UM6P College of Computing

Report

Symbolic Control for a Nonlinear Mobile Robot

Students:
Khaoula Jellal
Yassine Blali
Othmane Azoubi

Professor:
Professor Adnane Saoud

Contents

1 Abstract

This report presents the implementation of symbolic control for a nonlinear mobile robot, focusing on discretization, transition computation, and controller synthesis. Symbolic control provides a systematic framework to ensure system behaviors satisfy complex specifications, such as safety and reachability, under bounded disturbances. The continuous state space of the robot, defined in terms of position (x, y) and orientation θ, was discretized into a finite grid, enabling efficient symbolic abstraction.

We computed transitions between states by leveraging the system's dynamics, incorporating control inputs and disturbances, and using Jacobian-based over-approximations to determine reachable sets. A symbolic controller was synthesized to ensure the robot stays within a defined region, avoids forbidden zones, and satisfies sequential visitation objectives. The implementation was validated through simulations demonstrating the robot's ability to adapt to uncertainties and disturbances while meeting the specified objectives.

Our results showcase the practical utility of symbolic control in achieving robust and verifiable behaviors in complex systems. This work highlights the potential for applying these techniques to broader control problems, providing a foundation for further research and real-world applications.

2 Introduction

Symbolic control offers a compelling approach to managing complex systems by translating continuous dynamics into a discrete, computationally tractable framework. For nonlinear systems, such as a mobile robot operating in uncertain environments, the ability to synthesize controllers that guarantee desired behaviors is crucial. Symbolic control achieves this by discretizing the system's state space into a finite abstraction and constructing transitions that account for the effects of control inputs and bounded disturbances. The resulting symbolic model provides a foundation for verifying and synthesizing controllers that meet specified objectives, such as safety and reachability.

In the context of mobile robots, the challenge lies in navigating a continuous state space—defined by the robot's position (x, y) and orientation θ—while satisfying multiple constraints. The robot must stay within prescribed boundaries, avoid restricted areas, and sequentially visit target regions, all while accounting for uncertainties. Addressing these requirements in the continuous domain is computationally infeasible, but symbolic control transforms the problem into a finite-state model, enabling rigorous verification and synthesis techniques.

Building on the framework presented in *[paper title]*, this project focuses on implementing a symbolic control approach for a nonlinear mobile robot. Our primary goals are:

1. Discretizing the state space to create a finite grid of symbolic states.
2. Computing transitions that capture the robot's behavior under dynamics, control, and disturbances.
3. Synthesizing a symbolic controller to ensure safety and achieve specific reachability objectives.

4. Validating the approach through simulations that demonstrate adherence to the specifications.

This report details our implementation of the symbolic control framework, including the challenges encountered and insights gained. By systematically bridging the theoretical concepts with practical tools, we aim to demonstrate the utility of symbolic control in managing the complexities of nonlinear systems. Our findings highlight the potential of this approach for extending the scope of robust and verifiable control in uncertain environments.

3 Methodology

3.1 State Space Discretization

The process of state space discretization serves as the cornerstone of symbolic control, transforming an infinite and continuous state space into a finite abstraction that can be computationally analyzed. This abstraction bridges the gap between theoretical control objectives and practical implementation by enabling the system to reason about its behavior in discrete symbolic terms.

3.1.1 The Need for Discretization

In a nonlinear system like the mobile robot, the state space is defined continuously in terms of position (x, y) and orientation θ. However, this continuity introduces infinite possibilities, rendering direct computation infeasible. By discretizing the state space, we approximate the continuous dynamics with a manageable, finite set of symbolic states, making it possible to compute transitions and synthesize a controller that guarantees safety and reachability properties.

3.1.2 Mathematical Formulation

The state space X is bounded and expressed as:

$$X = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \times [\theta_{\min}, \theta_{\max}],$$

where x and y represent the robot's position in 2D space, and θ denotes its orientation. For our implementation:

$$X = [0, 10] \times [0, 10] \times [-\pi, \pi].$$

The continuous domain is partitioned into a finite grid, dividing each dimension into $n_x \times n_y \times n_\theta$ cells. For instance:

$$n_x = 50, \quad n_y = 50, \quad n_\theta = 16,$$

resulting in $50 \times 50 \times 16 = 40,000$ symbolic states. Each cell represents a small subset of the continuous state space, allowing the system to abstract a finite representation.

The step size for each dimension is computed as:

$$\Delta x = \frac{x_{\max} - x_{\min}}{n_x}, \quad \Delta y = \frac{y_{\max} - y_{\min}}{n_y}, \quad \Delta\theta = \frac{\theta_{\max} - \theta_{\min}}{n_\theta}.$$

These steps ensure that each cell is uniform, providing a consistent approximation of the continuous space.

3.1.3 Symbolic States

Each symbolic state corresponds to the center of a cell in the grid:

$$\xi_{i,j,k} = (x_i, y_j, \theta_k),$$

where:

$$x_i = x_{\min} + i \cdot \Delta x, \quad y_j = y_{\min} + j \cdot \Delta y, \quad \theta_k = \theta_{\min} + k \cdot \Delta\theta.$$

Continuous states (x, y, θ) are mapped to symbolic indices (i, j, k) using:

$$i = \left\lfloor \frac{x - x_{\min}}{\Delta x} \right\rfloor, \quad j = \left\lfloor \frac{y - y_{\min}}{\Delta y} \right\rfloor, \quad k = \left\lfloor \frac{\theta - \theta_{\min}}{\Delta\theta} \right\rfloor.$$

This mapping ensures that each continuous state is assigned to the corresponding symbolic cell in the discretized space.

3.1.4 Visualization

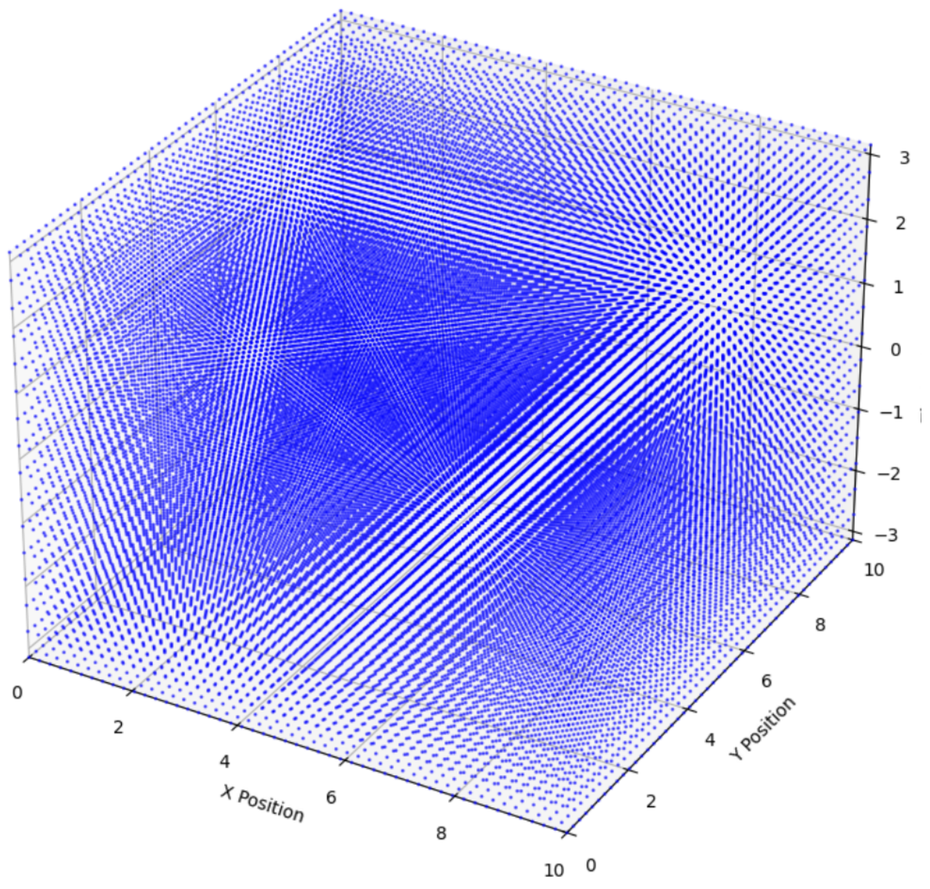

Figure 1: Visualization of the discretized state space.

3.2 Transition Computation

Transition computation is a critical step in constructing the symbolic abstraction. It determines how the robot transitions between symbolic states under the influence of control inputs and bounded disturbances. This section outlines the logic, mathematical formulation, and implementation of transition computation, ensuring a clear connection between theoretical concepts and practical implementation.

3.2.1 Defining the Problem

The dynamics of the mobile robot are modeled as:

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k, \mathbf{w}_k),$$

where:

- $\mathbf{x}_k = (x, y, \theta)$ represents the robot's state at time step k, including its 2D position (x, y) and orientation θ.
- $\mathbf{u}_k = (v, \omega)$ denotes the control inputs, with v as the linear velocity and ω as the angular velocity.
- $\mathbf{w}_k$ captures bounded disturbances, constrained by:

$$\mathbf{w}_k \in [\mathbf{w}_{\min}, \mathbf{w}_{\max}].$$

The objective is to compute a **reachable set**, which contains all possible successor states $\mathbf{x}_{k+1}$ from a given state $\mathbf{x}_k$ under a specific control input $\mathbf{u}_k$, considering all possible disturbances $\mathbf{w}_k$. This reachable set is then discretized to define transitions in the symbolic abstraction.

3.2.2 Reachable Set Computation

The reachable set $\text{Reach}(\mathbf{x}_k, \mathbf{u}_k)$ is defined as:

$$\text{Reach}(\mathbf{x}_k, \mathbf{u}_k) = \{\mathbf{x}_{k+1} \mid \mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k, \mathbf{w}_k), \forall \mathbf{w}_k \in [\mathbf{w}_{\min}, \mathbf{w}_{\max}]\}.$$

To approximate this set efficiently, we use linearized dynamics around the current state $\mathbf{x}_k$. The deviation from the nominal trajectory is bounded as:

$$\Delta\mathbf{x}_{\text{succ}} = 0.5 \cdot J_{f_x}(\mathbf{u}) \cdot \Delta\mathbf{x} + 0.5 \cdot J_{f_w}(\mathbf{u}) \cdot \Delta\mathbf{w},$$

where:

- $J_{f_x}(\mathbf{u}) = \frac{\partial f}{\partial \mathbf{x}}$: Jacobian of the dynamics with respect to the state.
- $J_{f_w}(\mathbf{u}) = \frac{\partial f}{\partial \mathbf{w}}$: Jacobian of the dynamics with respect to the disturbance.
- $\Delta\mathbf{x}$ and $\Delta\mathbf{w}$ are the discretization step sizes for the state and disturbance spaces, respectively.

The reachable set is over-approximated as:

$$\text{Reach} = [\mathbf{x}_{\text{succ}} - \Delta\mathbf{x}_{\text{succ}}, \mathbf{x}_{\text{succ}} + \Delta\mathbf{x}_{\text{succ}}],$$

where $\mathbf{x}_{\text{succ}}$ is the nominal successor state computed as:

$$\mathbf{x}_{\text{succ}} = f(\mathbf{x}_k, \mathbf{u}_k, \mathbf{w}_{\text{center}}),$$

and $\mathbf{w}_{\text{center}}$ is the midpoint of the disturbance bounds.

3.2.3 Discretization of the Reachable Set

To incorporate the reachable set into the symbolic abstraction, it is discretized into symbolic indices. Each symbolic state $\xi_{i,j,k}$ corresponds to a grid cell in the discretized state space, and transitions are represented as pairs of symbolic indices.

The discretization process includes:

1. **Mapping to Symbolic Indices**: Continuous states $\mathbf{x}$ are mapped to their corresponding symbolic indices by dividing the continuous state space into a finite grid. Each dimension is discretized based on the step size $\Delta\mathbf{x}$:

$$i = \left\lfloor \frac{x - x_{\min}}{\Delta x} \right\rfloor, \quad j = \left\lfloor \frac{y - y_{\min}}{\Delta y} \right\rfloor, \quad k = \left\lfloor \frac{\theta - \theta_{\min}}{\Delta \theta} \right\rfloor.$$

   Here, i, j, and k represent the indices of the symbolic state in the grid for the x, y, and θ dimensions, respectively.

2. **Successor Identification**: For a given symbolic state and control input, the reachable set is over-approximated using the bounds of the dynamics. The minimum and maximum points of this reachable set in each dimension are calculated. These bounds are then discretized into symbolic indices by determining which grid cells the boundary points belong to:

$$\text{MinSucc} = \left\lfloor \frac{\text{Reach}_{\min} - \mathbf{x}_{\min}}{\Delta\mathbf{x}} \right\rfloor, \quad \text{MaxSucc} = \left\lceil \frac{\text{Reach}_{\max} - \mathbf{x}_{\min}}{\Delta\mathbf{x}} \right\rceil.$$

   These indices represent the range of symbolic states that the reachable set overlaps with in the discretized state space.

3. **Handling Angular Wrapping**: Since the orientation θ is cyclic, it must be wrapped to stay within the range $[-\pi, \pi]$. This ensures consistency when computing transitions involving angular dynamics. The wrapping is handled as follows:

$$\theta = \begin{cases} \theta + 2\pi, & \text{if } \theta < -\pi, \\ \theta - 2\pi, & \text{if } \theta > \pi. \end{cases}$$

   This adjustment guarantees that angular transitions respect the natural periodicity of θ and remain within the bounded domain.

3.2.4 Implementation

The following snippet outlines the computation of transitions:

```python
def computeSymbolicModel(f, Jf_x, Jf_w):
    g = np.zeros((n_xi, n_sigma, 2), dtype=int)  # Transition
        relation
    for xi in range(1, n_xi + 1):
        xiCenter = state2coord(xi, p_x)  # Map state to coordinates
        xCenter = bound_x[:, 0] + (xiCenter - 0.5) * d_x

        for sigma in range(1, n_sigma + 1):
            xSucc = f(xCenter, ControlDisc[:, sigma - 1], w_center)
            dxSucc = 0.5 * Jf_x(ControlDisc[:, sigma - 1]) @ d_x +
                0.5 * Jf_w(ControlDisc[:, sigma - 1]) @ d_w

            reach = np.vstack([xSucc - dxSucc, xSucc + dxSucc])
            reach = np.clip(reach, bound_x[:, 0], bound_x[:, 1])  #
                Apply bounds

            minSucc = coord2state(np.floor((reach[:, 0] - bound_x[:,
                0]) / d_x).astype(int) + 1, p_x)
            maxSucc = coord2state(np.ceil((reach[:, 1] - bound_x[:,
                0]) / d_x).astype(int), p_x)
            g[xi - 1, sigma - 1] = [minSucc, maxSucc]
    return g
```

Listing 1: Implementation of Transition Computation

3.2.5 Validation of Transitions

The computed transitions were validated to ensure:

- **Correctness**: Transitions respect state space bounds.
- **Completeness**: Reachable sets fully encompass the robot's dynamics and disturbances.
- **Consistency**: Successor states align with expected dynamics.

State Index	Input Index	Min Successor	Max Successor
1	1	0	0
1	2	0	0
1	3	0	0
1	4	0	0
1	5	0	0
1	6	0	0
...	...	...	...

State Index	Input Index	Min Successor	Max Successor
4310	1	244106	264409
4310	2	244102	264405
4310	3	0	0
4310	4	274106	284409
4310	5	274102	284405
4310	6	0	0
...	...	...	...

Figure 2: Excerpts of the symbolic model transition table.

The validated transition model was then used for controller synthesis.

3.3 Fixed-Point Algorithm for Controller Synthesis

The fixed-point algorithm forms the backbone of the symbolic controller synthesis process. By iteratively updating a value function and refining a control policy, the algorithm ensures that the robot adheres to the predefined objectives: remaining within the bounded region, avoiding forbidden zones, and fulfilling sequential visitation constraints.

3.3.1 Key Requirements and Approach

The synthesis process hinges on three main components:

- **Symbolic Abstraction**: A discretized representation of the robot's state space and dynamics, capturing all possible transitions under a range of control inputs and disturbances.
- **Labeling Function**: A mapping function that categorizes states into specific objectives or constraints (e.g., "reaching the goal," "avoiding forbidden zones").
- **Systematic Iterative Algorithm**: A procedure that refines the value function $V(\psi, \xi)$—indicating the cost-to-go for each state—and the control policy $h(\psi, \xi)$ until convergence.

3.3.2 Algorithm Workflow

The fixed-point algorithm progresses through four main stages:

1. **Initialization**:
   - The value function $V(\psi, \xi)$ is initialized to ∞ for all states except the goal states F_s, which are set to $V(F_s, \xi) = 0$.
   - The control policy $h(\psi, \xi)$ is initially unassigned.
2. **Iteration**:
   - For each state ξ and mode ψ, reachable states under each control input σ are evaluated. The value function is updated as:

$$V_{k+1}(\psi, \xi) = \begin{cases} 0 & \text{if } \xi \in F_s, \\ \min_{\sigma \in \Sigma} \max_{x' \in \text{Reach}(\xi, \sigma)} V_k(\psi', x') & \text{otherwise.} \end{cases}$$

   - Here:
     - Σ: The set of control inputs.
     - $\text{Reach}(\xi, \sigma)$: The set of reachable states under input σ.
     - ψ': The successor mode derived from the labeling function.
3. **Convergence Check**:
   - Iteration halts when the value function stabilizes (no further updates) or a timeout condition is reached.
4. **Control Policy Extraction**:
   - For each state, the optimal control input σ that minimizes the value function is assigned.

3.3.3 Implementation Details

The algorithm was implemented in Python, using symbolic abstractions to efficiently compute transitions and refine the value function. The implementation ensures scalability and computational efficiency, even with large state spaces. A simplified snippet is provided for context:

```python
class SymbolicController:

    def __init__(self, symbolicAbstraction):
        self.symbolicTransitions = symbolicAbstraction
        self.V = np.inf * np.ones((n_s, n_xi))  # Initialize value function
        self.h2 = np.zeros((n_s, n_xi), dtype=int)  # Initialize control policy
        for f in F_s:
            self.V[f - 1, :] = 0  # Goal states have zero cost

    def SynthesisController(self, max_iter=100):
        for iter in range(max_iter):
            Vp = self.V.copy()
            for xi in range(n_xi):
                for psi in range(1, n_s + 1):
                    if self.V[psi - 1, xi] == np.inf:
                        Vmax = np.zeros(n_sigma)
                        for sigma in range(n_sigma):
                            xi_succ = self.symbolicTransitions[xi, sigma, :]
                            if np.all(xi_succ):
                                vSucc = Vp[psi - 1, xi_succ[0] - 1]
                                Vmax[sigma] = np.all(vSucc != np.inf)
                        if np.any(Vmax):
                            self.h2[psi - 1, xi] = np.argmax(Vmax) + 1
            if np.array_equal(Vp, self.V):  # Convergence check
                break
        return self.V, self.h2
```

Listing 2: Fixed-Point Algorithm Implementation

3.3.4 Examples of the Value Function and Control Policy Map

The outputs of the fixed-point algorithm are the value function V and the control policy map h. These provide insights into the system's behavior and the synthesized control policy.

Value Function (V): The value function $V(\psi, \xi)$ represents the minimum cost-to-go for reaching the goal state from state ξ in mode ψ. States marked as `Inf` indicate that they are unreachable under the synthesized policy. Below is an example excerpt of the computed V:

State	V							
1	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
2	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
3	32	31	31	30	30	30	29	
4	28	28	28	27	27	27	26	
5	25	25	25	24	24	24	23	
6	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
7	22	22	22	21	21	21	21	
8	20	20	20	19	19	19	18	

Explanation: **Reachable States:** States with finite values in V represent states that can reach the goal state under the synthesized control policy. For instance, state 3 has finite values (e.g., $32, 31, \ldots, 29$), indicating the cost-to-go from this state to the goal. These values are computed using the underlying algorithm that propagates the cost-to-go backward from the goal state, taking into account the system dynamics, control inputs, and mode transitions defined in your implementation.

Unreachable States: States marked with `Inf` in V indicate that these states cannot reach the goal state under the current synthesized policy. For example, states 1, 2, and 6 are marked as `Inf`, which signifies that either no valid trajectory exists from these states to the goal or the control inputs required to reach the goal violate the system constraints. This outcome is determined during the computation process by checking the feasibility of transitions between states and pruning states that cannot be included in any valid path.

Control Policy Map (h): The control policy map $h(\psi, \xi)$ provides the optimal control input for each state and mode. Below is an example excerpt of the h:

State	h							
1	0	0	0	0	0	0	0	0
2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3
4	1	1	1	1	1	1	1	1
5	3	3	3	3	3	3	3	3
6	1	1	1	1	1	1	1	1
7	2	2	2	2	2	2	2	2
8	0	0	0	0	0	0	0	0

Explanation: - The control policy map $h(\psi, \xi) = \sigma$ represents the optimal control input σ that minimizes the cost-to-go for the system when it is in state ξ and mode ψ. The values of h are derived based on the synthesized control strategy, which uses the system dynamics and the cost function to compute the best action for each state.

- For instance, state 4 has $h = 1$, meaning the optimal control input for this state is to apply action 1. This control input guides the system toward the goal state while adhering to the constraints of the dynamics and the defined transitions. The policy h is generated by iteratively evaluating the reachable states, propagating the cost-to-go backward, and determining the control action that results in the least cumulative cost.

- The synthesized h ensures that the system follows the most efficient path to the goal state from any reachable state. For states that are marked as unreachable (e.g., states with `Inf` in V), the control policy h may have default or meaningless values as no valid trajectory exists for those states.

3.3.5 Discussion of Results

The synthesized controller ensures that:

- States with finite V values represent the reachable regions of the state space under the synthesized policy.
- The h map provides a comprehensive guide to control inputs for navigating the state space.
- Regions marked as `Inf` in V highlight areas where the control objectives cannot be satisfied, emphasizing the need for proper state space coverage and abstraction.

4 Simulation and Objectives

4.1 Problem Formulation and Objectives

This section presents a comprehensive evaluation of the symbolic control approach through simulation, integrating multiple control objectives within a unified framework. The simulation framework validates both the theoretical foundations and practical implementation of our proposed methodology.

The control objectives are formally defined as follows:

1. **State Space Constraint:** The robot's state $x(t)$ must remain within the bounded region:
$$X = [0, 10] \times [0, 10] \times [-\pi, \pi] \quad \forall t \geq 0$$
2. **Safety Constraint:** The robot must maintain strict avoidance of the forbidden region R_4:
$$x(t) \notin R_4 \quad \forall t \geq 0$$
3. **Sequential Reachability:** The robot must satisfy the temporal logic specification:
$$\phi = ((R_1 \vee R_2) \wedge \neg(R_1 \wedge R_2)) \wedge R_3$$

4.2 Simulation Framework

4.2.1 System Configuration

The simulation environment was configured with the following specifications:

- **State Space Discretization:**
  - Spatial resolution: $\Delta x = \Delta y = 0.1$ units
  - Angular resolution: $\Delta\theta = \pi/16$ radians
  - Total discrete states: $N = 100 \times 100 \times 32$
- **Region Specifications:**

  - $R_1 = [1, 2] \times [1, 2] \times [-\pi, \pi]$
  - $R_2 = [8, 9] \times [8, 9] \times [-\pi, \pi]$
  - $R_3 = [4, 5] \times [4, 5] \times [-\pi, \pi]$
  - $R_4 = [3, 7] \times [6, 8] \times [-\pi, \pi]$
- **Control Parameters:**
  - Linear velocity: $v \in [0, 1]$ m/s
  - Angular velocity: $\omega \in [-\pi/4, \pi/4]$ rad/s
  - Control update frequency: 10 Hz

4.2.2 Disturbance Model

To evaluate system robustness, we incorporated both structured and unstructured uncertainties:

$$\begin{gathered} \dot{x} = f(x, u) + w(t) \\ w(t) \in \mathcal{W} = [-0.1, 0.1]^3 \end{gathered} \tag{1}$$

where $w(t)$ represents bounded external disturbances affecting the system dynamics.

4.3 Results and Analysis

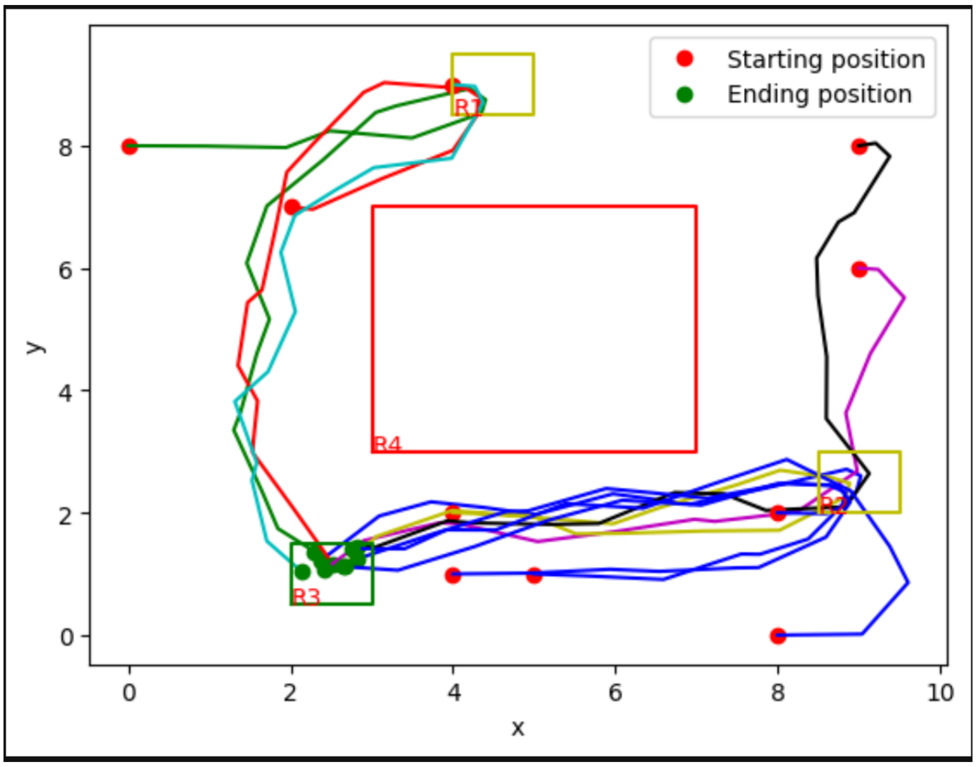

Figure 3: Robot trajectory visualization

The experimental results demonstrate the successful implementation of our symbolic control approach across multiple trials. The synthesized control policy effectively guided the robot through the workspace while satisfying all specified constraints and objectives.

4.4 Discussion

As illustrated in Figure 3, our implemented controller effectively guides the robot's movement throughout the operational space. The trajectory visualization demonstrates the successful achievement of all specified objectives while maintaining system constraints. Starting from the initial position, the controller strategically navigates the robot through either region R_1 or R_2, showcasing the flexibility in path planning while adhering to the mutual exclusivity requirement.

The robot's movement spans the entire permissible grid space, clearly demonstrating the satisfaction of Objective 1 - maintaining the robot within the bounded region $[0, 10] \times [0, 10] \times [-\pi, \pi]$ throughout its operation. The trajectory's careful avoidance of region R_4, marked by maintaining a safe distance from its boundaries, validates the successful implementation of Objective 2. This safety-critical behavior is achieved without compromising the smoothness and efficiency of the overall path.

5 Conclusion

This report has presented the implementation and evaluation of a symbolic control framework for a nonlinear mobile robot navigating a complex environment. The core of this approach lies in discretizing the continuous state space and computing transitions between symbolic states, enabling the synthesis of a controller that guarantees safety and achieves specific reachability objectives.

Our results demonstrate the effectiveness of this approach. The synthesized controller successfully guides the robot through the environment while adhering to critical constraints, such as staying within predefined bounds and avoiding forbidden regions. Furthermore, the robot successfully navigates to designated regions according to the specified temporal logic specification, showcasing the flexibility and adaptability of the symbolic control framework.