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

Research Paper

CS637 Course Project

TEAM MEMBERS

1. Ganesh Srivatshava (220301)
2. Dargu Sakshi Sameer (220323)
3. Harshita Chhuttani (220447)
4. Kshitij Pratap Singh (220554)

GOAL OF THE SYSTEM

The primary goal of the system is to balance safety and performance in autonomous Cyber-Physical Systems (CPS) by implementing a dynamic simplex strategy. This strategy allows for two-way switching between safety and performant controllers, minimizing frequent back-and-forth arbitration.

The system aims to ensure safety through myopic actions while using non-myopic planning methods, such as Monte Carlo Tree Search (MCTS), for optimal decision-making when switching back to the performant controller. This approach is evaluated through various autonomous vehicle studies in simulated urban environments, demonstrating fewer infractions and higher performance compared to state-of-the-art alternatives.

INTRODUCTION TO AUTONOMOUS CYBER PHYSICAL SYSTEMS

- Autonomous Cyber-Physical Systems (ACPS) are complex, interconnected systems that combine physical components with computational intelligence to operate and make decisions independently.
- They integrate sensors, actuators, computation, and networking capabilities, allowing them to monitor and control physical processes in real-time, often with minimal or no human intervention.

LEC (Learning enabled components)

- LECs are subsystems within a larger cyber-physical or autonomous system that use learning-based approaches (e.g., neural networks, reinforcement learning) to make decisions or perform specific functions.
- These components enable the system to interpret and act on sensory data, adapt to changing conditions, and handle tasks that are difficult to program explicitly.
- Learning Enabled Components (LEC) have greatly assisted cyberphysical 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

CHALLENGE

- 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 trained using ML methods. A critical challenge for these systems is making safe and efficient decisions under unanticipated system faults and dynamically changing operating conditions
- 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

LIMITATIONS OF REDUNDANT ARCHITECTURES

Redundant controller architectures have been widely adopted for safety assurance. 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, there are two limitations.

- 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.
- They do not support reverse switching from the safety controller to the performant controller, even when the threat to safety is no longer present.

Transitioning back to the performant controller is complex.

1

Safety is paramount: an unplanned switch could jeopardize the health of the system and any surrounding entities, including humans. Therefore, any transition must carefully assess the system’s evolution and environmental factors.

2

External factors can change dynamically, necessitating planning based on real-time information, though online planning is often slower than offline methods

3

Frequent switching between controllers can destabilize the system, further complicating the design of reverse-switching logic.

PROBLEM STATEMENT

“How can autonomous CPS systems implement a robust, two-way controller switching mechanism that maintains safety in the presence of faults and uncertainties while optimizing performance in dynamically changing environments?”

Thus we wish to design a method that balances the objectives of performance and safety, optimizing the system’s behavior while managing switching costs.

KEY CONTRIBUTIONS OF THE PAPER

Dynamic Simplex	We propose an online, two-way switching strategy that minimizes unnecessary back-and-forth switching.
Semi-Markov Decision Process Framework	We frame the decision-making problem as a semi-Markov decision process, using a myopic approach for forward switching (to prioritize safety) and a non-myopic approach for reverse switching. Monte Carlo Tree Search (MCTS) is employed to optimize the reverse switch.
Runtime safety monitoring	We incorporate data-driven safety monitors that assess system risk by tracking faults and environmental uncertainties in real-time.

PROBLEM SETUP

System Overview - 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.

Controller Functionality - The system employs controllers that adjust parameters like speed and steering angle to optimize performance (e.g., minimizing travel time) and ensure safety.

Controllers -

- Performant Controller (C_p) - Optimizes performance, typically trained with ML on data (scenes) capturing spatial and temporal features like road type and weather.
- Safety Controller (C_s) - Prioritizes safety actions like braking or speed reduction, activated when hazards are detected.

- **Decision Logic** - Runtime monitors detect hazards (e.g., sensor failure or out-of-distribution data) and trigger decision logic to switch control from the performant controller to the safety controller when needed.

- **Objective** - Design a strategy that balances performance and safety effectively.

Components of the Safety System

1. Runtime Monitors: These continuously assess the vehicle's environment and internal systems for potential hazards. Monitors raise alarms to indicate whether a detected issue is critical (requiring immediate action) or non-critical
2. Safety Controllers (Cs): A specialized subsystem that prioritizes safety over performance
3. Decision Logic: decides when to switch control from the performant controller to the safety controller based on hazard severity

For example:

- If a critical hazard is detected (e.g., a pedestrian suddenly crossing the road), the decision logic ensures an immediate switch to the safety controller for emergency braking.
- For non-critical hazards (e.g., slight sensor drift), it might log the issue or apply minor corrections without switching control.

PROBLEM FORMULATION

The autonomous CPS and the environment conditions (w) are our system of interest.

An assumption is that the decision-maker knows the spatial features (w^s), the finite set of scenes the vehicle will travel through and the future temporal features (w^q) are unknown. Evolution of the system is considered in continuous time.

Dynamics of the decision-making are governed by the following –

- When the spatial parameters change
- When the temporal parameters change
- When a component of the system fails
- When the traffic density changes
- When the runtime monitor state changes

Why Semi Markov Decision Process?

- The timing of events in the system isn’t predictable with a simple, fixed pattern; instead, it follows an *exogenous distribution*, meaning the intervals between events depend on external factors and vary in complex ways. For example, how often the road curvature changes depends on the speed of the car—if the car is moving quickly, it may encounter new curves sooner than if it were moving slowly.

- This unpredictability implies that the intervals between events are *not memoryless*, unlike in a traditional Markov process. In a memoryless process, the likelihood of an event happening next is constant and independent of how long it’s been since the last event. Here, though, event timing depends on past conditions, like the car's speed or other factors.

To model these varying intervals and the continuous progression of the system, the decision-making problem is framed as a *semi-Markov decision process (SMDP)*.

Semi-Markov Decision Process (SMDP) Framework

· An SMDP can handle these non-fixed timing intervals between events, allowing it to account for more realistic, complex transitions that depend on the system's evolving state.

An SMDP is represented by a tuple {S, A, T, R, τ}

S is the finite set of States. $s_{t} \in S$ is represented by the tuple (v_t, w_t^s, w^q_t, d_t, C_t, Φ_t, Ψ_t, ω_t).

- 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
- Ψ_t is the runtime monitor state (e.g. OOD detector)
- ω_t is the counter that keeps track of the number of switches that have been performed until the current time.

Semi-Markov Decision Process (SMDP) Framework

Assume $\Phi_t \in \{0,1\}$ for all i, t.

- A is the set of Actions. Our action is just a binary choice whether to or not to switch the container.
- For transitions, we need access to a set of generative models for stimulating them. This is because the evolution of our system is governed by several stochastic processes. 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 controllers.
- T is the state-action transition model
- R is the reward function
- τ is the temporal distribution over state transitions

FORWARD SWITCHING

Reward Function: The reward function is a weighted sum of performance and safety scores, penalizing frequent switching:

- $R(st, a) = \alpha_1 \cdot \lambda\square(st, a) - \alpha_2 \cdot \lambda f(st, a)$
- $\lambda\square$: Performance score based on speed.
- λf: Safety score based on collision likelihood.
- st: Refers to the state of the system
- a: the binary action of switching to the safety controller
- α_1 , α_2 : hyperparameters

REVERSE SWITCHING

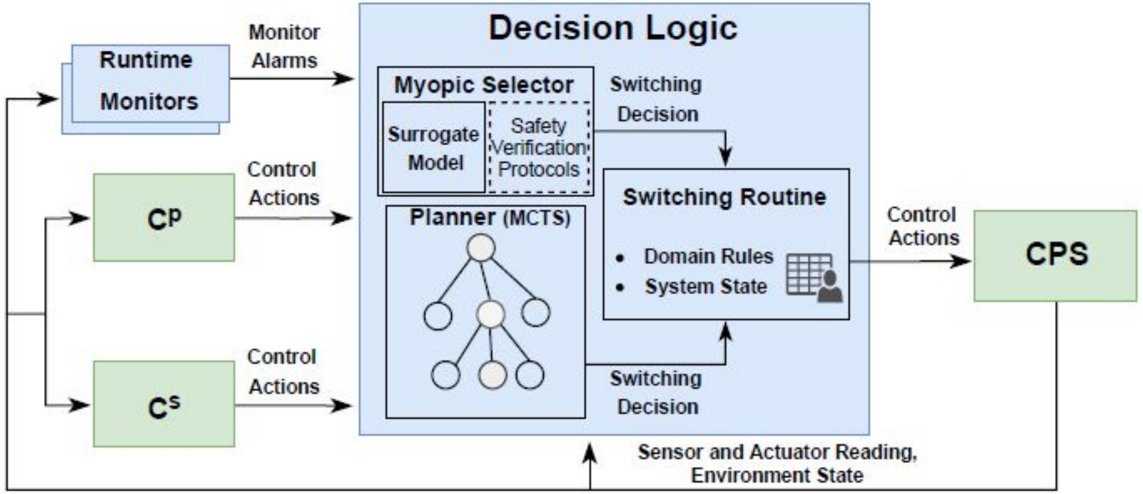
Reward Function: The reward function is a weighted sum of performance and safety scores, penalizing frequent switching:

- $R(s_t, a) = \alpha_1 \cdot \lambda□(s_t, a) - \alpha_2 \cdot \lambda_f(s_t, a) - \alpha_3 \cdot \lambda_c(s_t, a)\ \lambda□$
- λ_p: Performance score based on speed.
- λ_f: Safety score based on collision likelihood.
- λ_c: added another parameter λ^c to prevent back-and-forth switching. It adds a penalty based on the number of previously performed switches leading up to the current state.
- s_t: Refers to the state of the system
- a: the binary action of switching to the safety controller
- α_1 , α_2 , α_3 : hyperparameters

Based on the above SMDP, given a state, our goal is to choose actions based on a utility function (e.g., expected discounted reward).

DYNAMIC SIMPLEX STRATEGY

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.



DYNAMIC SIMPLEX STRATEGY

Objective: Balance safety and performance in CPS by switching between safety and performant controllers based on real-time risks.

Switching Criteria:

- **To Safety Mode:** Triggered by detected risks using safety protocols and past data.
- **To Performance Mode:** Returns to performant mode once conditions are safe.

Key Components:

1. **Myopic Selector:** Quickly assesses risks for immediate action.
2. **Non-Myopic Planner:** Predicts safe return to performance mode.
3. **Runtime Monitors:** Detect anomalies for re-evaluation.

Workflow: Continuously evaluates risks to switch controllers and maintain optimal performance.

DS ARCHITECTURE

- For the forward switch (i.e., from the performant to the safety controller), we take the action that maximizes the myopic one-step reward which ensures that any imminent threats to safety are thwarted

- 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

- 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
  3) a set of runtime monitors to monitor changes in environmental parameters and sensor faults.

1

control logic first checks which controller is driving the system

2

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.

3

Next it checks if the safety controller is operating, the switcher activates the planner.

4

The planner then 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.

5

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

Safety Aspect of Dynamic Simplex Strategy

Myopic Action Selector

General ways to ensure safety used to be taken care only by safety verification methods :

Reachability Analysis , OOD (out of distribution) detection methods .
Although it has ensured safety to a decent levels , these are of limited efficiency and accuracy , especially for a CPS like AV which is very time sensitive and needed to operate in dynamic environments .

Computing the exact reachable set for a nonlinear system is a complex problem and adding to that usage of LEC deepens the complexity of the problem.

This is where "Myopic" Action selector improves forward switching decision process

It integrates the output of Safety Verification methods and complements it with surrogate models into decision making procedure

Specifically , it takes the current state (St) as its input and returns an aggregated score which captures the immediate threat to the safety of system , along with performance objective as seen in this equation ,

$$R(s_t, a) = \alpha_1 \cdot \lambda\square(s_t, a) - \alpha_2 \cdot \lambda_f(s_t, a)$$

where $\lambda\square$ is based on normalized average speed , and λ_f is the collision likelihood.

Performance Aspect of Dynamic Simplex Strategy

Non-Myopic Planner

Earlier, reverse switching decisions between safety and performance controllers were based on simple policies or reachability-based methods. These included:

- **Reachability Analysis**: Checking if all reachable states under both controllers stay within the safe set.
- **Rule-Based Methods**: Relying on past performance to decide when to switch.

While basic methods were conservative and inefficient, the **Non-Myopic Planner** improves decision-making by using a semi-Markov decision process (SMDP) and Monte Carlo Tree Search (MCTS) to simulate future trajectories, optimizing switching decisions for better performance and adaptability.

How it Works:

- Input: Current state s_t
- Output: A switching decision that maximizes the expected discounted cumulative reward:

$$R(s_t, a) = \alpha_1 \cdot \lambda\square(s_t, a) - \alpha_2 \cdot \lambda_f(s_t, a) - \alpha_3 \cdot \lambda_c(s_t, a)$$

Where:

- λ_p: Performance metric (average speed)
- λ_f: Safety metric (collision risk)
- λ_c: Penalty for frequent switching

By using MCTS and real-time monitoring, the Non-Myopic Planner makes smarter, adaptive decisions for autonomous systems, optimizing both safety and efficiency.

Key Features

- **Generative Models:** Simulates future conditions using historical data like Weather, traffic, sensor failures, runtime monitor alarms.
- **Tree Policy:** Uses **Upper Confidence Bound for Trees (UCT)** to explore future trajectories and select actions.
- **Default Policy:** Random rollouts (switch or stay) to simulate possible outcomes.
- **Tree Search:** Iterative process with MCTS, focusing on promising actions based on environmental data.
- **State Transitions:** Discrete time steps with sampled parameters (location, velocity, traffic, sensors) to predict future states.
- **Reward Computation:** Surrogate model used to evaluate state-action pair rewards.

Advantages:

- Balances safety and performance optimization.
- Adapts to dynamic environments (weather, traffic).
- Minimizes excessive switching for efficient decision-making.

Ensures long-term, smarter decision-making, reducing unnecessary transitions while maintaining safety and performance.

Runtime Monitors

Sensor Fault Monitoring ensures safety in autonomous systems by tracking faults. These include **permanent faults** (e.g., broken camera) and **intermittent faults** (e.g., occlusion).

A common fault in vision systems is **occlusion**, where part of the camera view is blocked. It is detected by identifying black blobs in images, and the percentage of occluded pixels is monitored in real-time using a threshold.

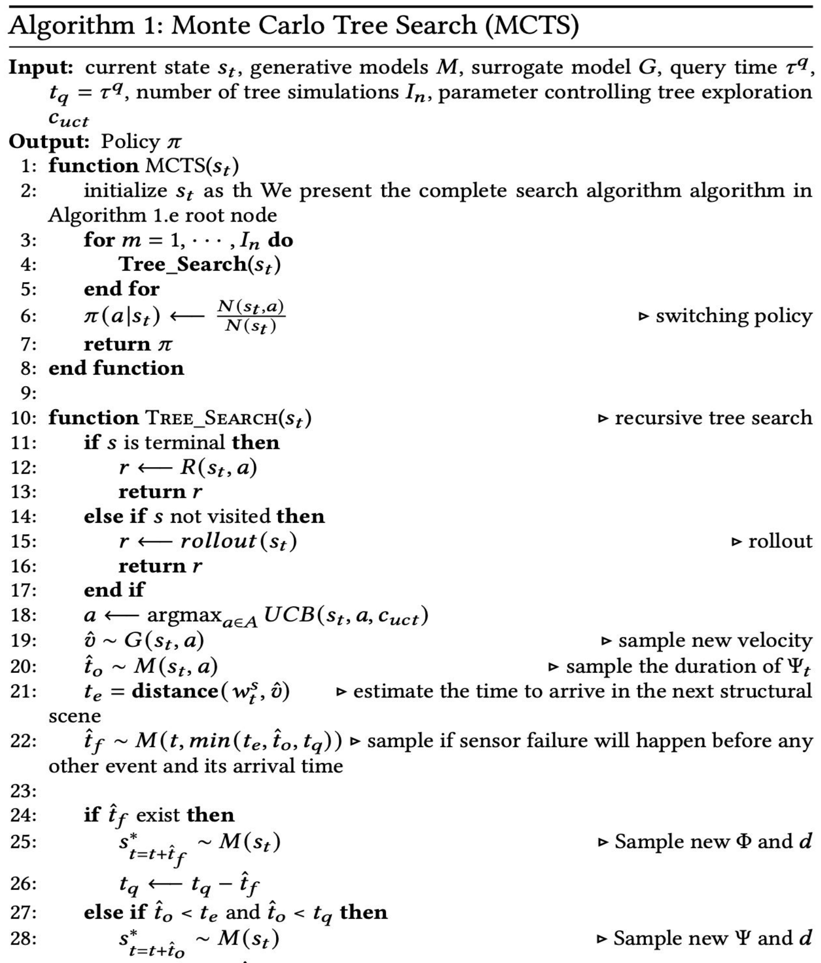
Anomalies are detected using a trained **Variational Autoencoder (VAE)**, where reconstruction error measures anomalies. The **Inductive Conformal Anomaly Detection (ICAD)** method classifies samples as OOD if the p-value of the error is below a threshold.

These monitors play a critical role in ensuring the system operates safely by detecting sensor faults and identifying anomalies, allowing for timely decision-making and fault handling in dynamic environments.

What happens when an obstacle is detected?

In trying to achieve the system’s performance objectives, the performant controller may neglect the safety objectives. As a result, these systems are also equipped with several runtime monitors, a safety controller Cs , 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.



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 th We present the complete search algorithm algorithm in Algorithm 1.e 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 $\pi$
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 $\Psi_t$
    $t_e = \textbf{distance}(w_t^s, \hat{v})$      ▷ 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 $\Phi$ 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 $\Psi$ and $d$
```

Inputs are already given in pic , output is policy π which is the probability distribution over possible actions .

Function MCTS

1.Initializes the current state s□ as the root node of the search tree.

2.Runs a loop for I□ simulations to explore possible actions and their outcomes using `Tree_Search`.

3.Calculate the **policy** using

π(a/st)= N(st,a)/N(st)

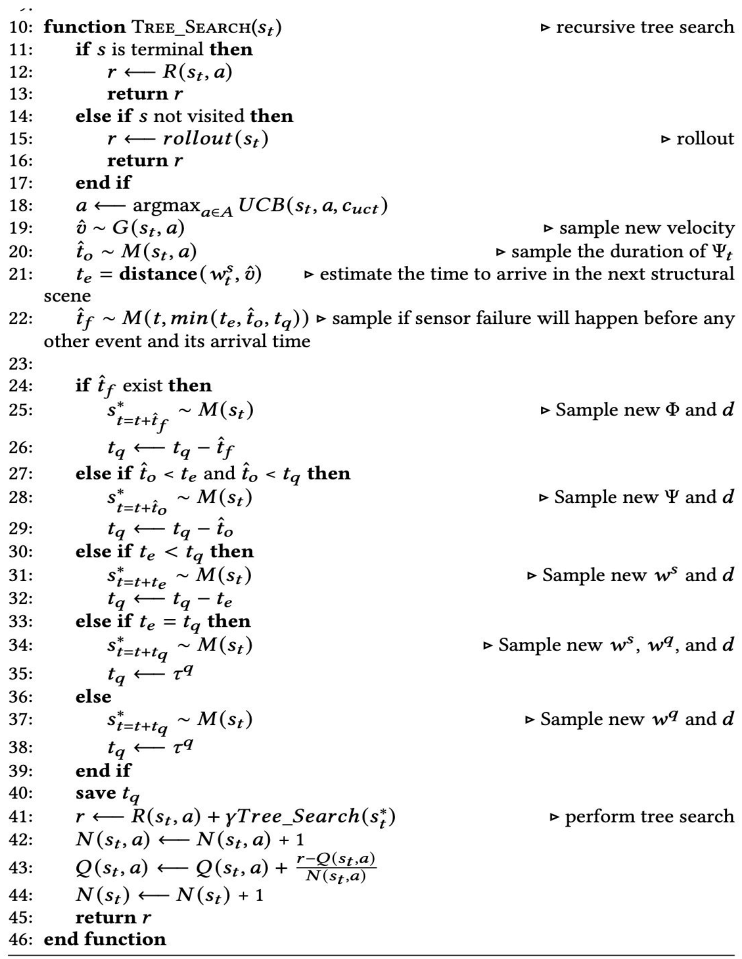
This computes the likelihood of selecting action a based on visit counts , then returns the action policy π, indicating the optimal action to take.

Function Tree Search

1.It explores possible future states recursively to build the search tree .

2.If the current state s is terminal, returns the reward for that state

3.If the state has not been visited before, perform a **rollout** (random simulation) to estimate a potential reward and return that value.



```
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 $\Psi_t$
    $t_e = \textbf{distance}(w^s_t, \hat{v})$        ▷ 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 $\Phi$ 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 $\Psi$ 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
```

Action Selection Using Upper Confidence Bound (UCB)

1.Selects an action a using the **Upper Confidence Bound (UCB)** formula

a= argmaxa∈A UCB(𝑠t,a,Cuot)

2.The UCB formula balances between:

Exploration: Trying actions that haven't been tried much.

Exploitation: Choosing actions known to yield high rewards.

State Transition and Simulation

1: Sample a new velocity v^ using the surrogate model G.

2: Estimate the time t_e required to reach the next significant scene using:

te=distance(ws,v^)

3: Determine the durations for various events:

t_o: Duration until the next planned event.

tf: Duration until a potential sensor failure occurs.

```
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 $\Psi_t$
    $t_e$ = 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 $\Phi$ 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 $\Psi$ 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
```

Decision Logic for State Transitions

1. Different scenarios handled:

If a **sensor failure** is imminent (*tf* is the smallest), transition to a new state Φ with updated sensor data.

If an **event** (*t*ₒ) occurs sooner than reaching the next scene, adjust the state with new conditions Ψ.

If the next **structural scene** change (*t*ₑ) happens sooner, update the environment (*w*□) to reflect the change.If none of the above happens within the query time *t*□, update the system based on the query duration

Reward Calculation and Backpropagation

After determining the outcome of an action, compute the **reward**: r = R(*s*□, a) + *γTree_Search*(*s*t*)

R(*s*□, a) is Immediate reward for taking action a in state *s*□.

γ is Discount factor that reduces the importance of future rewards.

Tree_Search(*s*t*): Recursively continues exploring from the new state *s*□*.

Update visit counts:

Increment visit counts N(*s*□, a) and N(*s*□) to reflect the exploration.

Update quality values   Q(*s*t,a)=Q(*s*t,a)+(r−Q(*s*t,a))/(N(*s*t,a))

This formula updates the estimated value of each action.

Reverse Switching Mechanism Summarized

- Given a state st such that $Ct = Cs$, we use **MCTS** to evaluate whether or not to switch the controller to Cp .
- MCTS operates by iteratively building a search tree that represents **future trajectories.**
- In each simulation, a child node is recursively selected until a leaf node is reached.
- 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 are used
- At a given state st (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., wst) and the velocity of the vehicle vt . Depending on the action taken, vt can change.
- We denote this **arrival time by te** . The time te can then be compared with τq 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).

EXPERIMENTS

- An AV example in CARLA simulator has been used to evaluate the dynamic simplex strategy.
- Tracks with different weather conditions are used .
- We consider the road type, road curvature, presence of traffic signs, and traffic density as **structural (spatial) features (w^s)**.
- Cloudiness, precipitation, and precipitation deposit parameters as **temporal features (w^q)**

A.1 Controllers

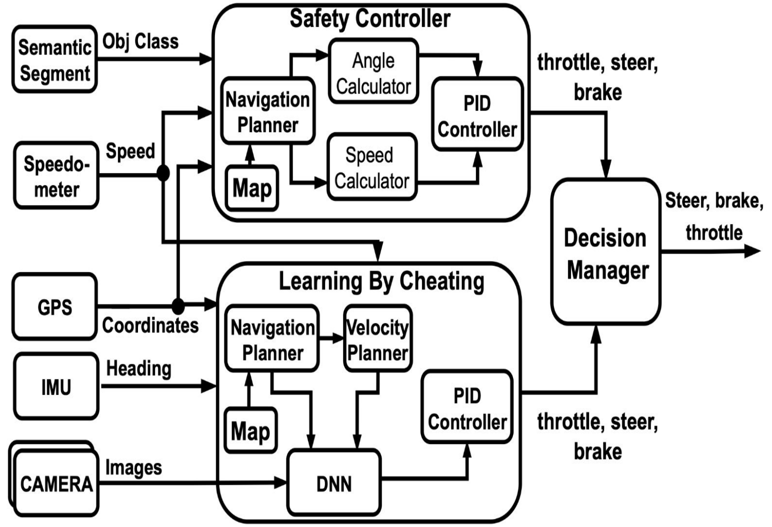


Figure 6: System model of the example AV in CARLA simulation.

Performant Controller : It uses navigation planner that takes waypoint info from the simulator and divides them into smaller position targets . It uses GPS and IMU sensors to get vehicles latest 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.

In our case our performant controller is a ML-based controller called as LBC(learning by cheating) , which uses DNN for navigation . It uses 6 sensors, 3 forward looking cameras , an IMU(inertial measurement unit) , GPS and a speedometer.

A.1 Controllers

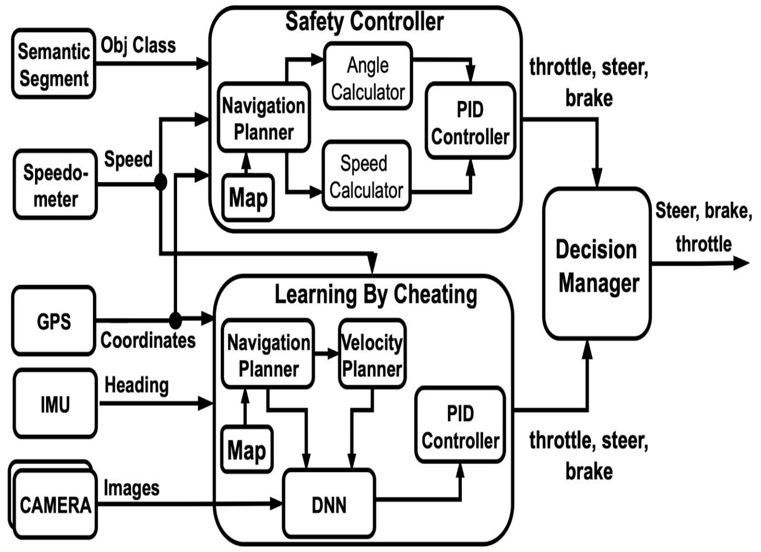


Figure 6: System model of the example AV in CARLA simulation.

Safety Controller : We use Autopilot controller as our safety controller.

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.

In our case , we use Autopilot as our safety controller it uses GPS ,speedometer & semantics segmentation camera sensor to compute control action

CONTINUED...

Controller operation : In addition to controller specific rules , we also use a warm-up phase which warms up the controller selected by decision maker before bringing it online .

Why warm up?
Because , we dont run both controllers in parallel to avoid computational costs , which means when one controller is operating , other is idle and unaware of systems current state . If 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

CONTINUED...

Data for Surrogate model : We run many simulations in CARLA , by randomly varying weather parameters , traffic density and by introducing camera faults(blur and occlusion) .

Run time monitors : For sensor failures , we design a monitor that can detect occlusion in the AV. 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.

RESULTS

Experimental Setup

- Execute each controller for 30 laps around the track
- For each lap, initialize the environment with a random weather condition to introduce variability

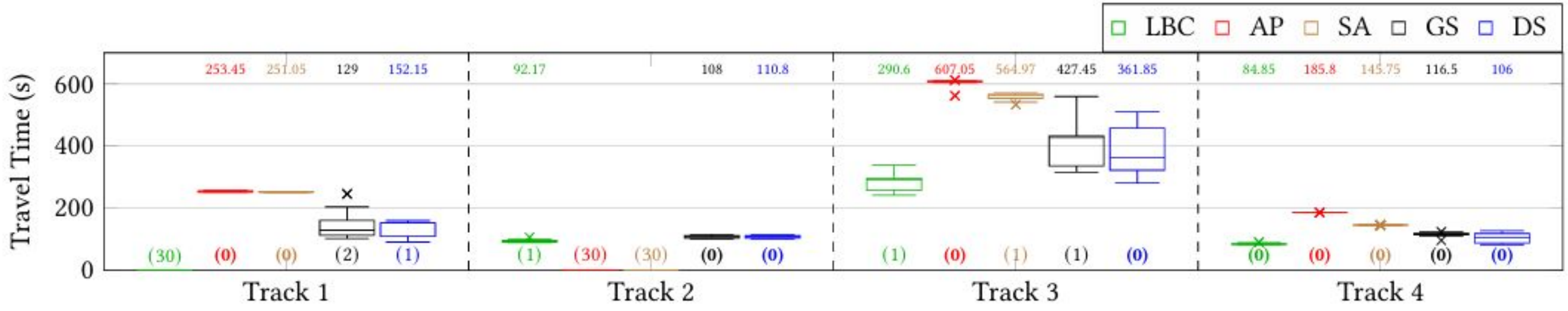
Evaluation Procedure

- First, evaluate the model without sensor failures to establish baseline performance metrics
- Subsequently, injector sensor failures into the system t random intervals during the laps to assess the model’s robustness under adverse conditions.

Analysis

- Analyse the impact of the sensor failures on the model’s performance
- Compare the results of both failure-free and failure-injected scenarios to study the effect of sensor faults system behaviour,

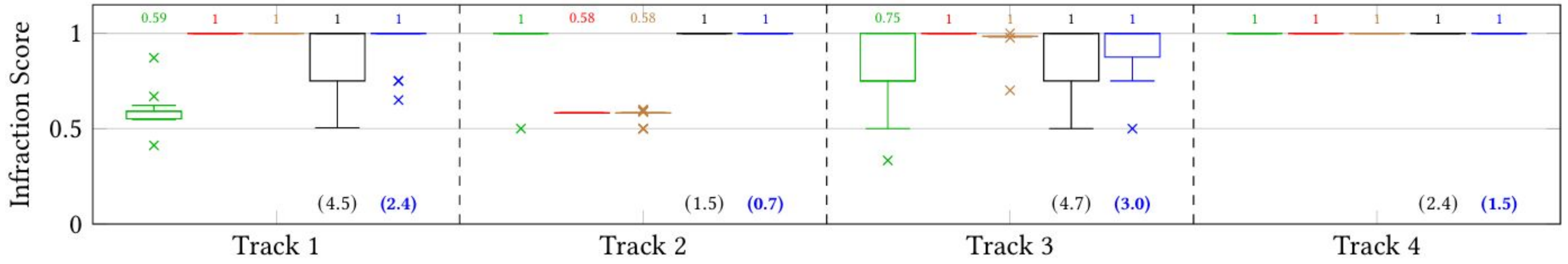
AV OPERATION WITH NO SENSOR FAILURE



Firstly, we begin by evaluating the performance of the controller configuration across all tracks without any *sensor failure.*

The above graph shows the travel time of the different controller configuration on x-axis. The lower the time, the better. At the bottom, the number in bracket shows the number of times a controller failed to complete the track. Median time taken across the 30 trials is mentioned at the top. Cross means the number of reverse switches (from safety to performance)

LBC is the fastest but it fails every time (30) on the first track. AP controller is very erratic. Longer travel time by DS (Dynamic Simplex) is the result of prioritizing safety.



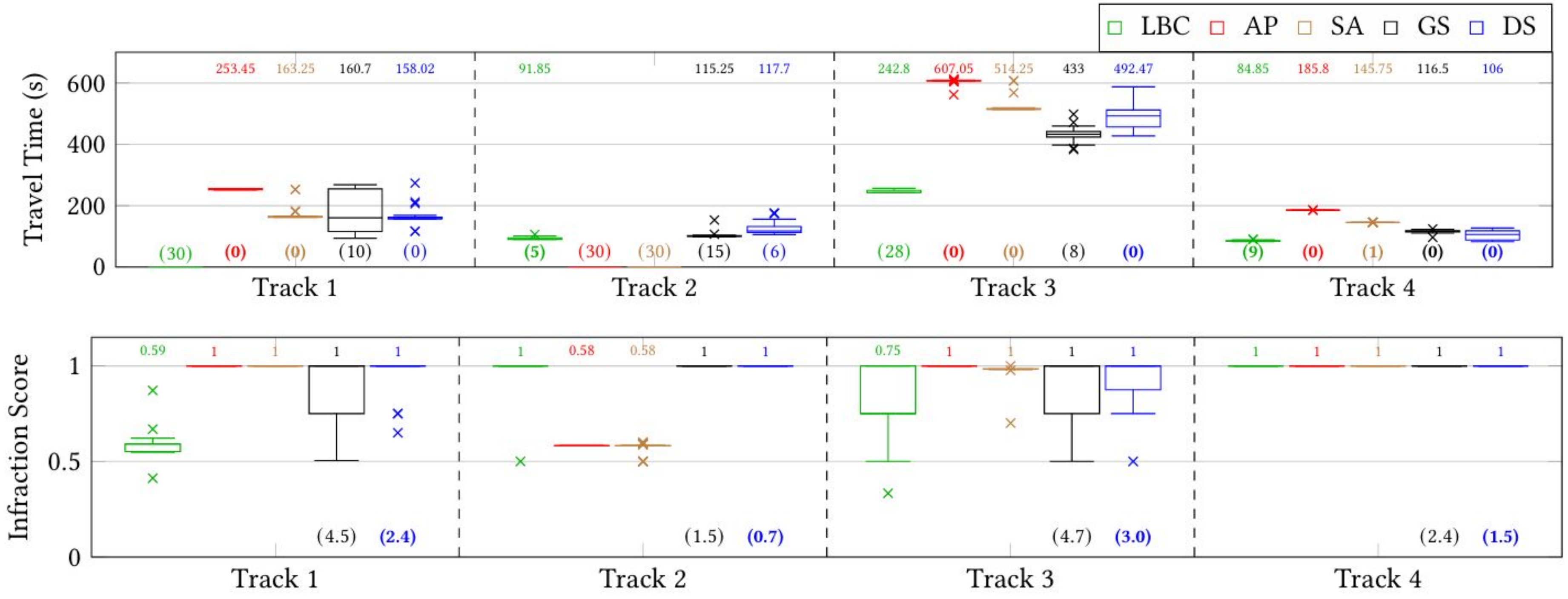
This plot shows the infraction score which is defined as $0.5{*}RC + 0.25{*}col_v + 0.25{*}col_o$, where RC is the percentage of the route that was completed by vehicle, col_v and col_o is the presence of collisions with other vehicles and objects. If any collision in observed, then col_v and col_o are 0 otherwise they are 1. This means the higher the score, the better the safety of the vehicle.

So DS has the best median infraction score and it achieves competitive travel times.

Traditional simplex architecture, SA is the safest. LBC has a low median infraction score of 0.75 on track 3.
DS performs less reverse switches than GS.

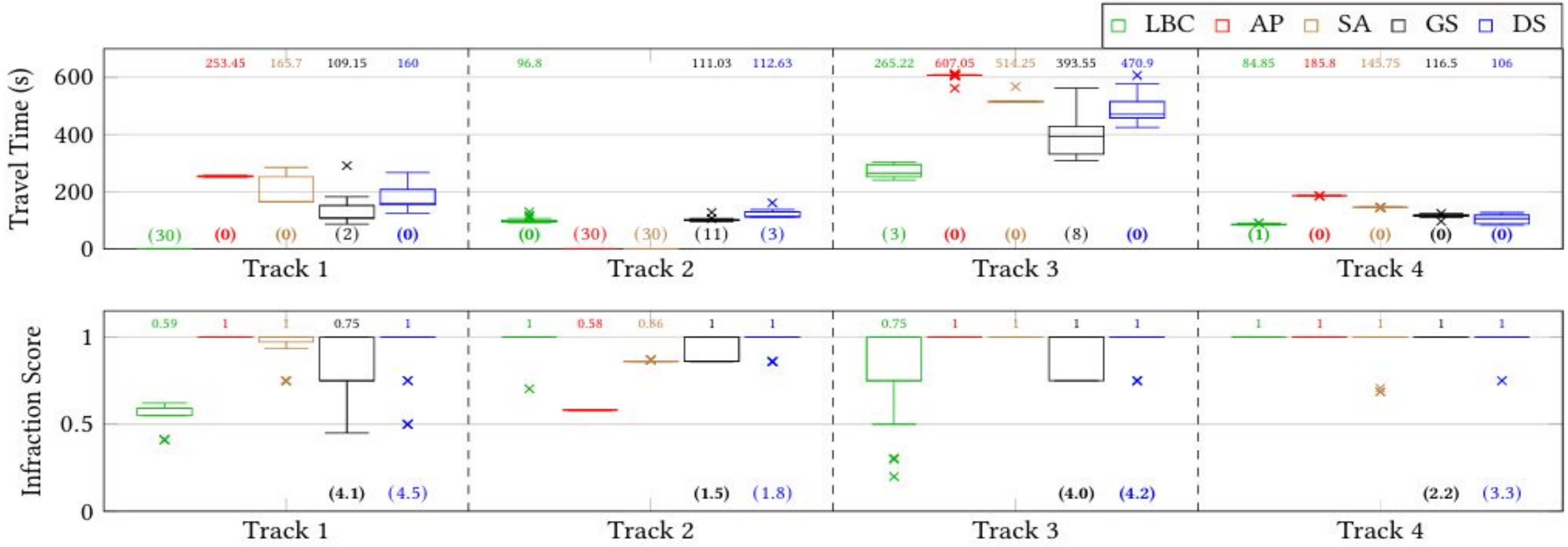
DS outperforms other approaches in terms of balancing safety and performance.

AV OPERATION WITH PERMANENT CAMERA OCCLUSION



LBC is severely affected (it collides in all cases). AP is unaffected by permanent occlusion. **DS outperforms SA and AP in travel time. DS also achieves a median infraction score of 1 on every track.**

AV OPERATION WITH INTERMITTENT SENSOR FAILURE

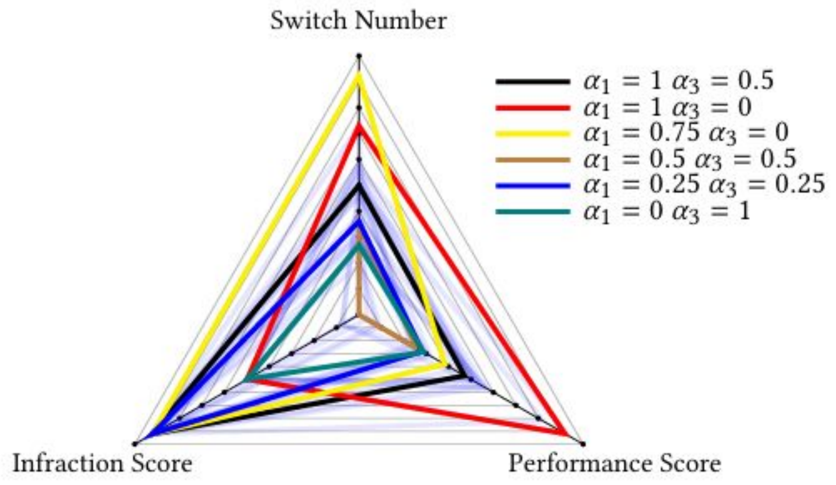


The intermittent failures do not affect the controllers as much as the permanent failures. GS controller fails to complete the tracks significantly more times than the other controllers.

DS controller shows a lower median travel time while performing equally in terms of safety than GS. GS sacrifices the safety and its ability to complete the track to offer lower median travel time on track 2 and 4.

SENSITIVITY ANALYSIS

The parameters α1, α2, α3 are the weights to **trade-off** between performance, safety and the avoiding frequent switching. We conducted a sensitivity analysis to investigate the effects of varying the weights α1, α2, and α3 on the system's performance. Here is a summary of the methodology and findings:

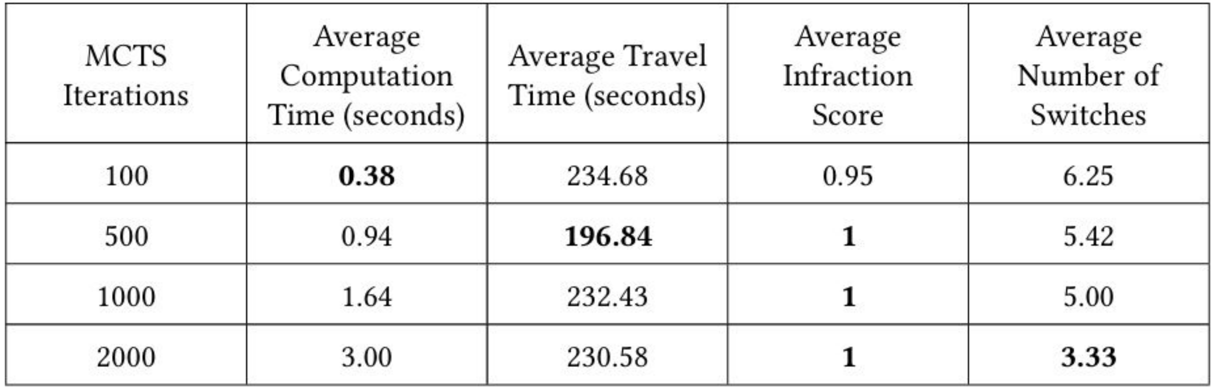


Methodology:
• Varying α3: We fixed α1 and α2 and varied α3 from 0 to 1.
• Varying α1: We then fixed α2 and α3 and varied α1 from 0 to 1.
• Visualization: For clarity, 6 out of 25 results were plotted with different colors, while the rest were shown in blue with low opacity.
• Safety Considerations: Tuning α2 was generally avoided as it could compromise system safety

Findings:
• Increasing the value of α3 from 0 to 1 gradually reduced the number of switches.
• Small values of α1 generally led to lower performance scores, indicating longer travel times.
• Setting α1 to 1 and α3 to 0 resulted in the best performance scores but at the expense of safety, as indicated by an increased infraction score.
• Conversely, setting α1 to 0 and α3 to 1 (penalizing switches heavily) significantly sacrificed performance, highlighting the need for balanced switching to optimize both safety and performance

COMPUTATIONAL TIME

- The Dynamic Simplex (DS) utilizes a Monte Carlo Tree Search (MCTS)-based planner for reverse switching. The time required increases proportionally with the number of MCTS iterations.
- Using 500 MCTS iterations provides the optimal average travel time while maintaining the system's safety.
- In contrast, a planner with **100 MCTS iterations** makes decisions more quickly but compromises the infraction score and increases the frequency of switches.



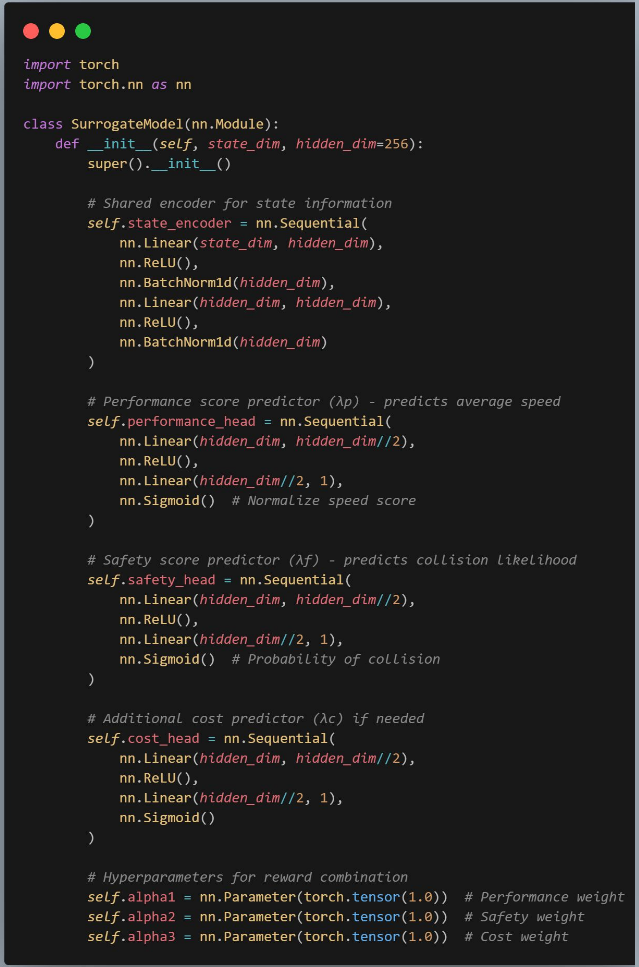
MCTS Iterations	Average Computation Time (seconds)	Average Travel Time (seconds)	Average Infraction Score	Average Number of Switches
100	**0.38**	234.68	0.95	6.25
500	0.94	**196.84**	**1**	5.42
1000	1.64	232.43	**1**	5.00
2000	3.00	230.58	**1**	**3.33**

OUR IMPLEMENTATION OF MODELS AND ALGORITHMS IN THE PAPER

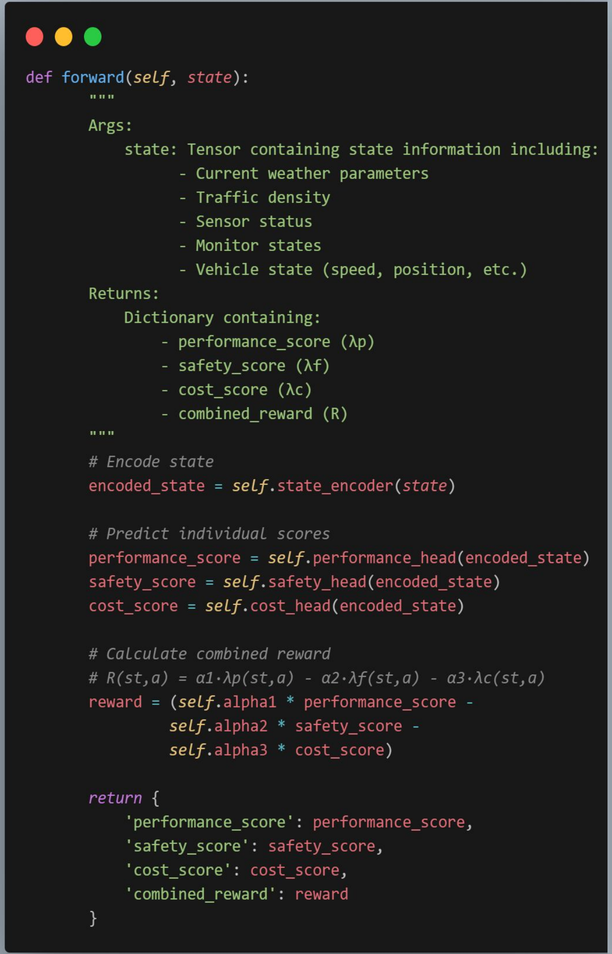
We have tried to replicate some of the models given in the paper like the surrogate models which is used to estimate the performance ($\lambda^{p}(s_t,a)$) and safety score ($\lambda^{f}(s_t,a)$). This will be trained on 50 road segments across the CARLA towns. Weather parameters, traffic density and camera faults will be randomly introduced.

We also implemented generative models which sample future trajectories. This is how they work-

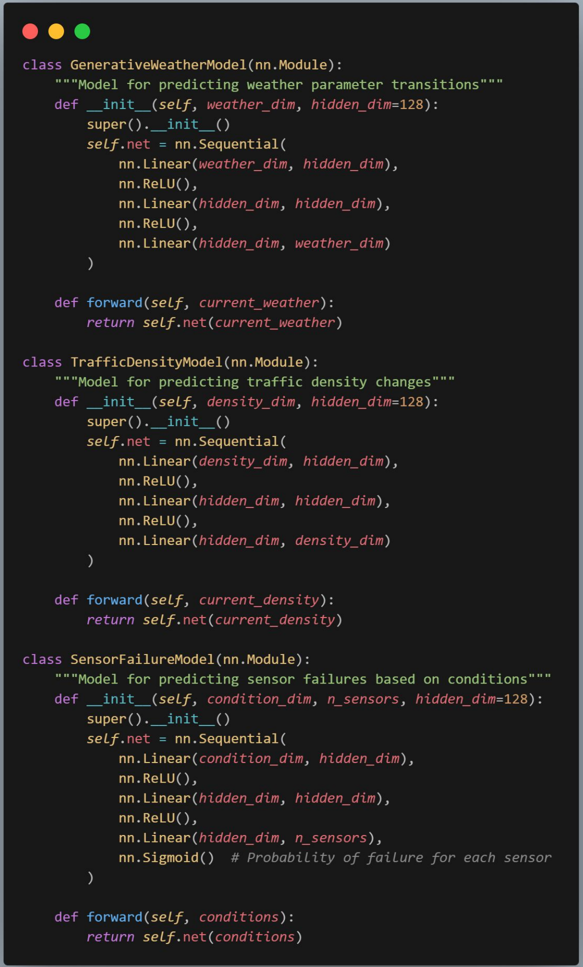
- Weather Modeling: We predict future weather conditions based on current data, using a model trained on historical simulation data.
- Traffic Density Modeling: A model predicts average traffic density from current conditions, trained on simulated traffic data.
- Sensor Failure Modeling: Sensor failures are modeled using historical data, accounting for temporary issues like bright images and persistent ones like broken lenses.
- Runtime Alarms: We simulate runtime alarms based on historical patterns of occurrence and duration.
- Safety and Performance Scoring: These models, combined with a safety verification protocol or surrogate models, calculate safety and performance scores for controllers.



```python
import torch
import torch.nn as nn

class SurrogateModel(nn.Module):
    def __init__(self, state_dim, hidden_dim=256):
        super().__init__()

        # Shared encoder for state information
        self.state_encoder = nn.Sequential(
            nn.Linear(state_dim, hidden_dim),
            nn.ReLU(),
            nn.BatchNorm1d(hidden_dim),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.BatchNorm1d(hidden_dim)
        )

        # Performance score predictor (λp) - predicts average speed
        self.performance_head = nn.Sequential(
            nn.Linear(hidden_dim, hidden_dim//2),
            nn.ReLU(),
            nn.Linear(hidden_dim//2, 1),
            nn.Sigmoid()  # Normalize speed score
        )

        # Safety score predictor (λf) - predicts collision likelihood
        self.safety_head = nn.Sequential(
            nn.Linear(hidden_dim, hidden_dim//2),
            nn.ReLU(),
            nn.Linear(hidden_dim//2, 1),
            nn.Sigmoid()  # Probability of collision
        )

        # Additional cost predictor (λc) if needed
        self.cost_head = nn.Sequential(
            nn.Linear(hidden_dim, hidden_dim//2),
            nn.ReLU(),
            nn.Linear(hidden_dim//2, 1),
            nn.Sigmoid()
        )

        # Hyperparameters for reward combination
        self.alpha1 = nn.Parameter(torch.tensor(1.0))  # Performance weight
        self.alpha2 = nn.Parameter(torch.tensor(1.0))  # Safety weight
        self.alpha3 = nn.Parameter(torch.tensor(1.0))  # Cost weight
```



```python
def forward(self, state):
        """
        Args:
            state: Tensor containing state information including:
                  - Current weather parameters
                  - Traffic density
                  - Sensor status
                  - Monitor states
                  - Vehicle state (speed, position, etc.)
        Returns:
            Dictionary containing:
                - performance_score (λp)
                - safety_score (λf)
                - cost_score (λc)
                - combined_reward (R)
        """
        # Encode state
        encoded_state = self.state_encoder(state)

        # Predict individual scores
        performance_score = self.performance_head(encoded_state)
        safety_score = self.safety_head(encoded_state)
        cost_score = self.cost_head(encoded_state)

        # Calculate combined reward
        # R(st,a) = α1·λp(st,a) - α2·λf(st,a) - α3·λc(st,a)
        reward = (self.alpha1 * performance_score -
                 self.alpha2 * safety_score -
                 self.alpha3 * cost_score)

        return {
            'performance_score': performance_score,
            'safety_score': safety_score,
            'cost_score': cost_score,
            'combined_reward': reward
        }
```



```python
class GenerativeWeatherModel(nn.Module):
    """Model for predicting weather parameter transitions"""
    def __init__(self, weather_dim, hidden_dim=128):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(weather_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, weather_dim)
        )

    def forward(self, current_weather):
        return self.net(current_weather)

class TrafficDensityModel(nn.Module):
    """Model for predicting traffic density changes"""
    def __init__(self, density_dim, hidden_dim=128):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(density_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, density_dim)
        )

    def forward(self, current_density):
        return self.net(current_density)

class SensorFailureModel(nn.Module):
    """Model for predicting sensor failures based on conditions"""
    def __init__(self, condition_dim, n_sensors, hidden_dim=128):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(condition_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, n_sensors),
            nn.Sigmoid()  # Probability of failure for each sensor
        )

    def forward(self, conditions):
        return self.net(conditions)
```

IDEAS TO IMPROVE THE DYNAMIC SIMPLEX

1. **Multi Modal Fusion**

   (i) Add other important sensors as LiDAR , radar which will help us in better distance measurement and object detection
   (ii) Use preprocessing techniques (e.g., noise reduction, normalization) to clean and standardize data from each sensor type.
   (iii) Implement algorithms such as Kalman filters, particle filters, or deep learning-based fusion methods to integrate sensor data.
   (iv) Ensure that the fusion process operates in real-time to support dynamic decision-making.
   (v) Modify the state representation in the semi-Markov decision process (SMDP) to include information derived from fused
   (vi)Adjust the controller switching logic based on insights gained from multi-modal sensor fusion, allowing for more informed decisions regarding when to switch between performant and safety controllers.

   <u>Evaluating this idea</u>
   (i) Test the system's ability to navigate through complex urban environments with high traffic density. The use of fused sensor data can help improve object recognition and tracking of nearby vehicles.
   (ii)Simulate scenarios where one or more sensors fail during operation. Assess how well the system can continue functioning safely by relying on remaining sensors through effective data fusion.
   (iii) Evaluate performance during nighttime driving or low-light conditions where camera visibility may be limited but LIDAR can provide accurate distance measurements.

CONTINUED...

2. **Using High-End GPU**

To surpass the performance of the authors' setup, upgrading to cutting-edge GPUs like the NVIDIA H100 or leveraging multiple GPUs in tandem could significantly reduce computation time. Further optimization of parallel processing, load balancing, and memory management will also be essential to outperform the current benchmark set by the NVIDIA RTX A6000, improving both speed and accuracy in simulations.

GITHUB REPOSITORY

OUR WORK IS AVAILABLE HERE

The End