MCTS-based planner for reverse switching, which requires an average time proportional to the increased number of MCTS iterations. We observe that 500 MCST iterations give the best average travel time without harming the system's safety. We also observe that the planner with 100 MCTS iterations takes the least time to make decision while sacrificing the infraction score and increasing the number of switches. However, it can still offer a competitive average travel time compared to the planner with 1000 and 2000 MCTS iterations, enabling the decision-maker to put constraints on the computational time and latency that can be afforded to make a decision depending on domain-specific requirements.