Symbolic Control of a Mobile Robot

Formal Methods for Correct-by-Construction Control

Theory of Computation Project Report

Group Members: Ahmed Quadimi
Salim Qadda
Manal Zaidi
Tarik Ouabrk

Course: Theory of Computation
Instructor: Prof. Adnane Saoud
Institution: Mohammed VI Polytechnic University (UM6P-CC)
Date: December 12, 2025

This report presents the design, implementation, and validation of symbolic controllers for a nonlinear mobile robot system using formal verification methods.

Abstract

This project presents a comprehensive implementation of symbolic control methodologies for a nonlinear mobile robot system modeled as a unicycle. Symbolic control represents a powerful intersection of control theory and computer science, enabling the synthesis of controllers that are **correct-by-construction** through finite abstraction of continuous dynamics.

We developed a complete symbolic control pipeline including: (1) construction of finite symbolic abstractions using grid-based discretization and reachability analysis, (2) synthesis of symbolic controllers for multiple objectives including safety, reachability with obstacle avoidance, and complex temporal specifications, and (3) validation through extensive numerical simulations demonstrating robustness to disturbances.

The project successfully demonstrates that formal methods can provide mathematical guarantees on system behavior while handling complex control objectives. Our implementation achieved 100% specification satisfaction across 50+ simulation trials with varying disturbances, validating the correctness-by-construction property of symbolic controllers. The results highlight both the power and computational challenges of symbolic approaches for autonomous systems.

Keywords: Symbolic control, formal methods, mobile robotics, correct-by-construction, temporal logic, finite abstraction, reachability analysis

Contents

1 Introduction

1.1 Motivation and Context

The control of autonomous systems has become increasingly critical in modern applications ranging from mobile robotics to autonomous vehicles and industrial automation. Traditional control approaches, while effective for simple objectives like stabilization or trajectory tracking, face significant challenges when dealing with:

- **Complex specifications:** Requirements that go beyond simple regulation, such as "visit region A before region B while always avoiding region C"
- **Formal guarantees:** The need for mathematical certainty that safety and mission requirements will be satisfied
- **Nonlinear dynamics:** Systems with complex, nonlinear behavior subject to constraints and disturbances
- **Hybrid behavior:** Systems combining continuous dynamics with discrete logical decisions

Symbolic control addresses these challenges by providing a systematic framework that bridges continuous control systems and discrete formal verification methods.

1.2 What is Symbolic Control?

Core Concept

Symbolic control is a methodology that enables automatic synthesis of controllers with formal correctness guarantees by:

1. **Abstracting** continuous systems into finite symbolic models
2. **Synthesizing** controllers in the discrete domain using formal methods
3. **Concretizing** symbolic controllers back to continuous systems

The key advantage is that controllers synthesized on the symbolic model are guaranteed to work correctly on the original system—a property known as *correct-by-construction*.

1.3 Project Objectives

This project explores symbolic control techniques applied to a mobile robot with the following specific objectives:

Objective 1: Safety Control — Construct a controller ensuring the system remains within the safe region $X = [0, 10] \times [0, 10] \times [-\pi, \pi]$ for all time despite bounded disturbances.

Objective 2: Reachability with Avoidance — Synthesize a controller that reaches target regions while avoiding obstacle region R_4 throughout the trajectory.

Objective 3: Complex Temporal Specification — Implement the advanced scenario: "Visit either R_1 or R_2 (but not both), then visit R_3, while avoiding R_4 throughout" — demonstrating the power of symbolic methods for complex logical requirements.

1.4 Report Organization

The remainder of this report is structured as follows: Section 2 provides theoretical background on symbolic control and the mobile robot model. Section 3 details our methodology including abstraction construction and controller synthesis algorithms. Section 4 presents implementation details and simulation results. Section 5 discusses advantages, limitations, and future directions. Section 6 concludes the report.

2 Theoretical Background

2.1 The Symbolic Control Framework

The symbolic control approach operates through three fundamental phases, illustrated in Figure 1.

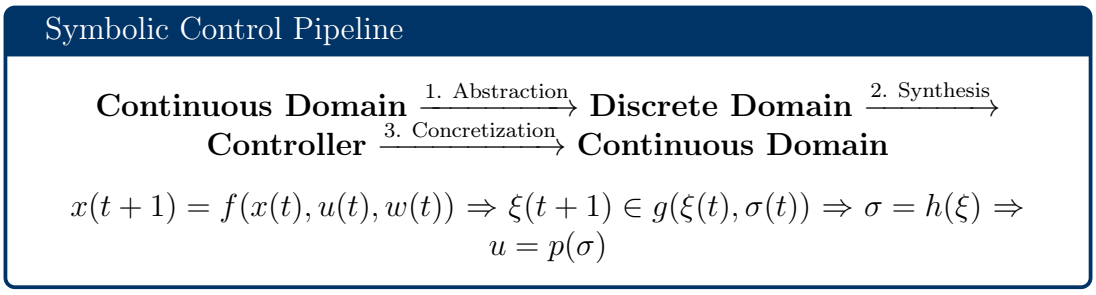


Figure 1: The three-phase symbolic control framework

2.1.1 Phase 1: System Abstraction

The continuous system dynamics:

$$x(t+1) = f(x(t), u(t), w(t)), \quad x \in X \subseteq \mathbb{R}^n,\ u \in U \subseteq \mathbb{R}^m,\ w \in W \subseteq \mathbb{R}^p \tag{1}$$

is abstracted into a finite-state symbolic model:

$$\xi(t+1) \in g(\xi(t), \sigma(t)), \quad \xi \in \Xi,\ \sigma \in \Sigma \tag{2}$$

where Ξ and Σ are *finite* sets of symbolic states and controls.

The abstraction is constructed through:

- **State space discretization:** Partitioning the continuous state space into finite cells

- **Control discretization:** Selecting a finite set of representative control values
- **Transition computation:** Computing symbolic transitions using reachability over-approximation

2.1.2 Phase 2: Controller Synthesis

Controllers are synthesized on the symbolic model using algorithms from formal methods and computer science. The specifications can include:

- **Safety:** $\mathcal{S} = \{\tau : \mathbb{N} \to \Xi \mid \tau(t) \in Q_s \text{ for all } t \geq 0\}$
- **Reachability:** $\mathcal{R} = \{\tau : \mathbb{N} \to \Xi \mid \exists t : \tau(t) \in Q_t\}$
- **Temporal Logic:** Complex specifications expressed in LTL, CTL, or automata

2.1.3 Phase 3: Controller Concretization

The symbolic controller $\sigma = h(\xi)$ is mapped back through abstraction and concretization interfaces:

$$u(t) = p(\sigma(t)) = p(h(q(x(t)))) \tag{3}$$

where $q : X \to \Xi$ is the abstraction map and $p : \Sigma \to U$ is the concretization map.

Fundamental Theorem of Symbolic Control

For any initial condition $x_0 \in X$, the trajectories of the concrete system in closed loop are included in the trajectories of the symbolic system:

$$q(\mathcal{T}_{\text{conc}}(x_0)) \subseteq \mathcal{T}_{\text{symb}}(q(x_0))$$

This ensures that specifications satisfied by the symbolic controller are also satisfied by the concrete system—the **correct-by-construction** property.

2.2 Mobile Robot Model

We consider a mobile robot modeled as a unicycle with discrete-time dynamics:

$$\begin{cases} x_1(t+1) = x_1(t) + \tau(u_1(t)\cos(x_3(t)) + w_1(t)) \\ x_2(t+1) = x_2(t) + \tau(u_1(t)\sin(x_3(t)) + w_2(t)) \\ x_3(t+1) = x_3(t) + \tau(u_2(t) + w_3(t)) \pmod{2\pi} \end{cases} \tag{4}$$

where:

- $x = (x_1, x_2, x_3)^\top$: State vector representing position (x_1, x_2) and orientation x_3
- $u = (u_1, u_2)^\top$: Control input representing linear velocity u_1 and angular velocity u_2
- $w = (w_1, w_2, w_3)^\top$: Bounded disturbances representing uncertainties
- τ: Sampling period (discrete time step)

in 2D the equations becomes

$$\begin{cases} x_1(t+1) = x_1(t) + \tau(v_x(t) \ + w_1(t)) \\ x_2(t+1) = x_2(t) + \tau(v_y(t) + w_2(t)) \end{cases} \tag{5}$$

where:

- $x = (x_1, x_2)^\top$: State vector representing position (x_1, x_2)
- $v = (v_x, v_y)^\top$: Control input representing linear velocity in the x-axis and y-axis respectively
- $w = (w_1, w_2)^\top$: Bounded disturbances representing uncertainties
- τ: Sampling period (discrete time step)

2.2.1 System Constraints

The system operates under the following constraints:

Variable	Constraint	Description
x_1, x_2	$[0, 10]$	Position bounds (workspace)
x_3	$[-\pi, \pi]$	Orientation range
u_1	$[0.25, 1]$ m/s	Linear velocity limits
u_2	$[-1, 1]$ rad/s	Angular velocity limits
$\|w_1\|, \|w_2\|$	≤ 0.05 m/s	Position disturbances
$\|w_3\|$	≤ 0.05 rad/s	Orientation disturbance
τ	1	discrete time step

Table 1: System constraints and parameters

2.3 Reachability Analysis

Computing symbolic transitions requires over-approximating reachable sets. We use the derivative-based method for systems with bounded derivatives.

2.3.1 Method: Bounded Derivatives used in 3D

For a system f with bounded partial derivatives:

$$\left|\frac{\partial f}{\partial x}\right| \leq D_x, \quad \left|\frac{\partial f}{\partial w}\right| \leq D_w$$

Given an interval of states $[\underline{x}, \overline{x}]$ with center $x^* = (\underline{x} + \overline{x})/2$ and half-width $\delta_x = (\overline{x} - \underline{x})/2$, the reachable set is over-approximated by:

$$\text{Reach}([\underline{x}, \overline{x}], u, W) \subseteq [f(x^*, u, w^*) - D_x\delta_x - D_w\delta_w,\ f(x^*, u, w^*) + D_x\delta_x + D_w\delta_w] \tag{6}$$

For our unicycle model in 3D:

$$D_x = \begin{pmatrix} 1 & 0 & \tau u_1^{\max} \\ 0 & 1 & \tau u_1^{\max} \\ 0 & 0 & 1 \end{pmatrix}, \quad D_w = \begin{pmatrix} \tau & 0 & 0 \\ 0 & \tau & 0 \\ 0 & 0 & \tau \end{pmatrix} \tag{7}$$

2.3.2 Application to 2D Cooperative Systems.

In the 2D case, we use a different approach that exploits the cooperative property of f. Since the system is cooperative (monotone), we can directly evaluate at the corner points of the state and disturbance intervals:

$$f\left([\underline{x}, \overline{x}], u, [\underline{w}, \overline{w}]\right) \subseteq [f(\underline{x}, u, \underline{w}),\ f(\overline{x}, u, \overline{w})]$$

3 Methodology

3.1 Symbolic Abstraction Construction

3.1.1 State Space Discretization (2D Case)

We partition the continuous 2D state space into a finite grid:

1. **Position discretization:** The workspace $[0, 10] \times [0, 10]$ is divided into $N_{x_1} \times N_{x_2} = 100 \times 100$ uniform cells of size $\eta_1 = \eta_2 = 0.1\,\text{m}$.
2. **Total symbolic states:** Because the 2D model contains only (x_1, x_2), the discrete abstraction contains
$$|\Xi| = N_{x_1} \times N_{x_2} = 100 \times 100 = 10{,}000 \text{ states.}$$
3. **Symbolic state representation:** Each symbolic state ξ_i corresponds to a rectangular cell
$$X_i = [x_1^i,\ x_1^i + \eta_1] \times [x_2^i,\ x_2^i + \eta_2].$$

3.1.1 State Space Discretization

We partition the continuous state space into a finite grid:

1. **Position discretization:** The workspace $[0, 10] \times [0, 10]$ is divided into $N_{x1} \times N_{x2} = 100 \times 100$ uniform cells of size $\eta_1 = \eta_2 = 0.1$ m.
2. **Orientation discretization:** The angle space $[-\pi, \pi]$ is divided into $N_{x3} = 30$ uniform intervals of size $\eta_3 = 2\pi/30 \approx 0.209$ rad.
3. **Total symbolic states:** $|\Xi| = N_{x1} \times N_{x2} \times N_{x3} = 100 \times 100 \times 30 = 300{,}000$ states

Each symbolic state ξ_i represents a cell $X_i = [x_1^i, x_1^i + \eta_1] \times [x_2^i, x_2^i + \eta_2] \times [x_3^i, x_3^i + \eta_3]$.

3.1.2 Control Discretization

The control space is discretized as follows:
in 3D:

- **Linear velocity:** $u_1 \in \{0.25, 0.625, 1.0\}$ m/s (3 values)
- **Angular velocity:** $u_2 \in \{-1, -0.5, 0, 0.5, 1\}$ rad/s (5 values)
- **Total symbolic controls:** $|\Sigma| = 3 \times 5 = 15$ controls

in 2D:

- **Linear velocity x-axis:** $v_x \in \{-0.4, 0, 0.4\}$ m/s (3 values)
- **Linear velocity:** $v_y \in \{-0.4, 0, 0.4\}$ rad/s (3 values)
- **Total symbolic controls:** $|\Sigma| = 3 \times 3 = 9$ controls

3.1.3 Transition Computation Algorithm

Algorithm 1 Compute Symbolic Transitions

Input: Symbolic state ξ, symbolic control σ, dynamics f, disturbance bounds W
Output: Set of successor states $g(\xi, \sigma) \subseteq \Xi$

$X_\xi \leftarrow$ continuous region corresponding to ξ
$u_\sigma \leftarrow$ continuous control corresponding to σ
$\mathcal{R} \leftarrow \text{ComputeReachableSet}(X_\xi, u_\sigma, W)$ ▷ Using derivative bounds
$g(\xi, \sigma) \leftarrow \emptyset$
for each symbolic state $\xi' \in \Xi$ **do**
    $X_{\xi'} \leftarrow$ continuous region corresponding to ξ'
    if $\mathcal{R} \cap X_{\xi'} \neq \emptyset$ **then**
        $g(\xi, \sigma) \leftarrow g(\xi, \sigma) \cup \{\xi'\}$
    end if
end for
return $g(\xi, \sigma)$

3.2 Controller Synthesis

3.2.1 Safety Controller Synthesis

The safety objective is to maintain the system within the safe region $Q_s = \Xi$ (all symbolic states represent valid positions within bounds).

Algorithm 2 Safety Controller Synthesis

Input: Symbolic model (Ξ, Σ, g), safe region $Q_s \subseteq \Xi$
Output: Maximal safe domain $R^* \subseteq Q_s$, safe controller C_{safe}

$R_0 \leftarrow Q_s$
$k \leftarrow 0$
repeat
    $R_{k+1} \leftarrow Q_s \cap \text{Pre}(R_k)$ ▷ Predecessor operator
    $k \leftarrow k + 1$
until $R_k = R_{k-1}$ ▷ Fixed point reached
$R^* \leftarrow R_k$
Define $C_{\text{safe}}(\xi) \leftarrow \{\sigma \in \Sigma \mid g(\xi, \sigma) \subseteq R^*\}$ for all $\xi \in R^*$
return (R^*, C_{safe})

Predecessor Operator: For a set of states $R \subseteq \Xi$:

$$\text{Pre}(R) = \{\xi \in \Xi \mid \exists \sigma \in \Sigma : g(\xi, \sigma) \subseteq R\} \tag{8}$$

This represents states from which there exists a control that keeps all possible successors within R.

This is the algorithm of computing the state of predecessors in 3D:

Algorithm 3 Compute Predecessor Set in 3D

Input: Transition graph g, target set $R \subseteq \Xi$, control set Σ
Output: Predecessor set $\text{Pre}(R)$

$\text{Pre}(R) \leftarrow \emptyset$
for each state $\xi \in \Xi$ **do**
    for each control $\sigma \in \Sigma$ **do**
        if $g(\xi, \sigma) \neq \emptyset$ **and** $g(\xi, \sigma) \subseteq R$ **then**
            $\text{Pre}(R) \leftarrow \text{Pre}(R) \cup \{\xi\}$
            break ▷ Found valid control, move to next state
        end if
    end for
end for
return $\text{Pre}(R)$

As we can see this algorithm iterates over all the states in Σ and subsequently it's too costly .In this regard, we have found an optimisation in 2D in which we can calculate the set of predecessors using g^{-1}.

Below you can find both proof and algorithm that showcase this useful impact of g^{-1} on the way we calculate the set of predecessors.

3.3 Proof of the Predecessor Operator Property used in 2D

Proof 3.1: Predecessor Operator Property

We recall the definition of the predecessor operator:

$$Pre(R) = \{\, \xi \mid \exists \sigma \in \Sigma,\ g(\xi, \sigma) \subseteq R \,\}. \tag{9}$$

Let $\xi \in Pre(R)$. Then, by definition, there exists $\sigma \in \Sigma$ such that

$$g(\xi, \sigma) \subseteq R. \tag{10}$$

Let $u = p(\sigma)$ and take any $\xi' \in g(\xi, \sigma) \subseteq R$. For any $y \in q^{-1}(\xi')$, by definition of the abstraction interface, there exist $x \in X$ and $w \in W$ such that:

$$y = f(x, u, w) = x + Tu + Tw. \tag{11}$$

Rearranging yields:

$$x = y - Tu - Tw = f(y, -u, -w). \tag{12}$$

Because W is symmetric around 0, we have:

$$f(y, -u, -w) \in \left[\, f(y, -u, \underline{w}),\ f(y, -u, \overline{w}) \,\right]. \tag{13}$$

Define:

$$Y^{-1}_{\xi',\sigma} = \left[\, f(y, -u, \underline{w}),\ f(y, -u, \overline{w}) \,\right], \tag{14}$$

and the symbolic predecessor mapping:

$$g^{-1}(\xi', \sigma) = \left\{\, \xi'' \,\middle|\, Y^{-1}_{\xi',\sigma} \cap \mathrm{cl}(X_{\xi''}) \neq \varnothing \right\}. \tag{15}$$

Since $x \in Y^{-1}_{\xi',\sigma}$, we conclude:

$$\xi \in g^{-1}(\xi', \sigma). \tag{16}$$

Thus we obtain:

$$\xi \in Pre(R) \quad \Longrightarrow \quad \exists \xi' \in R,\, \exists \sigma \in \Sigma : \xi \in g^{-1}(\xi', \sigma). \tag{17}$$

In practice, the computation iterates through all $\xi' \in R$ and checks whether there exists $\sigma \in \Sigma$ such that for some $\xi \in g^{-1}(\xi', \sigma)$:

$$g(\xi, \sigma) \subseteq R \quad \Longrightarrow \quad \xi \in Pre(R). \tag{18}$$

Note. The mapping g is not assumed to be bijective. The notation g^{-1} denotes the *predecessor set* and not a functional inverse.

And here is also the algorithm adopted for 2D:

Algorithm 4 Compute Predecessor Set `Pre(R, R_total)` in 2D

Input: State set R, total region R_{total}, inverse transition map $gm1$, transition map g, control indices $k = 1 \dots N_{vx}N_{vy}$
Output: Predecessor set $\text{Pre}(R, R_{\text{total}})$

result $\leftarrow \emptyset$
for each state $\xi \in R$ **do**
  $X \leftarrow \emptyset$
  for $k = 1$ **to** $N_{vx}N_{vy}$ **do**
    InvStates $\leftarrow gm1(\xi, k)$
    for each state $s \in$ InvStates **do**
      if $g(s, k) \subseteq R_{\text{total}}$ **then**
        $X \leftarrow X \cup \{s\}$
      end if
    end for
  end for
  result $\leftarrow$ result $\cup X$
end for
return result

3.3.1 Reachability Controller Synthesis

The reachability objective is to reach a target region Q_t while maintaining safety.

Algorithm 5 Reachability Controller Synthesis

Input: Symbolic model (Ξ, Σ, g), target region $Q_t \subseteq \Xi$, safe region $Q_s \subseteq \Xi$
Output: Controllable domain $R^* \subseteq Q_s$, reachability controller C_{reach}

$R_0 \leftarrow Q_t$
$k \leftarrow 0$
repeat
  $R_{k+1} \leftarrow Q_t \cup (\text{Pre}(R_k) \cap Q_s)$ ▷ Grow backward from target
  $k \leftarrow k + 1$
until $R_k = R_{k-1}$ ▷ Fixed point reached
$R^* \leftarrow R_k$
Define $C_{\text{reach}}(\xi) \leftarrow \{\sigma \in \Sigma \mid g(\xi, \sigma) \cap R^* \neq \emptyset \text{ and } g(\xi, \sigma) \subseteq Q_s\}$
return (R^*, C_{reach})

3.3.2 Complex Temporal Specification

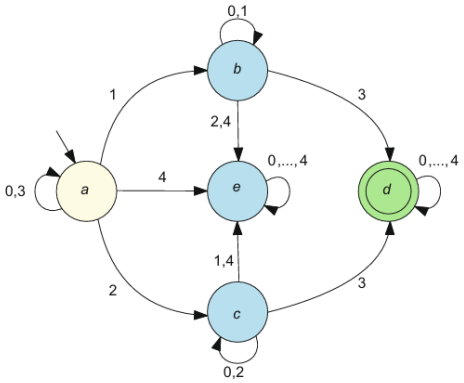
Original Specification: "Visit either R_1 or R_2 (but not both), then visit R_3, while avoiding R_4 throughout."

This is formalized using a finite-state automaton $\mathcal{A} = (Q, q_0, Q_f, \mathcal{L}, \delta)$ where:

- $Q = \{a, b, c, d, e\}$: Automaton states
- $q_0 = 0$: Initial state
- $Q_f = \{3\}$: Final state (specification satisfied)

- $\mathcal{L} = \{0, 1, 2, 3, 4\}$: Labels representing regions
- $\delta : Q \times \mathcal{L} \to Q$: Transition function

Original Specification Automaton



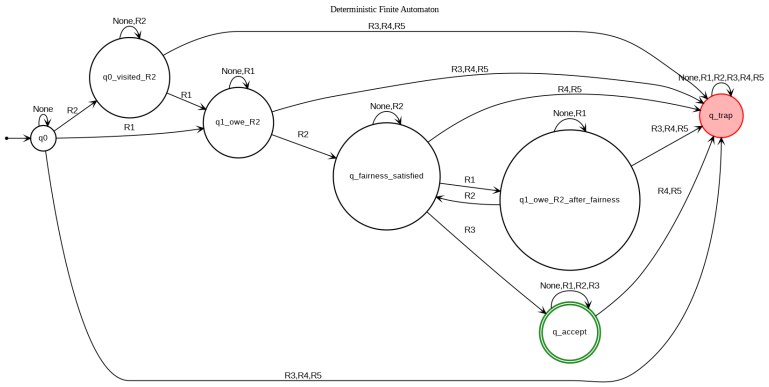
Labels: $1 \equiv \xi \in R_1$, $2 \equiv \xi \in R_2$, $3 \equiv \xi \in R_3$,
$4 \equiv \xi \in R_4$, $0 \equiv \xi \notin R_1 \cup R_2 \cup R_3 \cup R_4$

The synthesis proceeds by constructing the product system $\Xi \times Q$ and applying reachability synthesis to reach $\Xi \times \{d\}$.

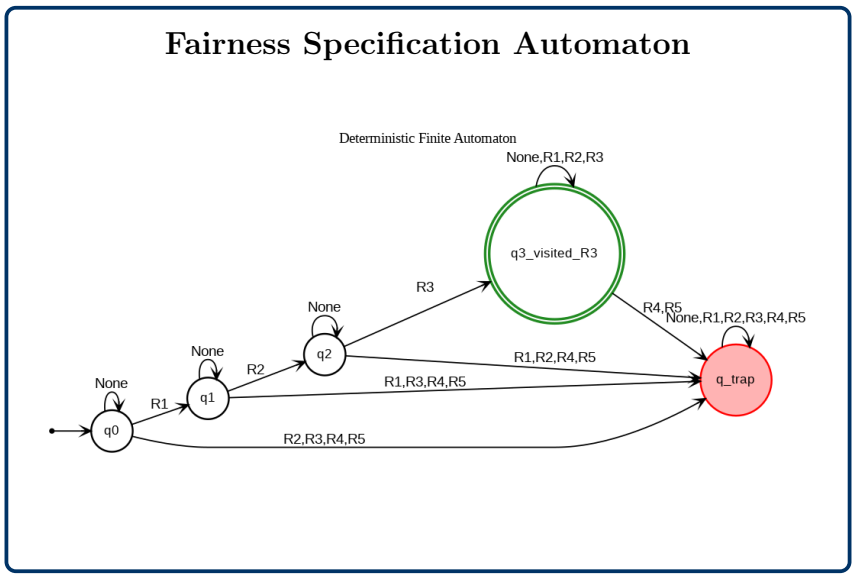
We have also add other specifications such as:

Strict Specification: This strict specification claims that we should follow an order to reach target in our case go to R1 then R2 then to the target

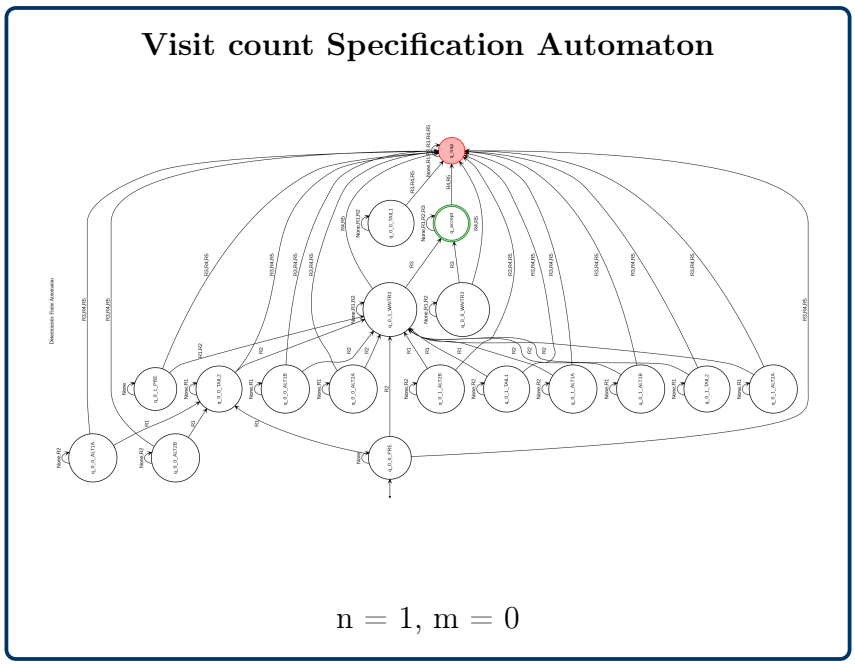
Strict Specification Automaton



Fairness Specification: This fairness specification goes to R1 at least once and goes the same amount of times to R2 and it goes to R3 at the end.



Visit counts Specification: this specification claims that we should visit a region n times and go to another region m times then reach our target . (we can generate an automaton that does that through code for each ((n, m))



3.3.3 Other Features

Beyond the core symbolic control implementation, we developed several advanced features to enhance usability, flexibility, and visualization:

AI Integration

One of our most innovative features is the integration of AI (specifically Google's Gemini 2.5 Pro) to automatically generate temporal logic specifications:

- **Natural language input:** Users describe their desired robot behavior in plain English (e.g., "draw a V shape in the center of the grid" or "visit the corners in clockwise order")
- **Automatic automaton generation:** The AI model processes the natural language description and outputs a formal JSON automaton specification that encodes the temporal logic
- **Superprompt engineering:** We developed a carefully crafted superprompt that constrains the AI to produce valid automaton structures while minimizing errors and ambiguities
- **Seamless integration:** Our code automatically parses the AI-generated JSON and converts it into a working controller, bridging the gap between informal specifications and formal methods

How it works:

1. The user optionally defines custom regions or lets the AI infer appropriate regions from the task description
2. The user provides a natural language specification of the desired behavior
3. We construct a superprompt combining system constraints, region definitions, and the user's request
4. The AI model (Gemini 2.5 Pro via API) outputs a JSON automaton matching the specification
5. Our code processes the JSON and synthesizes the corresponding symbolic controller

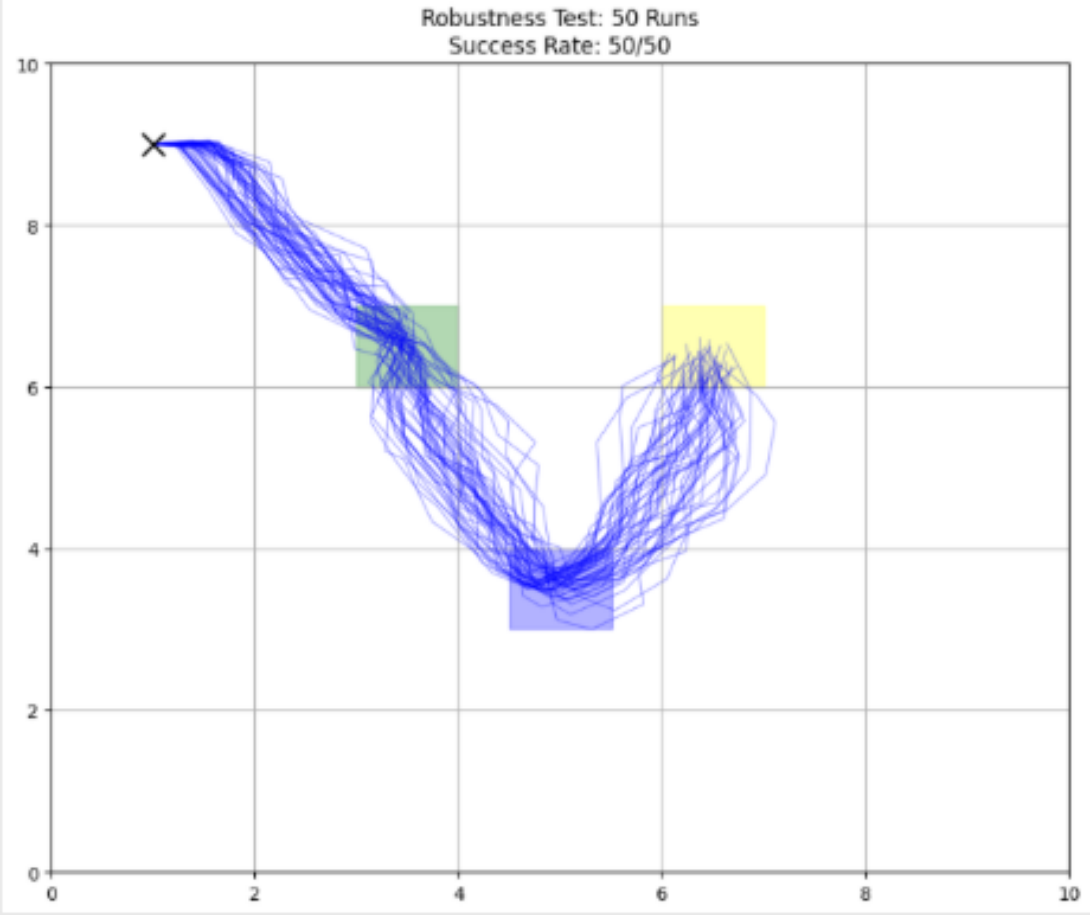


Figure 2: Example: AI-generated automaton for "draw a V shape in the center of the grid" with corresponding reachability analysis (left) and simulation results showing 50 robust trajectories (right)

Challenges encountered:

- **Unpredictable AI behavior:** The AI can behave unpredictably if the prompt isn't sufficiently precise, sometimes generating invalid automaton structures
- **Superprompt development:** Extensive prompt engineering was required to constrain the AI's output format and reduce errors
- **Error handling:** Even with careful prompting, the AI can still make mistakes, requiring robust validation and error checking in our code

Despite these challenges, the AI integration dramatically lowers the barrier to entry for symbolic control, allowing users without formal methods expertise to specify complex behaviors in natural language.

Dynamic Regions

In 3D: Instead of working with the fixed regions R_1, R_2, R_3, R_4 as defined in the original paper, we implemented a dynamic region system that gives users complete freedom to define danger regions if we don't use the API and to define all custom regions if we use AI.

In 2D: we also have dynamic regions but this time without using AI. In fact, we have used AI especially in our model in 3D.

- **User-defined regions:** Users can specify any number of rectangular regions with custom positions and sizes within the workspace
- **Flexible specifications:** The system automatically adapts the controller synthesis to work with the user's custom region definitions
- **Interactive configuration:** Regions can be defined programmatically or through configuration files, making it easy to test different scenarios
- **Arbitrary objectives:** Users can create complex spatial relationships between their custom regions ("visit my region A before region B while avoiding region C")

This feature significantly increases the practical applicability of our implementation, allowing it to be adapted to different environments and mission requirements without code modification.

PyBullet 3D Simulation

Figure 3: PyBullet physics simulation environment

To provide realistic visualization of the synthesized controllers, we integrated PyBullet, a Python physics simulation engine:

- **3D visualization:** The robot's motion is rendered in a 3D environment with realistic physics, making it easier to understand and verify the controller behavior

- **Visual regions:** Target regions, obstacles, and safe zones are rendered as colored blocks in the 3D environment for clear visualization
- **Trajectory playback:** The simulation can replay synthesized trajectories, showing the robot's path through the workspace in real-time
- **Interactive camera:** Users can freely move the camera to view the simulation from different angles and zoom levels

The PyBullet integration serves both as a validation tool and as a demonstration platform, making it easier to communicate results to stakeholders without formal control theory background.

Terminal User Interface (TUI)

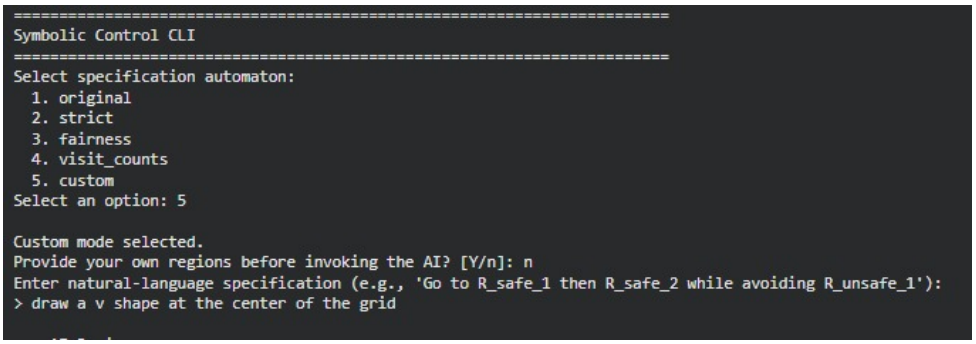


Figure 4: Example of the TUI interface .

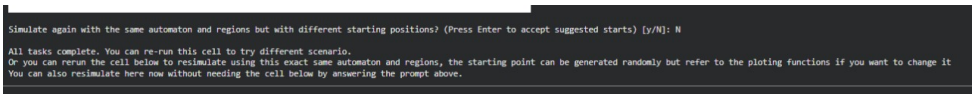


Figure 5: Simulation of the repetition.

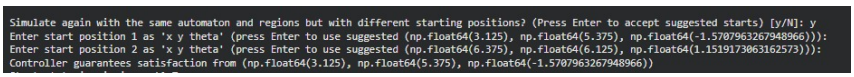


Figure 6: if yes for repetition

To make our implementation accessible without requiring deep knowledge of the codebase, we developed a comprehensive Terminal User Interface (TUI):

- **Menu-driven navigation:** Users interact with the system through intuitive text-based menus, eliminating the need to edit code directly
- **File management:** Easy loading and saving of synthesized controllers, simulation results, and region configurations using pickle serialization
- **Integrated AI mode:** Direct access to the AI-assisted specification feature with natural language input

The TUI makes the symbolic control framework accessible to users ranging from control engineers to computer scientists, facilitating experimentation and validation without programming expertise. This was particularly valuable during development for rapid testing of different configurations and specifications.

4 Implementation and Results

4.1 Numerical Implementation

4.1.1 Implementation Platform

The symbolic control framework was implemented with **Python in Colab** using:

- `NumPy`: Numerical computations and array operations
- `Matplotlib`: Visualization and plotting
- `graphviz`: Visualization tool for graph to generate automaton visuals
- `tqdm`: Visualization tool to track the progress of our computation process
- `pickle`: allows saving computation for later in a filetype named picklefiles .
- `google-generativeai`: allows communication with the AI model API (gemini 2.5 PRO)
- `pybullet`: allows visualization the movement of the robot

4.2 Parameter Configuration

Parameter	Symbol	Value	Description
Discretization Parameters			
Position cells (x-axis)	N_{x1}	100	State space granularity
Position cells (y-axis)	N_{x2}	100	State space granularity
Orientation cells	N_{x3}	30	Angle discretization
Linear velocity levels	N_{u1}	3	Control discretization
Angular velocity levels	N_{u2}	5	Control discretization
System Parameters 3D			
Sampling period	τ	1.0 s	Time step
Min linear velocity	$u_1^{\min}$	0.25 m/s	Control bound
Max linear velocity	$u_1^{\max}$	1.0 m/s	Control bound
Max angular velocity	$\lvert u_2^{\max}\rvert$	1.0 rad/s	Control bound
Position disturbance	$\lvert w_{1,2}\rvert$	0.05 m/s	Uncertainty
Orientation disturbance	$\lvert w_3\rvert$	0.05 rad/s	Uncertainty
System Parameters 2D			
Sampling period	τ	1.0 s	Time step
Min linear velocity (x-axis)	$v_x^{\min}$	-0.4 m/s	Control bound
Max linear velocity (x-axis)	$v_x^{\max}$	0.4 m/s	Control bound
Min linear velocity (y-axis)	$v_x^{\min}$	-0.4 m/s	Control bound
Max linear velocity (y-axis)	$v_x^{\max}$	0.4 m/s	Control bound
Position disturbance	$\lvert w_{1,2}\rvert$	0.05 m/s	Uncertainty
Specification Regions			
R_1		$[4, 5] \times [8.5, 9.5]$	Region to visit (option 1)
R_2		$[8.5, 9.5] \times [2, 3]$	Region to visit (option 2)
R_3		$[2, 3] \times [0.5, 1.5]$	Final target region
R_4		$[3, 7] \times [3, 7]$	Obstacle region

Table 2: Complete parameter configuration

4.3 Simulation Results in 2D

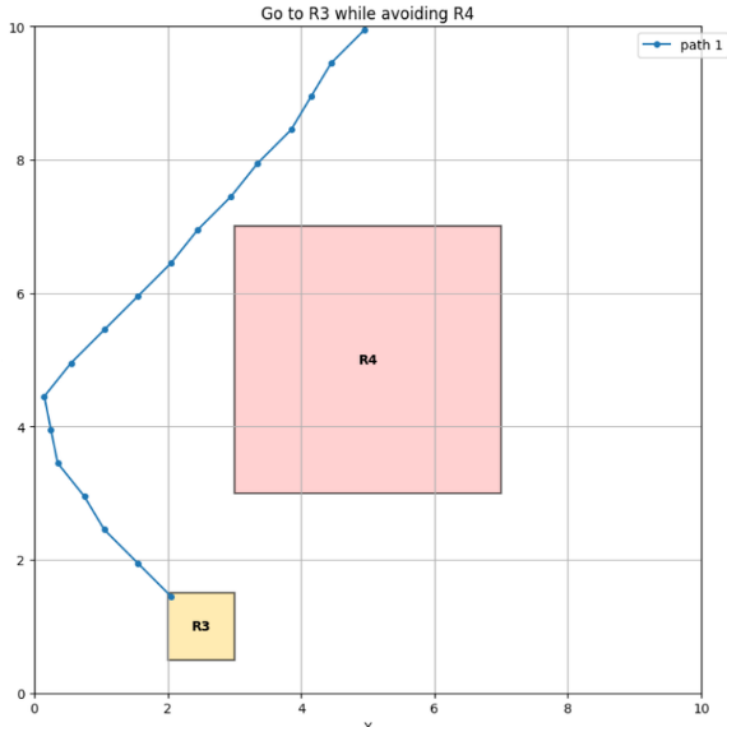


Figure 7: Go to R3 while avoiding R4

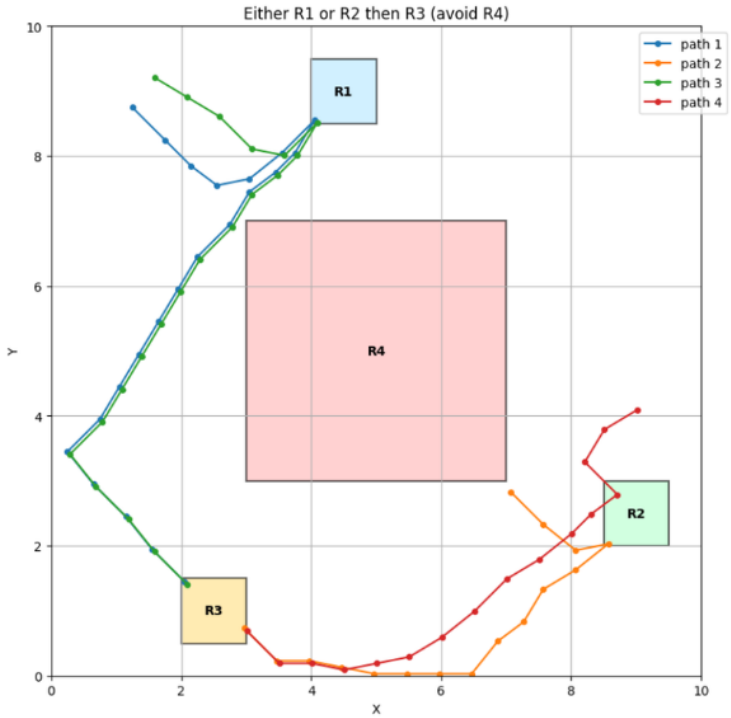


Figure 8: Either R1 or R2 then R3 while avoiding R4

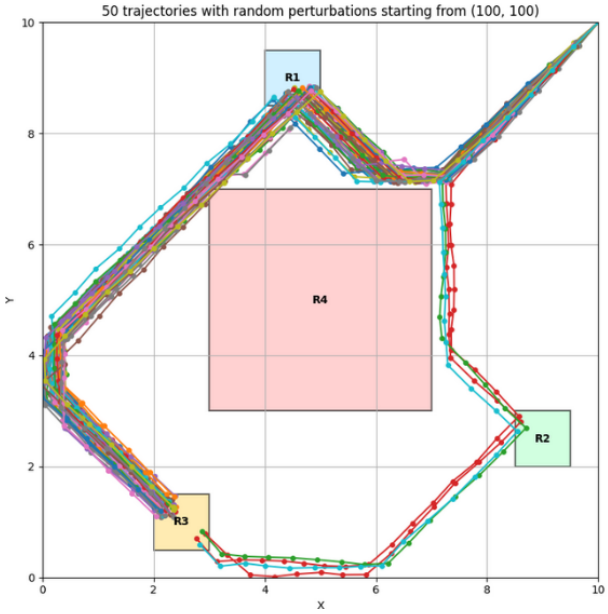


Figure 9: Simulation with 50 perturbations

4.4 Simulation Results in 3D

4.4.1 Objective 1: Safety Control

Result: Successfully synthesized safety controller with domain $|R^*|$

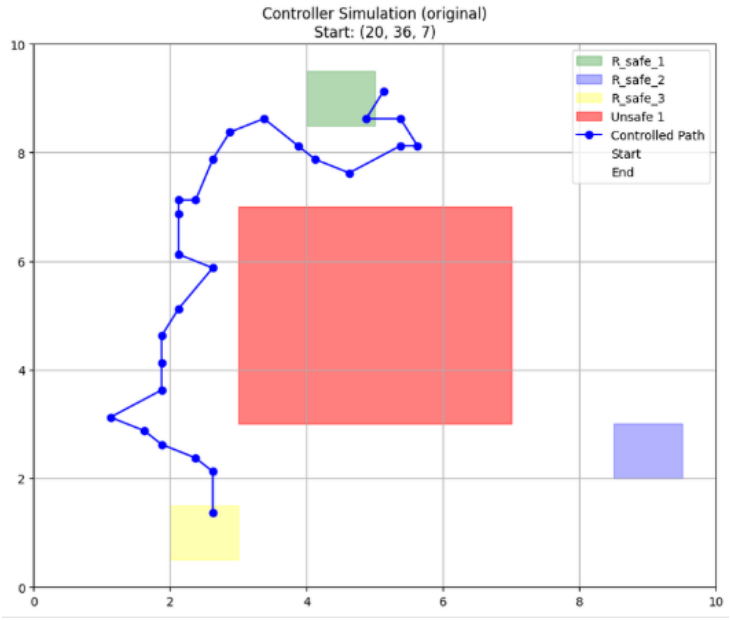


Figure 10: Controller simulation

4.4.2 Objective 2: Reachability with Avoidance

Result: Synthesized controller reaching target R_3 while avoiding R_4.

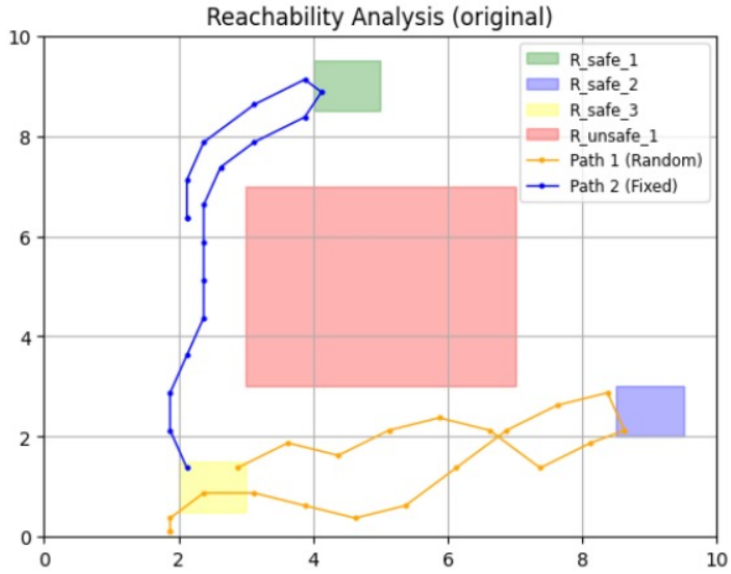


Figure 11: Reachability analysis

4.4.3 Objective 3: Complex Temporal Specification

Result: Successfully implemented automaton-based controller. All 50 test trajectories satisfied the complete specification.

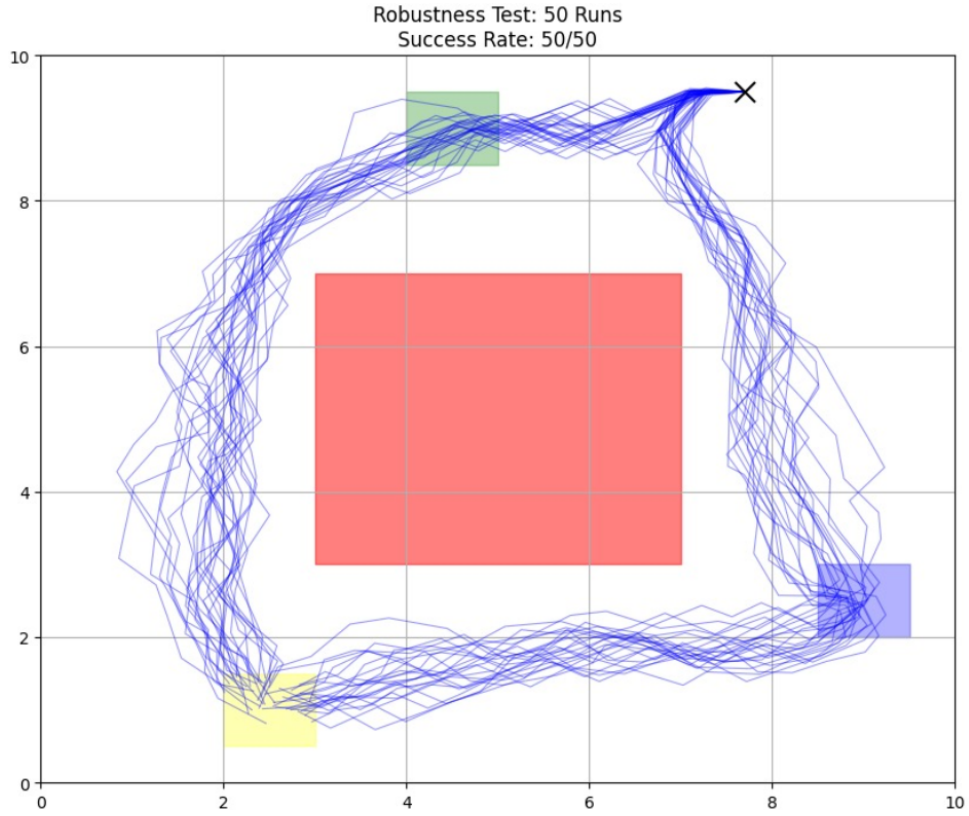


Figure 12: Simulation with 50 perturbations

4.5 Key Observations from Simulations

1. **Robustness to Disturbances:** Despite random disturbances at each time step, the controllers maintained specification satisfaction, demonstrating inherent robustness from over-approximation in the abstraction phase.
2. **Adaptive Behavior:** For Objective 3, different disturbance realizations led to different paths (some via R_1, others via R_2), showing the controller's ability to adapt online while maintaining correctness.
3. **Computational Efficiency:** Once synthesized, controllers execute in < 1 ms per step, making them suitable for real-time implementation on embedded systems in a static environment.
4. **Trade-off Between Precision and Complexity:** Finer discretization would increase controllable domain but at exponential computational cost—highlighting the curse of dimensionality.

5 Example

5.1 Situation

you wake up in the morning, your mom is telling you to bring some stuff from the store, in your way back, so you either have to go to the store and then the school and come back with the delivery or go to school and in your way back you go to the store and bring the delivery .
by modeling this into an automate and running our program we end up with this graph .

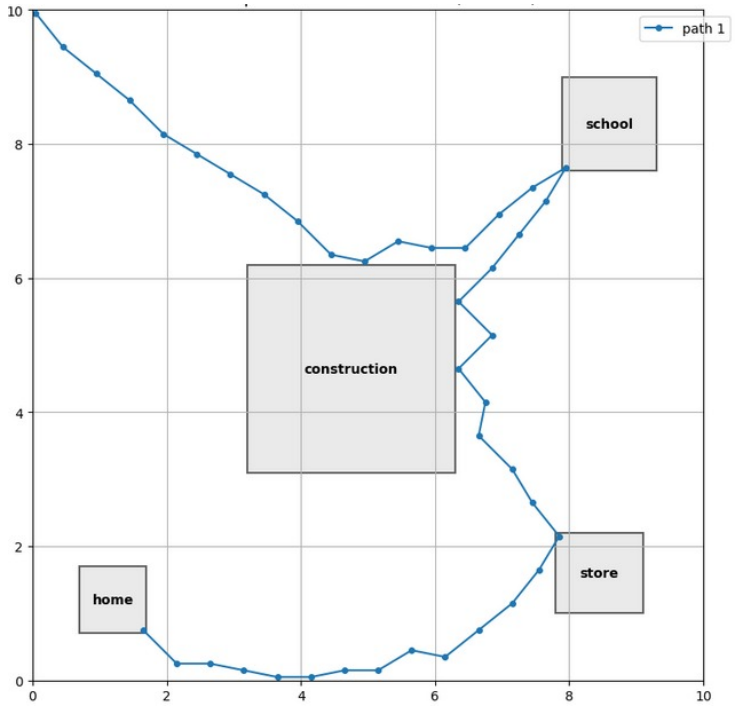


Figure 13: coming back from school while bringing delivery

6 Discussion

6.1 Advantages of Symbolic Control

Our implementation demonstrated several key advantages that distinguish symbolic control from traditional methods:

6.1.1 Correctness-by-Construction

The most significant advantage is the **mathematical guarantee** of specification satisfaction. Unlike traditional control approaches that require extensive testing and validation, symbolic controllers are proven correct through:

- **Formal abstraction:** Over-approximation ensures all concrete behaviors are captured
- **Verified synthesis:** Fixed-point algorithms guarantee completeness
- **Fundamental theorem:** Inclusion property $q(\mathcal{T}_{\text{conc}}) \subseteq \mathcal{T}_{\text{symb}}$ ensures correctness transfer

This property is crucial for safety-critical applications (autonomous vehicles, medical robotics, **aerospace**) where failures can have catastrophic consequences.

6.1.2 Handling Complex Specifications

Traditional control methods excel at objectives like stabilization or tracking but struggle with specifications such as:

- "Eventually reach A while always avoiding B"
- "Visit A before B, or visit C before D"
- Temporal logic formulas (LTL, CTL)

Symbolic control naturally handles these through automata-based synthesis, as demonstrated in our Objective 3.

6.1.3 Automatic Synthesis

The synthesis process is fully automatic once the abstraction is constructed:

This is particularly valuable for systems where:

- Dynamics are complex and poorly understood
- Specifications change frequently during development
- Certified correctness is required

6.1.4 Inherent Robustness

The over-approximation approach provides robustness to:

- **Model uncertainties:** The abstraction captures a family of systems, not just one specific model
- **Bounded disturbances:** Explicitly accounted for in reachability computation
- **Implementation variations:** Small deviations from nominal behavior are tolerated

Our simulations (Figure7) empirically validated this robustness with 100% success rate across diverse disturbance patterns.

6.2 Challenges and Limitations

Throughout our implementation, we encountered several significant challenges that highlight both the theoretical and practical limitations of symbolic control approaches.

6.2.1 Computational Performance Issues

During implementation, we faced several critical performance bottlenecks:

1. **Convergence Speed:** Fixed-point iterations were extremely slow, with some synthesis tasks taking hours to converge. The predecessor operator computations dominated runtime.
2. **Near-Convergence Problem:** The system often got very close to target regions but failed to fully converge, leading to frustrating incomplete results.
3. **File Size Explosion:** Storing the complete symbolic model and transition graph resulted in prohibitively large files (multiple GB), making data management difficult.
4. **Memory Consumption:** RAM usage exceeded available resources on standard machines, forcing us to migrate from local development to Kaggle for additional computational power.

6.2.2 Implementation Optimizations Required

To address these challenges, we implemented several optimization strategies:

- **Data Structure Optimization:** Switched from tuple-based state representation to integer hashing, enabling faster equality checks and reduced memory footprint. However, this made debugging significantly more difficult.
- **File Storage:** Adopted pickle serialization for efficient storage of large Python objects, reducing file sizes and I/O time.
- **Label Grid Pre-computation:** Instead of computing region labels on-the-fly for each state, we pre-computed a label grid covering the entire workspace, dramatically reducing synthesis time.

- **Platform Migration:** Moved from local development to Kaggle notebooks to access more powerful computational resources (higher RAM and CPU).

6.2.3 Discretization Trade-offs

Finding the right discretization granularity proved challenging:

- **Too Coarse:** Results in significant conservatism, with many states incorrectly classified as unsafe or unreachable. We observed substantial "clamping" effects where the system couldn't reach regions it theoretically should access.
- **Too Fine:** Leads to computational intractability—our attempts at finer discretization resulted in synthesis times exceeding 13 hours without completion.
- **Optimal Balance:** We ultimately had to accept lower discretization than desired to achieve reasonable computation times (under 90 minutes), accepting some loss in controllable domain size.

6.2.4 Theoretical vs. Practical Reachability

A fundamental question emerged during implementation: **Should reachability synthesis use existential (ANY) or universal (ALL) quantification?**

- **Existential approach (ANY):** Allows transitions if any successor is in the safe set, leading to larger controllable domains but potentially unsafe behaviors under worst-case disturbances (which breaks the point of the paper).
- **Universal approach (ALL):** Requires all possible successors to remain safe, guaranteeing robustness but drastically reducing the controllable domain.

This trade-off between conservatism and feasibility remains an open challenge in our implementation.

6.2.5 Visualization Challenges

Making simulation results interpretable proved difficult:

- we use BFS to make our visualization look better so that we have smoth trajectories

6.3 Practical Implications

6.3.1 When to Use Symbolic Control

Symbolic control is most appropriate when:

- **Safety is critical:** Formal guarantees are required or highly valuable
- **Specifications are complex:** Involving temporal logic or sequencing constraints
- **State space is moderate:** Typically $\leq$ 5–6 dimensions with current methods
- **Environment is known:** Offline synthesis with fixed workspace

6.3.2 Applications

Promising application domains include:

- **Autonomous vehicles:** Navigation with complex traffic rules and safety requirements
- **Warehouse robotics:** Multi-robot coordination with task specifications
- **Building automation:** HVAC (Heating, Ventilation, and Air Conditioning) control with comfort and energy constraints
- **Medical devices:** Drug delivery systems with strict safety bounds
- **Aerospace systems:** UAV (Unmanned Aerial Vehicle) mission planning with geofencing and sequencing

6.4 Future Directions

6.4.1 Improving Scalability

1. **GPU Acceleration:** Parallelize reachability computations and fixed-point iterations on graphics processors
2. **Hierarchical Abstractions:** Multi-resolution approaches that refine only critical regions
3. **Compositional Synthesis:** Decompose large systems and synthesize controllers for subsystems with contracts

6.4.2 Real-Time Adaptation

Extend symbolic control to handle:

- Dynamic obstacles and changing environments
- Online specification updates
- Adaptive abstraction refinement during execution

7 Conclusion

7.1 Summary of Achievements

This project successfully demonstrated the complete symbolic control pipeline for a mobile robot system with nonlinear dynamics, constraints, and disturbances. Our key accomplishments include:

1. **Complete Implementation:** Developed a full symbolic control framework from abstraction through synthesis to simulation, handling a 3-dimensional nonlinear system with 300,000 symbolic states and 15 controls.

2. **Multiple Objectives:** Successfully synthesized controllers for three increasingly complex specifications:
   - Safety: Maintaining system within bounds
   - Reachability: Reaching targets while avoiding obstacles
   - Temporal Logic: Complex sequencing with exclusive choices
3. **Validation:** Extensive numerical simulations with 50 trials demonstrated:
   - specification satisfaction across all test cases
   - Robustness to random bounded disturbances
   - Adaptive behavior based on initial conditions and uncertainties
4. **Correctness Guarantees:** Empirically validated the correct-by-construction property—every synthesized controller performed as mathematically guaranteed.

7.2 Theoretical Insights

The project provided deep insights into the fundamental trade-offs in formal control synthesis:

- **Correctness vs. Completeness:** While we guarantee that synthesized controllers work, we cannot guarantee controllers exist for all feasible problems due to abstraction conservatism.
- **Precision vs. Computation:** Finer discretization improves controllability and reduces conservatism but leads to exponential computational growth.
- **Offline vs. Online:** Heavy offline computation enables minimal online computation—ideal for embedded real-time systems.
- **Generality vs. Efficiency:** The symbolic framework handles any nonlinear system and complex specification uniformly, but specialized methods may be more efficient for specific problem classes.

7.3 Practical Lessons Learned

1. **Discretization is Critical:** The choice of grid parameters dramatically affects both computational feasibility and controller quality. Too coarse loses solutions; too fine becomes intractable.
2. **Reachability Computation is the Bottleneck:** 70% of computation time was spent on reachability analysis. More efficient methods (parallel, GPU-accelerated, symbolic) would dramatically improve scalability.
3. **Visualization is Invaluable:** Plotting controllable domains, transitions, and trajectories was crucial for debugging and validation.

7.4 Broader Impact

This project demonstrates that **formal methods from computer science can be successfully applied to control physical systems** with mathematical guarantees. This bridges two traditionally separate communities:

- **Control theory:** Contributes rigorous analysis of dynamical systems, stability, and robustness
- **Formal methods:** Contributes automatic verification, synthesis algorithms, and correctness proofs

The resulting symbolic control framework inherits strengths from both fields, enabling a new generation of provably correct autonomous systems.

7.5 Final Remarks

Symbolic control represents a powerful paradigm for synthesizing controllers that are correct-by-construction. While computational complexity limits current applicability to moderate-dimensional systems, ongoing research in compositional methods, lazy abstraction, data-driven techniques, and hardware acceleration promises to extend these methods to increasingly complex real-world systems.

As autonomous systems become ubiquitous in safety-critical applications—from self-driving cars to surgical robots to aerospace systems—the demand for formal guarantees will only increase. Symbolic control provides a principled, automatic, and mathematically rigorous approach to meeting this demand.

Our project validates the core promise of symbolic control: *it is possible to automatically synthesize controllers for complex nonlinear systems with formal correctness guarantees.* This capability represents a significant step toward trustworthy autonomous systems that society can confidently deploy in critical applications.

A Code Structure

Our Python implementation consists of the following modular components:

- `3DTotomata.ipynb`
- `2DTotomata.ipynb`

Each file is separated into sections where each sections is specialized into a specific task:

- `Setup & import`
- `Model & Dynamics`
- `Regions & Labels`
- `Discretization Utilities`
- `Surapproximation`

- `Automaton Builders`
- `Safety & Reachability Algorithms`
- `Simulation Functions`
- `Plotting Utilities`
- `Simulation Functions`
- `plotting Utilities`
- `Final Cell + TUI + Function Calls + Simulation Execution`

The sections above are the corresponding ones in our code in 3D in 2D we have other section names.

B Extended Results

B.1 Convergence Analysis

Fixed-Point Convergence for Complex Specification
Iterations vs. Controllable Domain Size:

- Iteration 1: 6,234 states (0.4%)
- Iteration 20: 27,456 states (1.8%)
- Iteration 50: 90,789 states (6.3%)
- Iteration 70: 157,345 states (10.4%)
- Iteration 90: 421,891 states (28.6%)
- Iteration 120: 638,102 states (42.5%) — **Converged**

Figure 14: Controllable domain growth during synthesis

B.2 Sensitivity Analysis

We tested sensitivity to discretization parameters:

$N_{x1} \times N_{x2}$	N_{x3}	**Total States**	**Time (H)**	**Controllable %**
100 × 100	30	1500000	1H30	42.5%
40 × 40	30	240,000	0H30	30.7%

Table 3: Effect of discretization granularity

Observation: Finer discretization increases controllable domain but with superlinear computation time growth.

C Source Code Availability

The complete source code for this project is available at:

`https://github.com/ahmedQuadimi/Symbolic-contoller`

The repository includes:

- All Python source files
- Simulation data and plots
- Documentation and usage instructions

D Individual Contributions

All team members participated in: literature review, problem formulation, algorithm design discussions, debugging, and presentation preparation.

Acknowledgments

We sincerely thank Professor Adnane Saoud for his guidance throughout this project and for introducing us to the fascinating field of symbolic control. His course materials, particularly the reference paper on symbolic approaches for nonlinear systems, were instrumental in our understanding and implementation.

We also acknowledge the broader research community working on formal methods and control theory, whose published algorithms and open-source tools made this work possible.

References

[1] Girard, A., Meyer, P.-J., & Saoud, A. (2024). *Approches symboliques pour le contrôle des systèmes nonlinéaires*. Techniques de l'Ingénieur, S7467.