**Mohammed VI Polytechnic University**

College of Computing

# Symbolic Control for a Mobile Robot

**Team Members**

Yassine Zitouni
Zakariae Jali
Adam Hajjaji
Haytam Rhallab

**Supervisor**

Prof. Adnane Saoud

December 12, 2025

# Contents

# 1 Introduction

The growing complexity of autonomous systems raises a fundamental question: *how can correct behavior be guaranteed under all operating conditions?* Traditional control methods are powerful, but often lack formal guarantees for tasks such as safety, reachability, or mission execution.

Symbolic control addresses this limitation by abstracting continuous dynamics into finite models that can be analyzed using automata theory and formal verification techniques.

In this project, symbolic control is applied to a mobile robot with unicycle dynamics to demonstrate the complete workflow from abstraction to validated controller synthesis.

## 1.1 Motivation

The primary motivation for symbolic control lies in the concept of *correct-by-construction* controllers. Such controllers are synthesized in a way that guarantees satisfaction of the specification by design, rather than through extensive testing or simulation alone.

Compared to classical control approaches, symbolic methods make it possible to:

- provide **formal guarantees** using automata-theoretic and logical tools;
- reason over **finite, discrete models** that can be exhaustively analyzed;
- synthesize controllers that enforce safety and reachability constraints;
- express and handle complex missions involving ordering or conditional logic.

## 1.2 Project Objectives

The main objectives of this project are:

- to construct a finite symbolic abstraction of the mobile robot;
- to formalize safety, reachability, and automaton-based specifications;
- to synthesize controllers satisfying these specifications;
- to validate the resulting controllers through simulation.

# 2 Theoretical Background

Symbolic control provides a framework for designing controllers with formal guarantees by abstracting continuous dynamical systems into finite models. Controller synthesis is then performed on the abstract system using tools from automata theory and graph-based reasoning, before being refined back to the continuous domain.

## 2.1 Symbolic Control Framework

The symbolic control workflow consists of three main steps:

- **Abstraction**: discretization of the state and input spaces to obtain a finite transition system;
- **Synthesis**: computation of a controller on the symbolic model satisfying a given specification;
- **Concretization**: refinement of symbolic actions into continuous control inputs.

This abstraction-based approach enables reasoning over system behavior using finite-state methods while preserving guarantees on the original system.

## 2.2 System Abstraction

We consider a discrete-time nonlinear system

$$x(t+1) = f(x(t), u(t), w(t)), \quad x \in X,\ u \in U,$$

where $w$ represents bounded disturbances. The continuous state space is partitioned into a finite set of symbolic states $\Xi$, and a finite input alphabet $\Sigma$ is selected.

The resulting symbolic model is defined by a transition relation

$$\xi(t+1) \in g(\xi(t), \sigma(t)), \qquad \xi \in \Xi,\ \sigma \in \Sigma,$$

which over-approximates the reachable behavior of the continuous system.

## 2.3 Mobile Robot Model

In addition to the unicycle model, we also consider a simplified planar kinematic model that describes motion in two dimensions without orientation. The dynamics of the 2D model are given by:

$$\begin{cases} x(t+1) = x(t) + \tau\,(u_1(t) + w_1(t)), \\ y(t+1) = y(t) + \tau\,(u_2(t) + w_2(t)), \end{cases}$$

The dynamics of the 3D model are given by:

$$\begin{cases} x_1(t+1) = x_1(t) + \tau(\,u_1(t)\cos(x_3(t)) + w_1(t)), \\ x_2(t+1) = x_2(t) + \tau(\,u_1(t)\sin(x_3(t)) + w_2(t)), \\ x_3(t+1) = x_3(t) + \tau(\,u_2(t) + w_3(t)) \pmod{2\pi}. \end{cases}$$

## 2.4 Reachability Over-Approximation

Symbolic transitions are computed using over-approximations of reachable sets under bounded disturbances. By bounding the effect of uncertainties and state variations, it is possible to conservatively estimate all symbolic successors reachable from a given state and control input.

This guarantees that the symbolic model captures all possible behaviors of the continuous system required for controller synthesis.

### 2.4.1 Derivative-Based Reachability Bounds

To construct the symbolic transition relation, reachable sets are over-approximated using bounds on the system derivatives. Assuming the dynamics $f$ are Lipschitz continuous, there exist constants $D_x$ and $D_w$ such that:

$$\left|\frac{\partial f}{\partial x}\right| \le D_x, \qquad \left|\frac{\partial f}{\partial w}\right| \le D_w.$$

Consider an interval state $[\underline{x}, \bar{x}]$ with center

$$x^* = \frac{\underline{x} + \bar{x}}{2}, \quad \delta_x = \frac{\bar{x} - \underline{x}}{2}.$$

Then, for a fixed control input $u$ and bounded disturbance set $W$, the reachable set satisfies the conservative bound:

$$\text{Reach}([\underline{x}, \bar{x}], u, W) \subseteq \Big[ f(x^*, u, w^*) - D_x\delta_x - D_w\delta_w,\ f(x^*, u, w^*) + D_x\delta_x + D_w\delta_w \Big]. \tag{2.1}$$

This bound guarantees that all possible trajectories originating from the interval state are contained within the symbolic successors.

# 3 Safety Specification and Controller Synthesis

Safety is the requirement that the robot never leaves a designated set of acceptable symbolic states. In our setting, this means the controller must keep the robot inside a region that excludes obstacles and boundary-violating cells.

## 3.1 Safety as a Set-Based Property

Let $\Xi$ be the set of symbolic states and let $Q_{\text{safe}} \subseteq \Xi$ be the subset of states considered safe. A symbolic trajectory is a sequence $(\xi_0, \xi_1, \xi_2, \dots)$ with

$$\xi_{t+1} \in g(\xi_t, \sigma_t), \qquad \sigma_t \in \Sigma.$$

A *safety specification* asks that all visited states remain in $Q_{\text{safe}}$:

$$\text{Safety:} \quad \forall t \geq 0,\ \xi_t \in Q_{\text{safe}}.$$

Intuitively, once the system starts in a safe cell, the controller must never allow a step that would push the abstract state into an obstacle cell, outside the workspace, or into any region we classified as forbidden.

**Safety in three viewpoints.**

1. **Logical view:** "Nothing bad ever happens" – the robot never visits a forbidden symbolic cell.
2. **Graph view:** the controller chooses edges of the symbolic graph so that the path always stays inside $Q_{\text{safe}}$.
3. **Practical view:** safe cells correspond to feasible poses of the robot with enough distance from obstacles and boundaries.

## 3.2 Predecessor Operator

A key ingredient in the construction of a safety controller is the *predecessor operator*, which tells us from which states we can safely enter a given region.

For any set $R \subseteq \Xi$, we define:

$$\mathrm{Pre}(R) = \left\{ \xi \in \Xi \;\middle|\; \exists \sigma \in \Sigma \text{ such that } g(\xi, \sigma) \subseteq R \right\}.$$

A state belongs to $\mathrm{Pre}(R)$ if there exists at least one input $\sigma$ that keeps all its successors inside $R$ in one step.

## 3.3 Fixed-Point Construction of the Safe Region

The goal of the safety controller is to determine the largest set of symbolic states from which the robot can be kept inside the safe region $Q_{\text{safe}}$ for all time. States that cannot be controlled to remain inside $Q_{\text{safe}}$ are gradually removed, until only the states that admit a safe control action remain.

We start from the full safe region and repeatedly apply the predecessor operator:

$$R_0 = Q_{\text{safe}}, \qquad R_{k+1} = Q_{\text{safe}} \cap \mathrm{Pre}(R_k).$$

This iterative process eliminates unsafe or "uncontrollable" symbolic cells. Once the set stops changing, we reach the fixed point

$$R^\star = R_k,$$

which represents the maximal set of states from which safety can be guaranteed.

**Algorithm 1** Fixed-Point Safety Controller Construction

**Input:** Symbolic model $(\Xi, \Sigma, g)$, safe region $Q_{\text{safe}}$
**Output:** Maximal safe domain $R^\star$ and controller $C_{\text{safe}}$

$R_0 \leftarrow Q_{\text{safe}}$
$k \leftarrow 0$
**repeat**
    $R_{k+1} \leftarrow Q_{\text{safe}} \cap \mathrm{Pre}(R_k)$
    $k \leftarrow k + 1$
**until** $R_k = R_{k-1}$
$R^\star \leftarrow R_k$
**for all** $\xi \in R^\star$ **do**
    choose $\sigma \in \Sigma$ such that $g(\xi, \sigma) \subseteq R^\star$
    $C_{\text{safe}}(\xi) \leftarrow \sigma$
**end for**
**return** $(R^\star, C_{\text{safe}})$

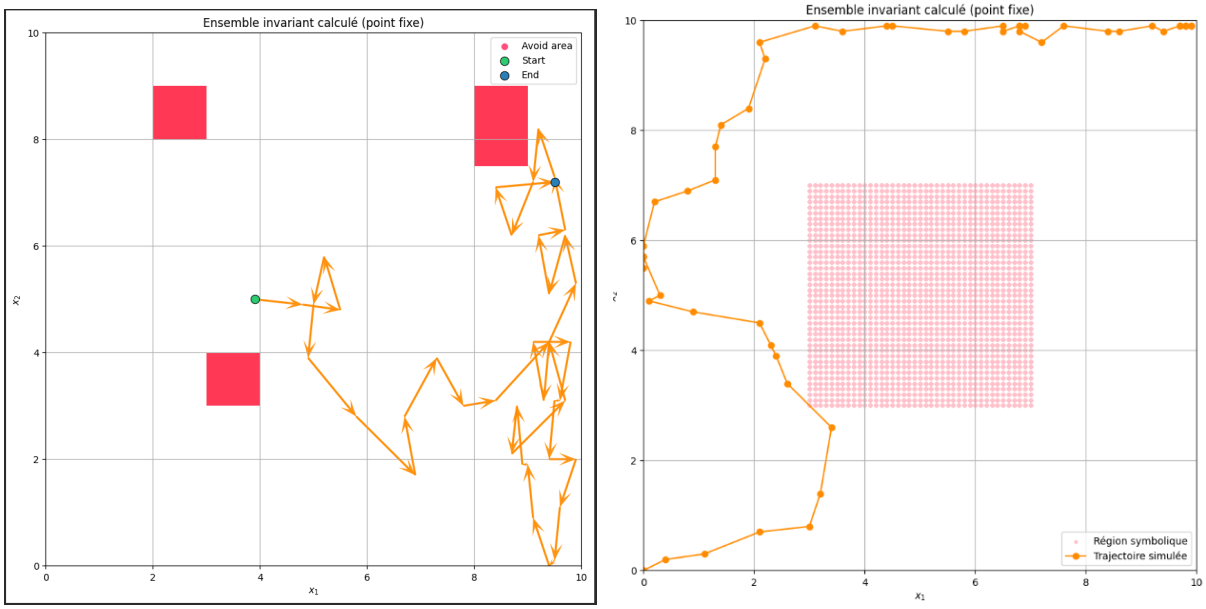
The resulting set $R^\star$ is *controlled invariant*: starting from any initial symbolic state $\xi_0 \in R^\star$,

repeatedly applying the safety controller $C_{\text{safe}}$ ensures that the entire trajectory remains in $R^\star$ — and therefore in $Q_{\text{safe}}$ — for all time.

## 3.4 Practical Issue Encountered

During our implementation, we observed that $R^\star$ could become very small or even empty when the set of available controls was too restrictive. For instance, allowing only positive values for the forward velocity $u_1$ made it difficult for the controller to "escape" from configurations pointing toward obstacles.

To address this, we extended the input set to include additional actions (e.g., smaller velocities or negative increments), which provided more options for steering the robot back toward safe regions. This simple modification significantly enlarged $R^\star$ and produced a more robust safety controller.



# 4 Reachability Controller Synthesis

Reachability focuses on ensuring that the robot eventually enters a designated goal region while avoiding invalid states along the way. Unlike the safety objective, which forbids entering certain regions, reachability encourages the system to move *towards* a target set of symbolic states.

**What Reachability Guarantees.** A controller satisfies a reachability objective if all closed-loop trajectories eventually intersect the target region $Q_{\text{goal}} \subseteq \Xi$. Formally, the set of admissible trajectories is:

$$R = \left\{ \tau : \mathbb{N} \to \Xi \;\middle|\; \exists t \in \mathbb{N},\ \tau(t) \in Q_{\text{goal}} \right\}.$$

## 4.1 Backward Expansion of the Target Region

To compute which symbolic states can eventually reach $Q_{\text{goal}}$, we apply a backward search through the transition system. Starting from the goal region, we iteratively enlarge a set of states that can lead to it.

The construction follows the recursive update:

$$R_{k+1} = R_k \cup \text{Pre}(R_k),$$

where $\text{Pre}(R)$ denotes all symbolic states with at least one successor inside $R$.

**Backward Reachability Algorithm**

- **Initialization:** $R_0 = Q_{\text{goal}}$
- **Iteration:** $R_{k+1} = R_k \cup \text{Pre}(R_k)$
- **Termination:** stop once $R_{k+1} = R_k$

When the process stabilizes, the limit set $R^\star$ represents all symbolic states from which the target can be reached at least once.

## 4.2 Constructing a Reachability Controller

Once $R^\star$ is built, constructing a controller is straightforward: for every state $\xi \in R^\star$, we select an action that moves the robot closer to the goal region.

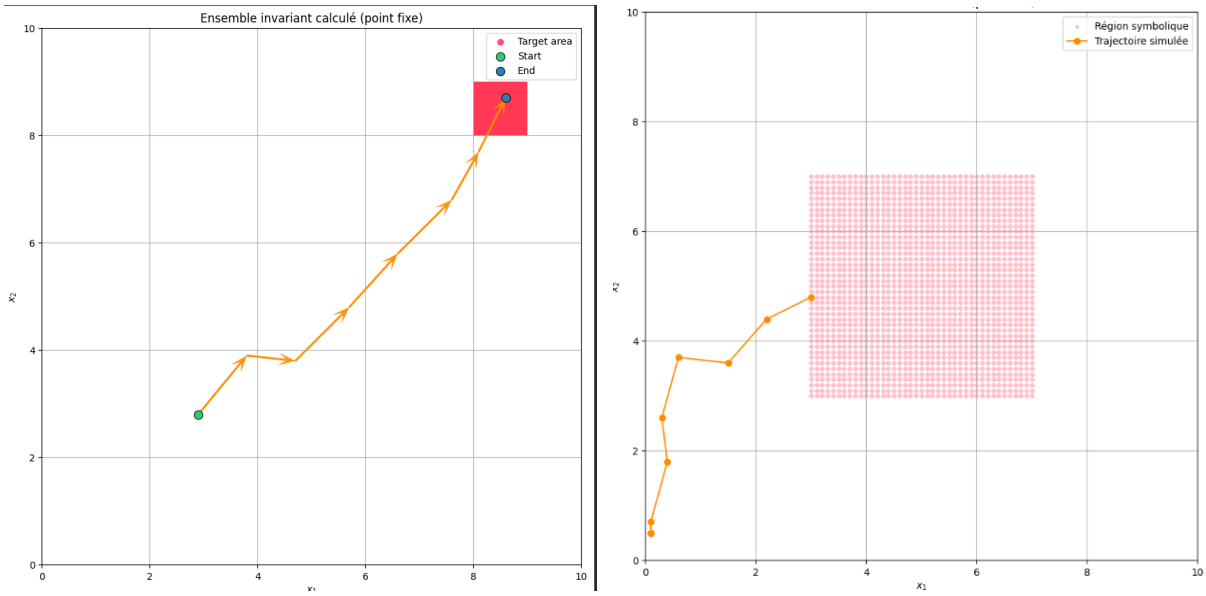
A reachability controller must satisfy:

$$g(\xi, \sigma(\xi)) \cap R^\star \neq \emptyset, \qquad \forall\, \xi \in R^\star.$$

To guide the robot efficiently toward the target, we choose control inputs that:

- avoid symbolic successors outside $R^\star$,
- reduce unnecessary detours,
- encourage monotonic progress toward the goal region.

This ensures that the robot does not wander through irrelevant symbolic states.



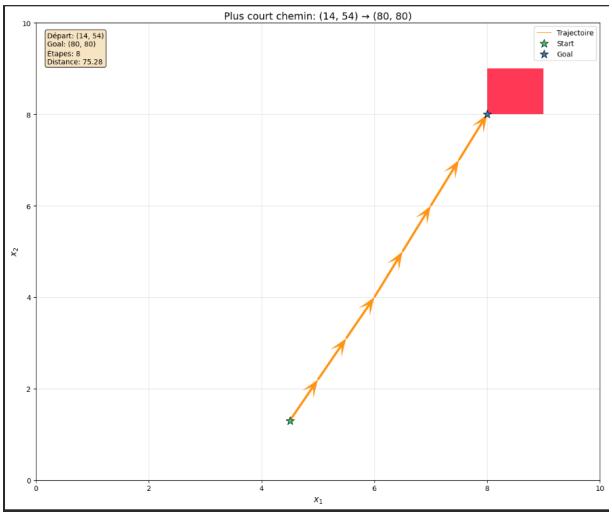
# 5 Dijkstra Optimization for Reachability

The reachability controller guarantees that the robot eventually reaches the target region, but it does not impose any notion of optimality on the generated paths. In practice, this led to unstable and irregular trajectories, since the controller may arbitrarily choose among multiple valid symbolic successors.

To improve path quality, we introduced an optimization step based on Dijkstra's shortest-path algorithm. Unlike Breadth-First Search, which minimizes only the number of symbolic transitions, Dijkstra minimizes the true geometric distance to the target, resulting in more efficient and realistic paths.

The algorithm is applied on the symbolic graph restricted to the reachability set $R^\star$, with edge weights defined by the Euclidean distance between symbolic cell centers. Thanks to the non-negative weights, the search terminates as soon as a target state is extracted from the priority queue, guaranteeing optimality while reducing computation time.

This optimization preserves the correctness of the reachability controller and produces smooth, direct trajectories toward the goal, as illustrated in the corresponding figures.



# 6 Safety ∩ Reachability

In many tasks, the robot must satisfy safety and reachability simultaneously: it must remain in a safe region at all times while eventually reaching a target. This combined objective cannot be enforced by a simple intersection of the independently synthesized controllers.

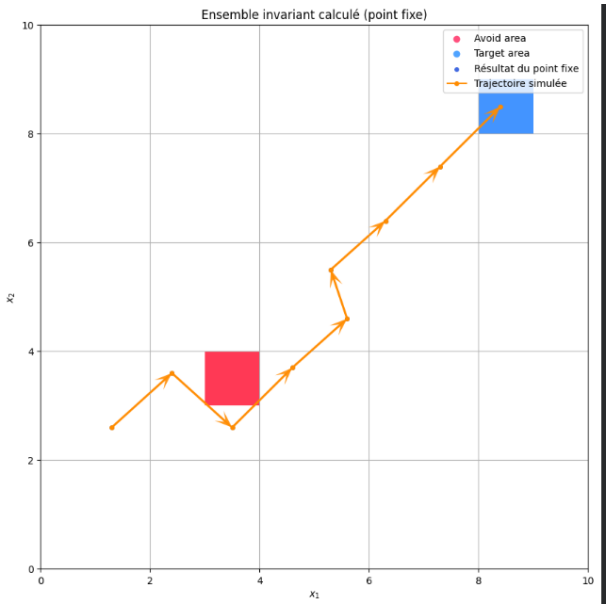
## 6.1 Key Idea

Safety is treated as a hard constraint, while reachability is enforced only within the safe region. A naive intersection may create deadlock states, where no control action is both safe and goal-directed.

To avoid this issue, a fixed-point computation is performed starting from the safe target set:

$$R_0 = Q_{\text{safe}} \cap Q_{\text{target}}, \qquad R_{k+1} = Q_{\text{safe}} \cap \text{Pre}(R_k).$$

## 6.2 Result



# 7 Automata-Based Control

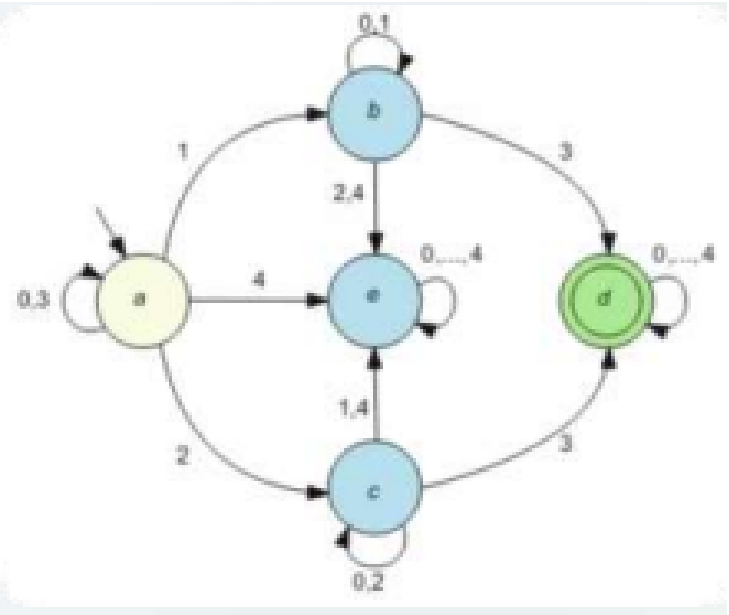
While safety and reachability specifications allow us to constrain the robot's motion in space, they are not sufficient to express more complex missions involving ordered objectives, conditional behaviors, or repeated tasks. To address this limitation, we introduce an automata-based control layer that enables the formal specification and execution of high-level missions.

Automata theory provides a natural framework to encode such specifications, by representing mission progress as transitions between discrete states driven by the robot's interaction with its environment.

## 7.1 Automaton Specification

Each region of interest in the workspace is associated with a symbolic label. These labels are used as inputs to a finite automaton whose states represent the progression of the mission. Transitions between automaton states are triggered when the robot enters specific regions or satisfies predefined conditions.

In this setting, the automaton encodes the desired sequence of actions or goals, such as visiting regions in a particular order while avoiding forbidden areas. This allows the specification to capture temporal constraints that cannot be expressed using reachability alone.



## 7.2 Controller with Memory

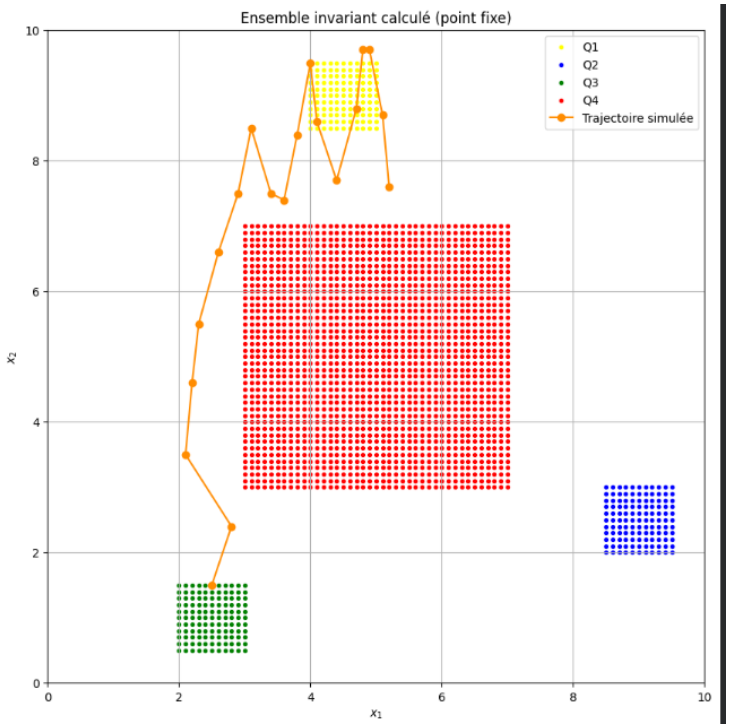
To enforce the automaton specification, the symbolic controller is augmented with a discrete memory variable corresponding to the current automaton state. At each time step, the control action is selected based on both the symbolic state of the robot and the automaton state.

This coupling ensures that only actions compatible with the current stage of the mission are allowed. When the robot reaches a region that triggers an automaton transition, the memory is updated accordingly, and the controller adapts its behavior to satisfy the next part of the specification.

## 7.3 Execution and Results

The resulting automata-based controller guarantees that the robot respects both the low-level motion constraints and the high-level mission logic. Simulation results demonstrate that the robot successfully follows the specified sequence of regions while remaining safe and eventually completing the mission, as shown in the trajectory plots.

This approach illustrates how automata can be effectively combined with symbolic control to achieve expressive, correct-by-construction robot behavior.



# 8 Automaton-Based Mission Control: Mars Rover Example

Automata-based control is particularly well suited for missions that require the robot to execute a sequence of objectives in a specific order, while continuously respecting safety constraints. To illustrate this capability, we consider a Mars rover mission involving exploration, data collection, and safe return.

In this scenario, the rover must first visit a science zone to collect samples, then move to an imaging zone to acquire panoramic pictures, and finally return to the base. Throughout the mission, hazardous regions must be avoided at all times. Such a specification cannot be expressed using a single reachability or safety objective, as it inherently involves ordering and progression.

## 8.1 Mission Automaton Mars

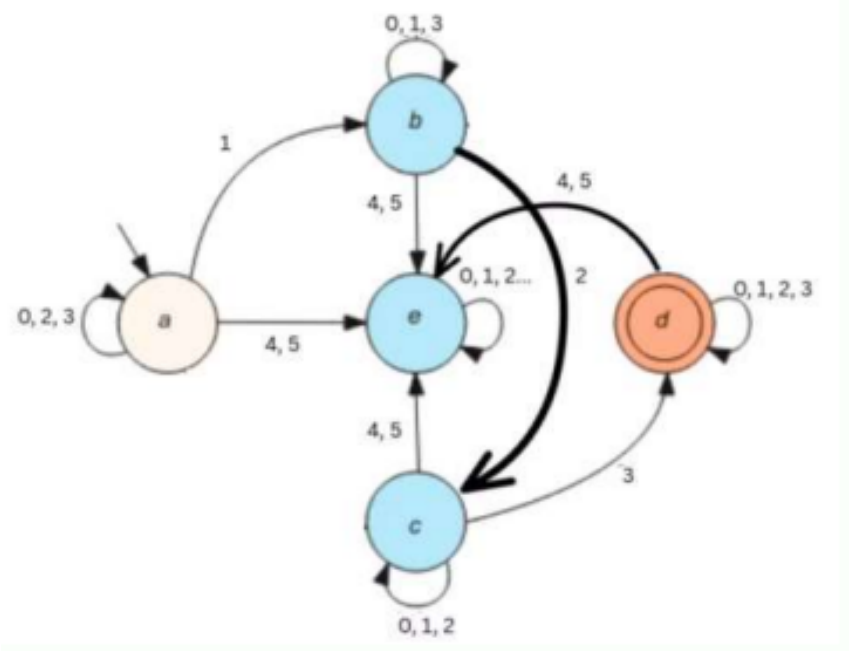
The mission is encoded as a finite automaton whose states represent the current stage of the task. Each automaton state corresponds to a logical phase of the mission, such as initialization, exploration of the first zone, exploration of the second zone, and mission completion.

Transitions between automaton states are triggered by the robot entering specific regions of the workspace. Once a transition occurs, the automaton moves to the next state, enforcing the correct order of execution. This structure guarantees that mission objectives are completed sequentially and cannot be skipped or repeated arbitrarily.

## 8.2 Coupling with the Symbolic Controller

To enforce the automaton specification, the symbolic controller is augmented with a discrete memory variable representing the current automaton state. At each time step, control actions are selected based on both the robot's symbolic state and the automaton state.

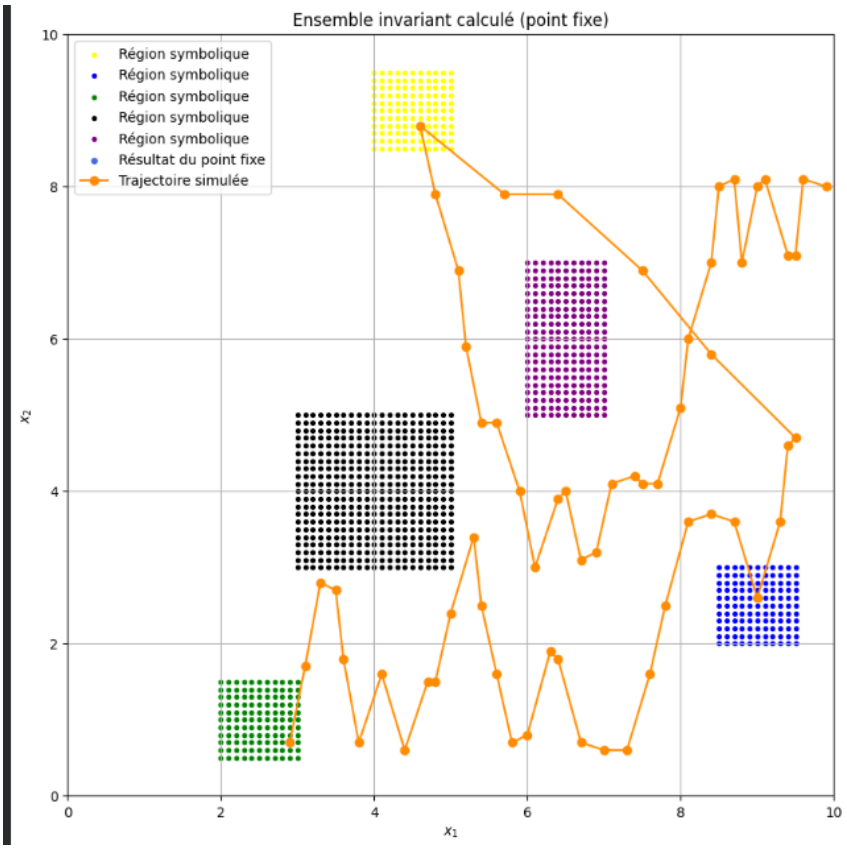
This coupling ensures that only actions compatible with the current phase of the mission are allowed. As the rover progresses and triggers automaton transitions, the controller dynamically adapts its behavior to guide the robot toward the next objective.




## 8.3 Execution and Trajectory Analysis

The resulting closed-loop behavior combines low-level motion constraints with high-level mission logic. As shown in the automaton and trajectory figures, the rover successfully visits the required zones in the specified order, avoids hazardous regions, and completes the mission without violating safety constraints.

This example demonstrates how automata-based control extends symbolic control beyond simple objectives, enabling the formal and reliable execution of complex robotic missions.


Ensemble invariant calculé (point fixe)
Région symbolique
Région symbolique
Région symbolique
Région symbolique
Région symbolique
Résultat du point fixe
Trajectoire simulée
$x_1$
$x_2$
0
2
4
6
8
10