SYMBOLIC CONTROL

Of a non linear mobile robot

AHMED QUADIMI

TARIK OUABRK

MANAL ZAIDI

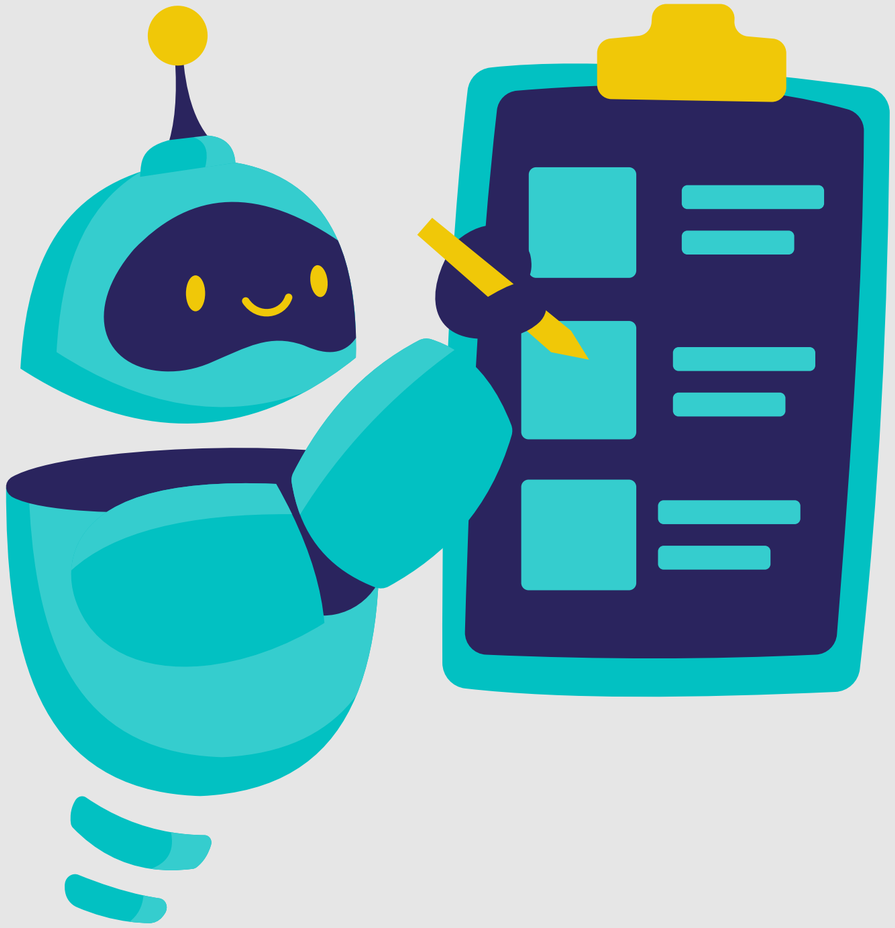
SALIM QADDA

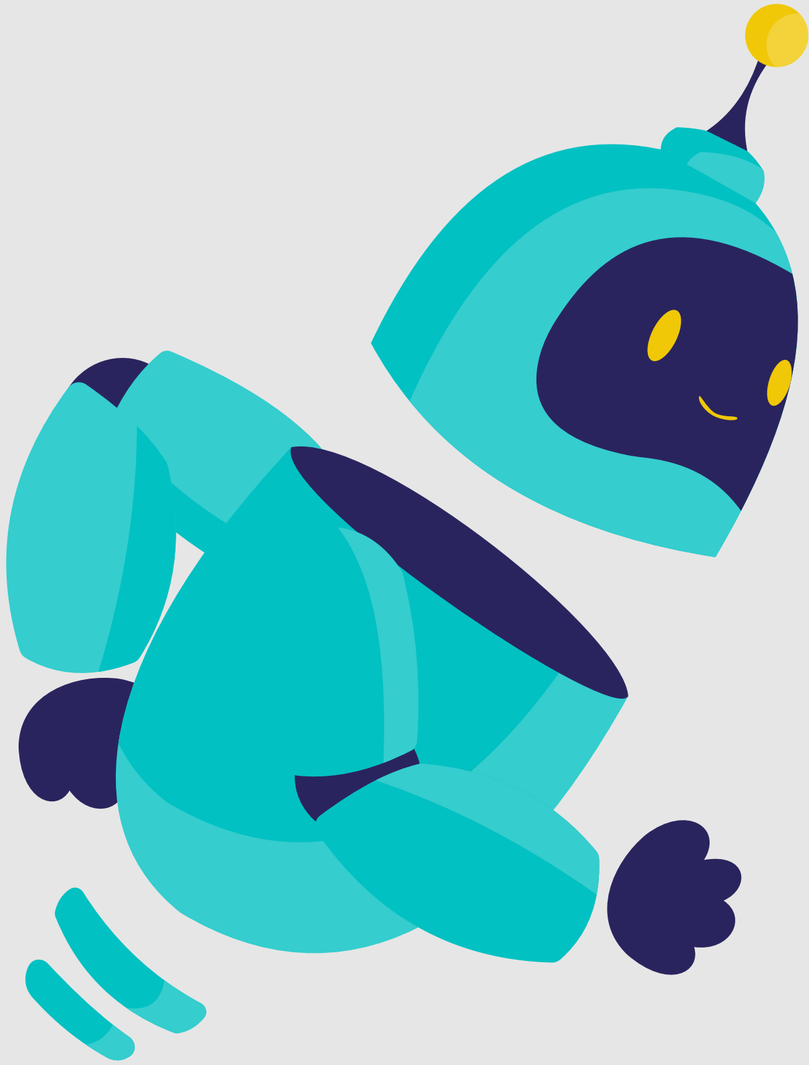
INTRODUCTION

FINITE ABSTRACTION CONSTRUCTION

CONTROLLER SYNTHESIS FOR MULTIPLE OBJECTIVES

VALIDATION THROUGH EXTENSIVE NUMERICAL SIMULATIONS

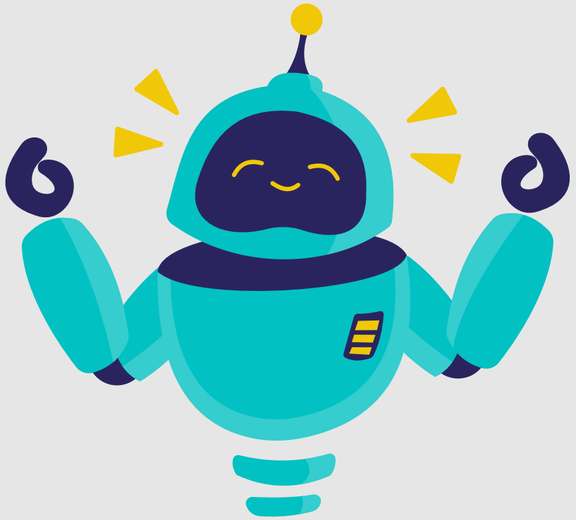




2D

WHY DO WE USE 2D INSTEAD OF 3D?

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



COMPLEXITY AND DEBUGGING

2D

OUR MAIN FUNCTION

$$g(\xi,\sigma)=\left\{\xi^{+}\in\Xi\mid\mathbb{Y}_{\xi,\sigma}\cap\mathrm{cl}\left(\mathbb{X}_{\xi+}\right)\neq\emptyset\right\}$$

THE ISSUE? COMPUTING ALL STATES TO CHECK IF THE CONDITION IS VALID TAKES TOO MUCH TIME

2D

INTERVAL CONCRETIZATION

$$(i_{\min}, j_{\min}) = q\left(Y^{\min}\right),$$

$$(i_{\max}, j_{\max}) = q\left(Y^{\max}\right),$$

$$g(\xi, \sigma) = \left\{(i, j) \;\middle|\; i_{\min} \leq i \leq i_{\max},\ j_{\min} \leq j \leq j_{\max}\right\}.$$

2D

PRE STRUGGLE: TAKES TOO LONG

```
def Pre_naive(R):
    result = set()
    for element in all_states:
        for k in range(1, Nvx * Nvy + 1):
            if g(element, k).issubset(R):
                result.add(element)
                break
    return result
```

2D

SOLUTION: USE G-1

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

Let $\xi \in Pre(R)$. Then $\exists \sigma \in \Sigma$ such that $g(\xi, \sigma) \subseteq R$. Take $u = p(\sigma)$.

Let $\xi' \in g(\xi, \sigma) \subseteq R$ and let $y \in q^{-1}(\xi')$.

So $\exists x \in X$, $\exists w \in W$ such that

$$y = f(x, u, w) = x + Tu + Tw,$$

hence

$$x = y - Tu - Tw = f(y, -u, -w).$$

We have

$$\text{W is symmetric around } 0 \implies f(y, -u, -w) \in \left[\, f(y, -u, \underline{w}),\ f(y, -u, \overline{w}) \,\right]$$

Take

$$g^{-1}(\xi', \sigma) = \left\{\, \xi'' \mid Y^{-1}_{\xi', \sigma} \cap \mathrm{cl}(X_{\xi''}) \neq \right\},$$

with

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

2D

SOLUTION: USE G-1

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

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

Thus,

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

Now we simply iterate through R: take $\xi \in R$ and

$$\sigma \in \Sigma \ / \ \xi' \in g^{-1}(\xi, \sigma)$$

If

$$g(\xi', \sigma) \subseteq R,$$

then

$$\xi' \in Pre(R).$$

Note. The mapping g is not assumed to be bijective. The notation g^{-1} used above does *not* denote an inverse function;

Proof written by

Salim Qadda

2D REACHABILITY

COMPUTES PRE FOR THE SAME STATE DURING DIFFERENT ITERATIONS: REDUNDANT

Entrées : $Q_a \subseteq \Xi$

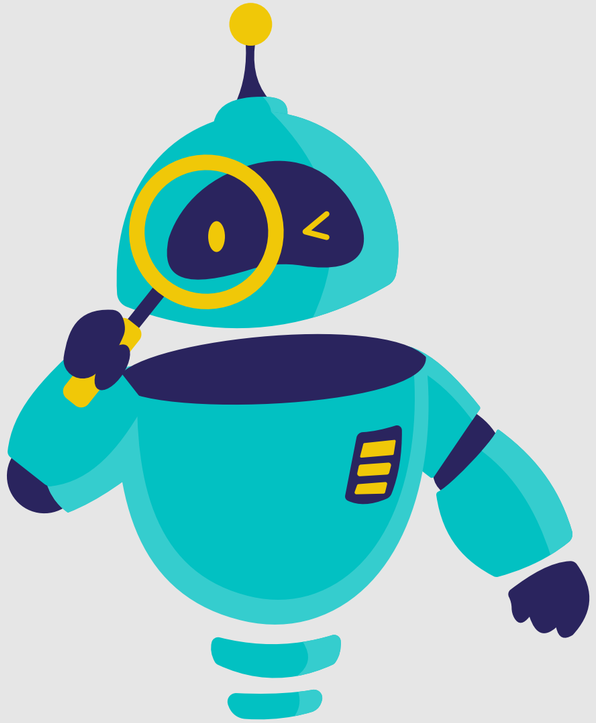
$R_0 = Q_a$

répéter

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

jusqu'à $R_{k+1} = R_k$;

Sorties : $R^* = R_k$



2D

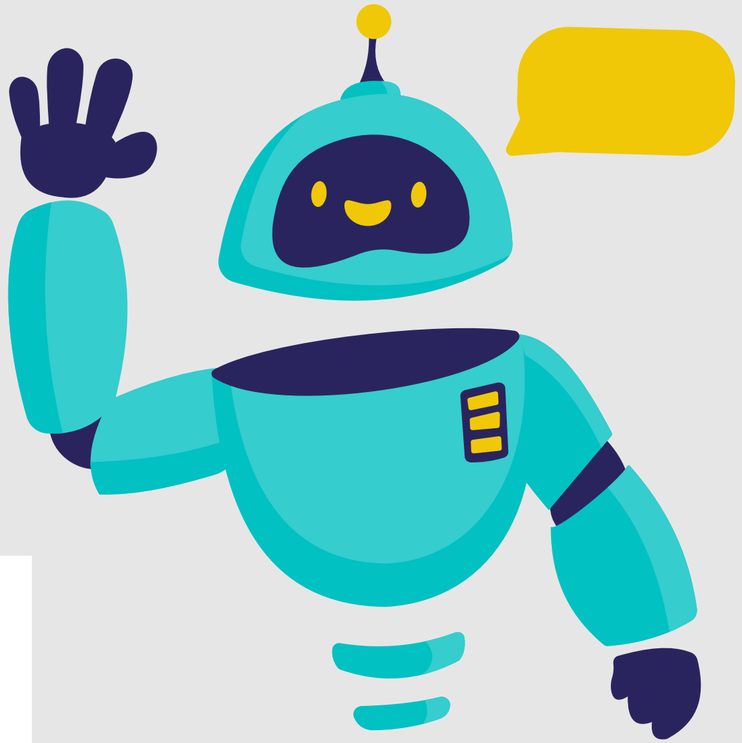
REACHABILITY

WE CAN USE A QUEUE

2D PRODUCT SYSTEM

SAME LOGIC, BUT WE DEFINE OUR STATE:

$$(\xi, q) \in \Xi \times Q_{\text{spec}}$$

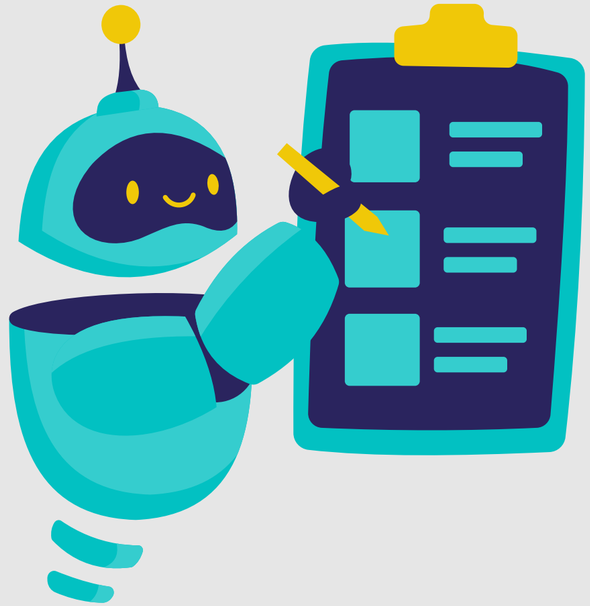


2D

REACHABILITY – PRODUCT SYSTEM

KEEP THE SAME LOGIC OF REACHABILITY WE USED BEFORE, BUT THIS TIME WE USE:

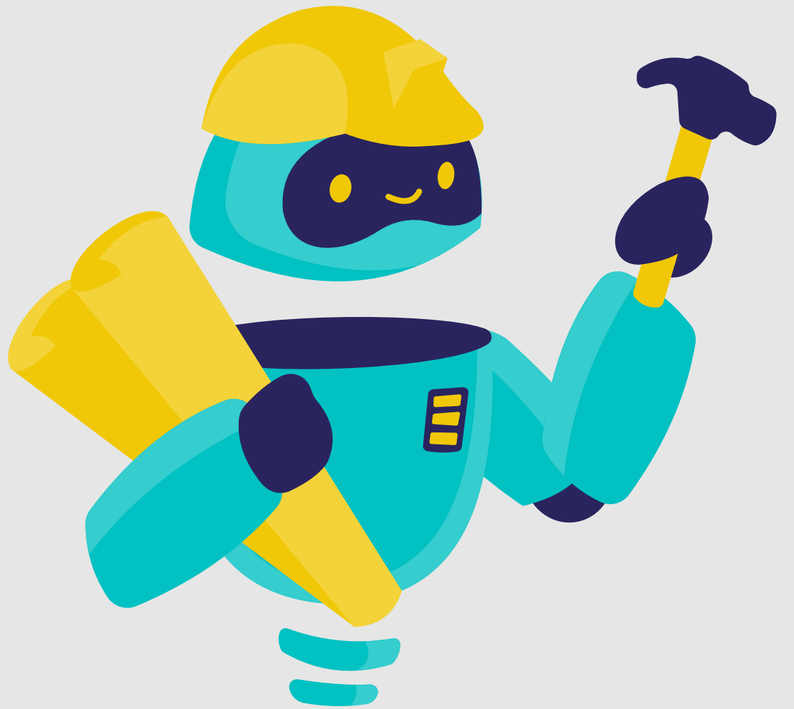
$$\mathrm{Qr} = \{(\xi, q) \in \Xi \times Q_{\mathrm{acc}}\}$$



2D CONTROLLER

WE DEFINE OUR CONTROLLER:

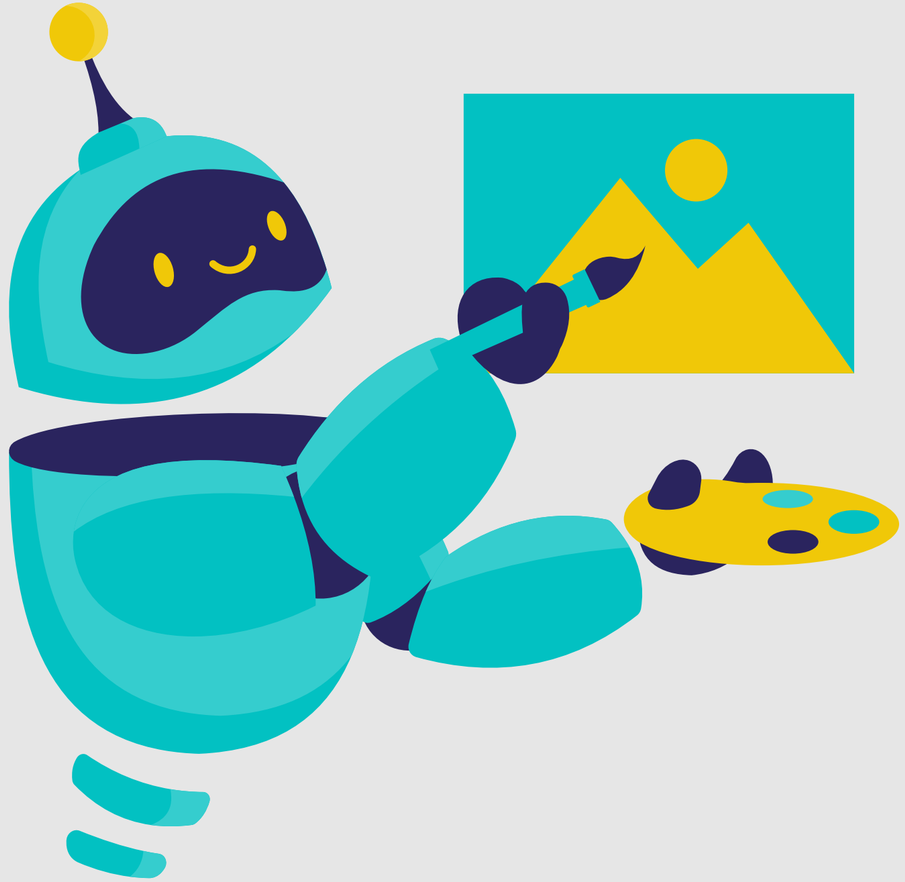
$$\mathrm{H}(\xi) = \{\, \sigma \in \{1, \ldots, N_u\} \mid g(\xi, \sigma) \subseteq R \,\},$$
$$\text{with } \mathrm{R} = \mathrm{Reachability}(\mathrm{Qr})$$

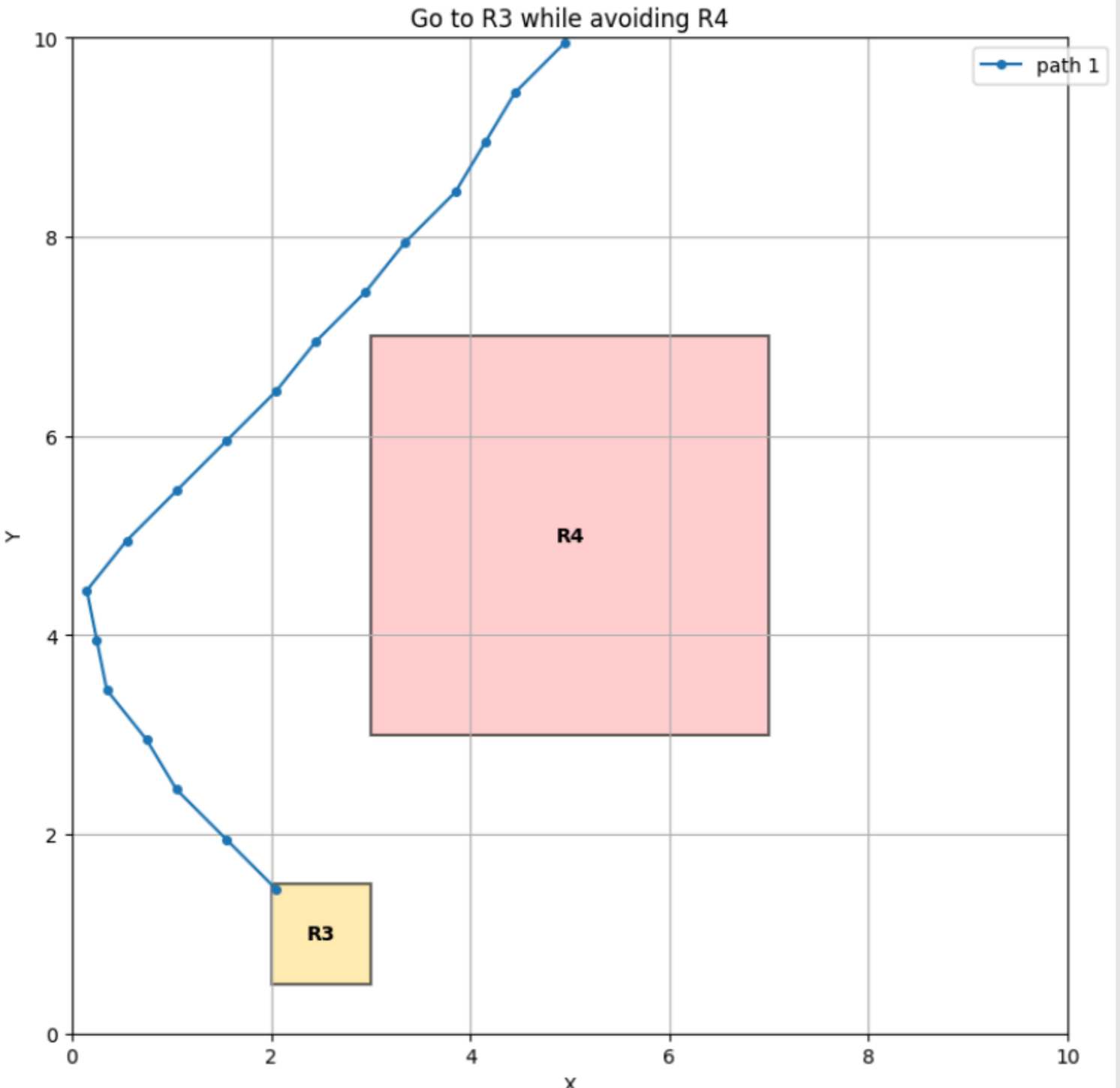


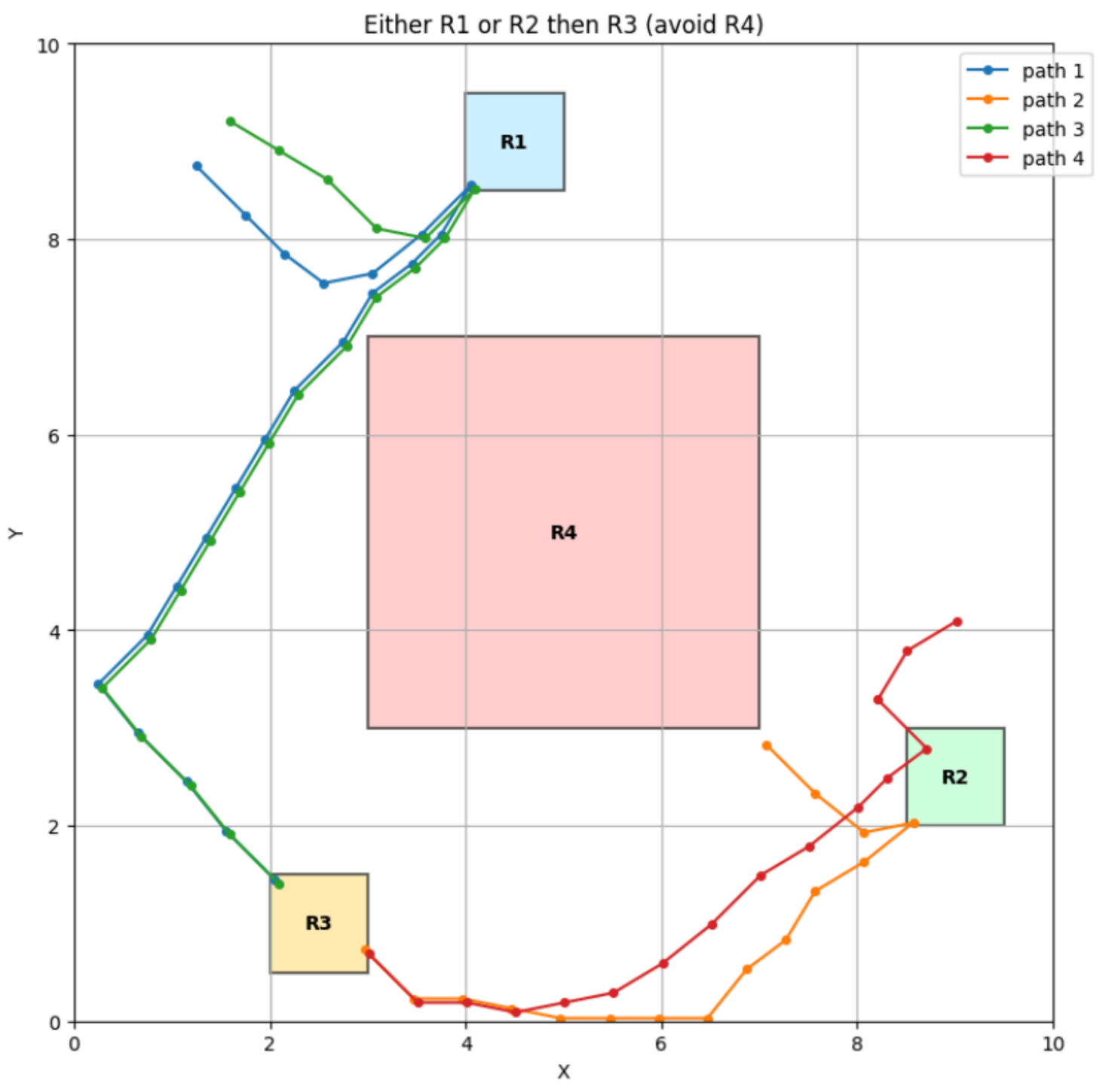
2D

TIME TO VISUALIZE

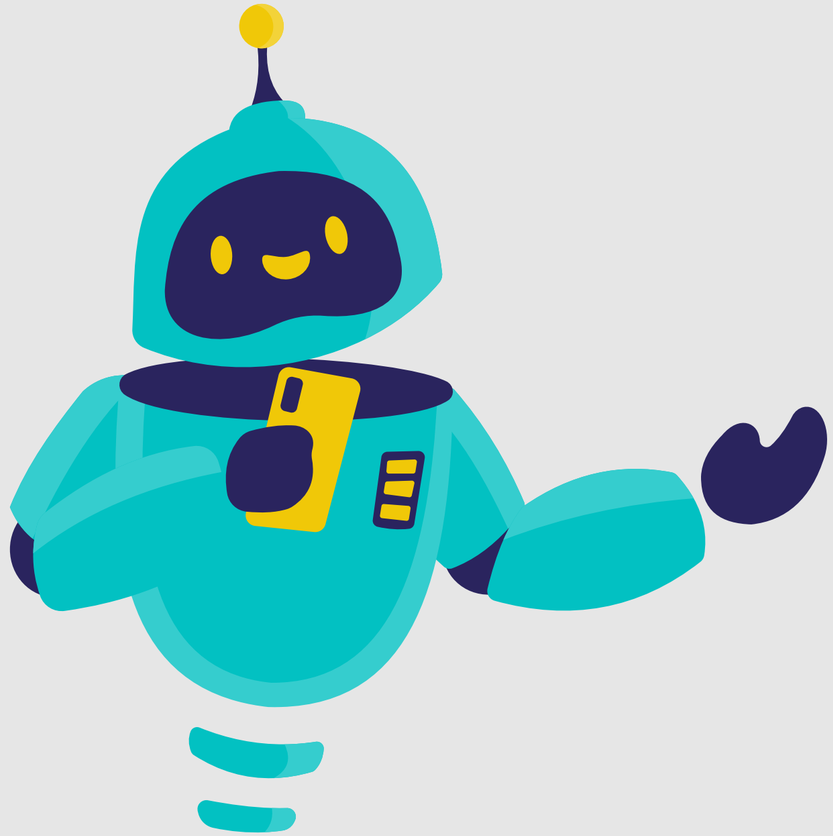
WE APPLY BFS ON OUR CONTROLLER FOR CLEAN VISUALIZATION



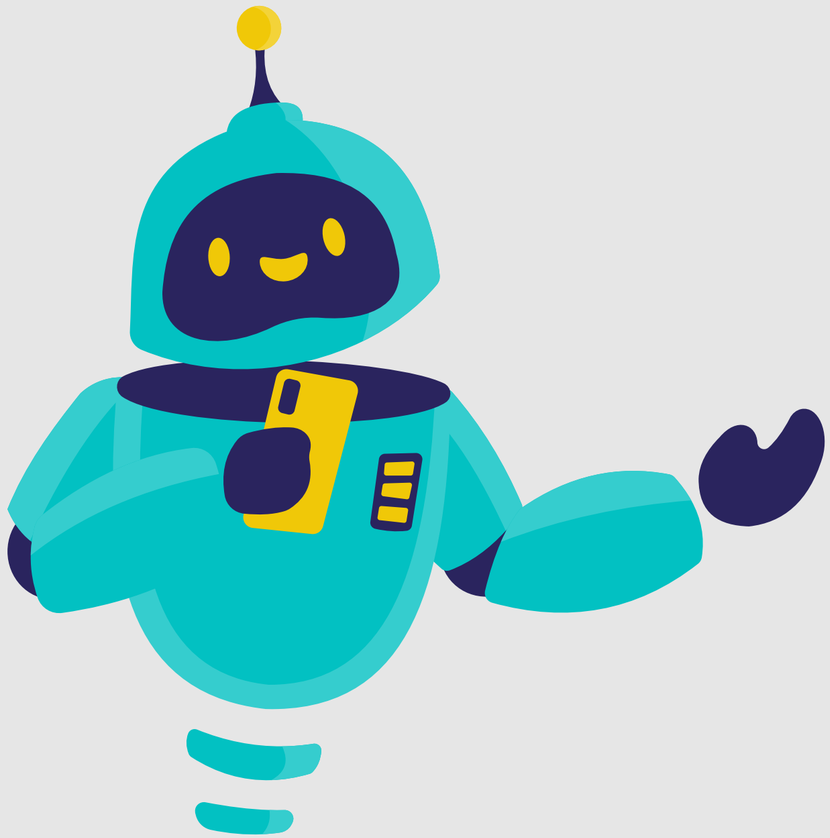

Go to R3 while avoiding R4
path 1
R4
R3
X
Y
0
2
4
6
8
10


Either R1 or R2 then R3 (avoid R4)
path 1
path 2
path 3
path 4
R1
R2
R3
R4
X
Y
0
2
4
6
8
10

3D CONTROLLER



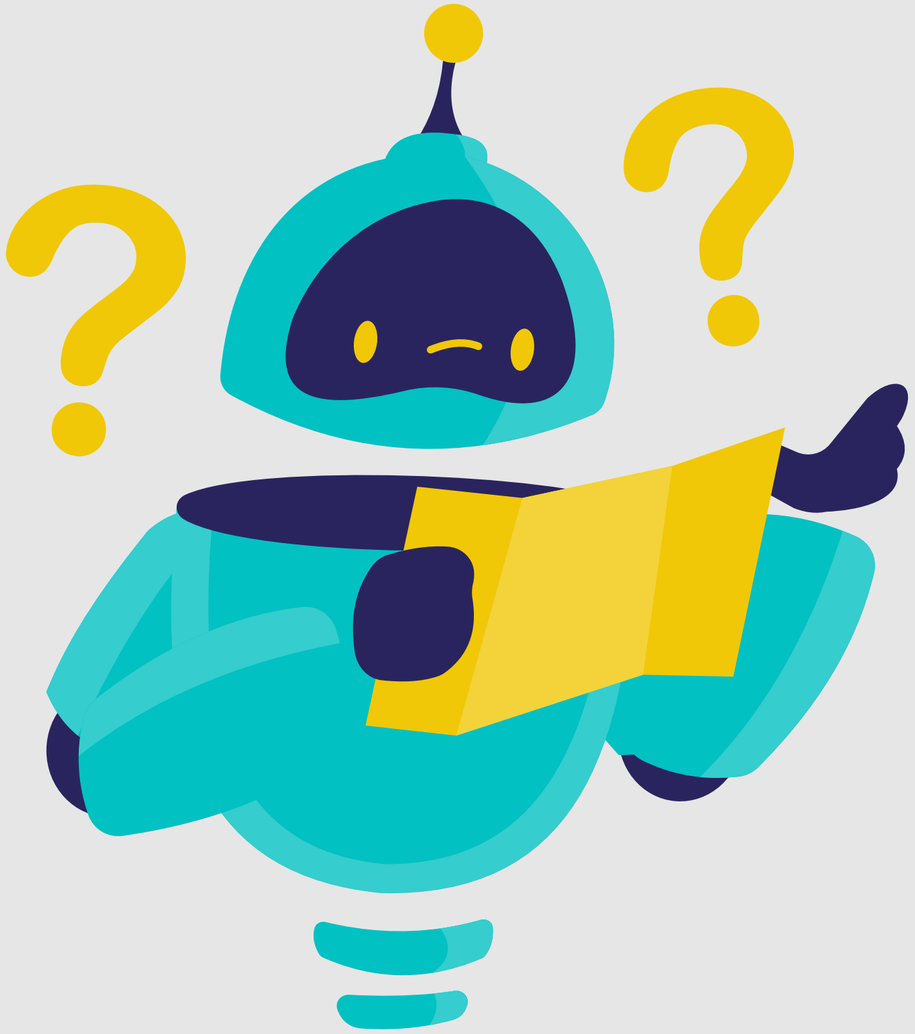
3D EQUATION



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

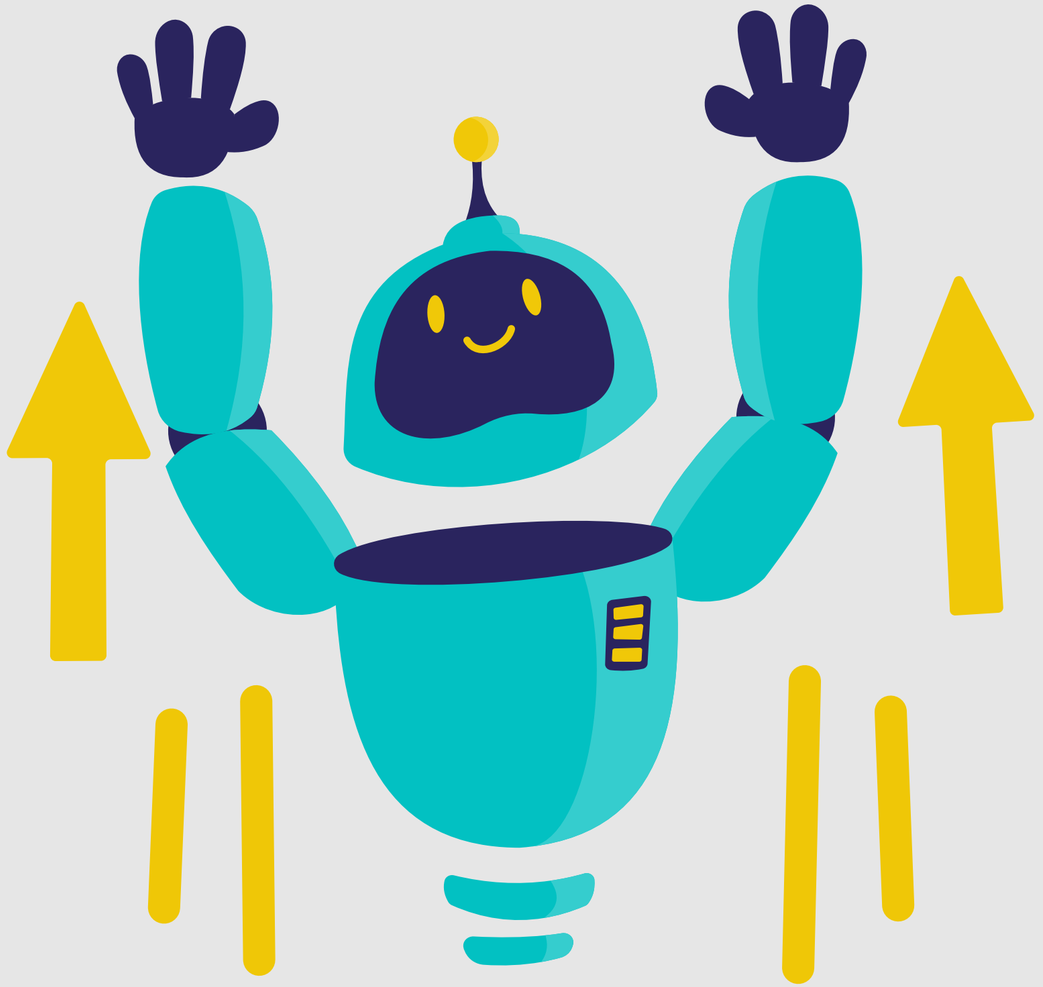
STRUGGLES

- TOO SLOW
- IT NEVER CONVERGES
- IT’S CLOSE TO R3 BUT DOESN’T REACH
- THE FILES ARE TOO BIG
- IT USES A LOT OF RAM



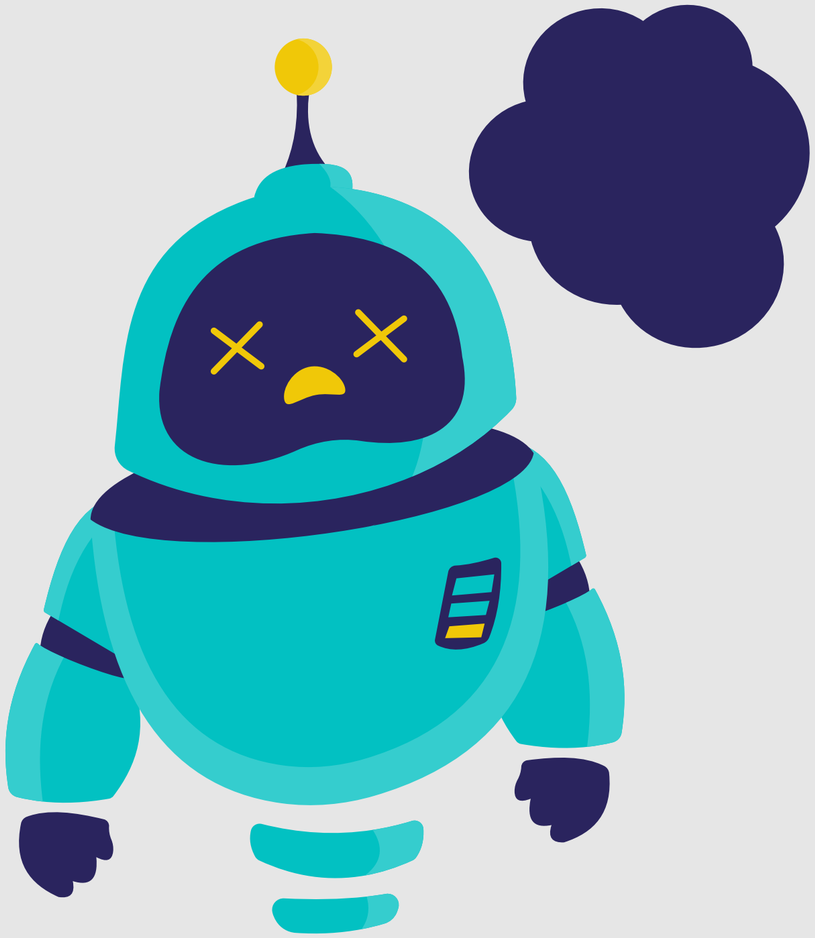
KAGGLE

WE HAD SWITCHED TO KAGGLE TO GET MORE RAM AND PROCESSOR POWER



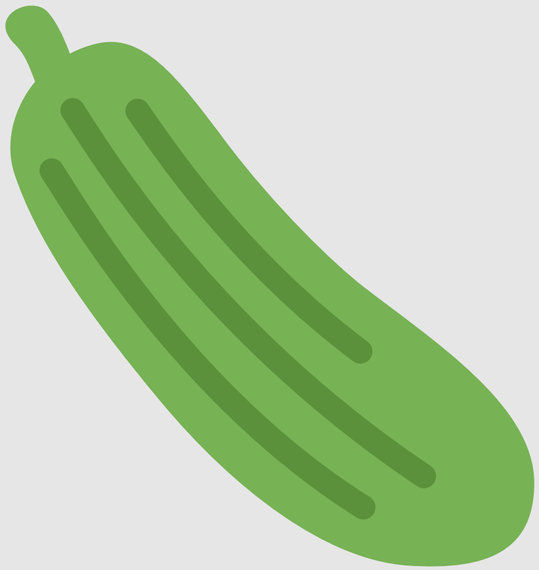
KAGGLE

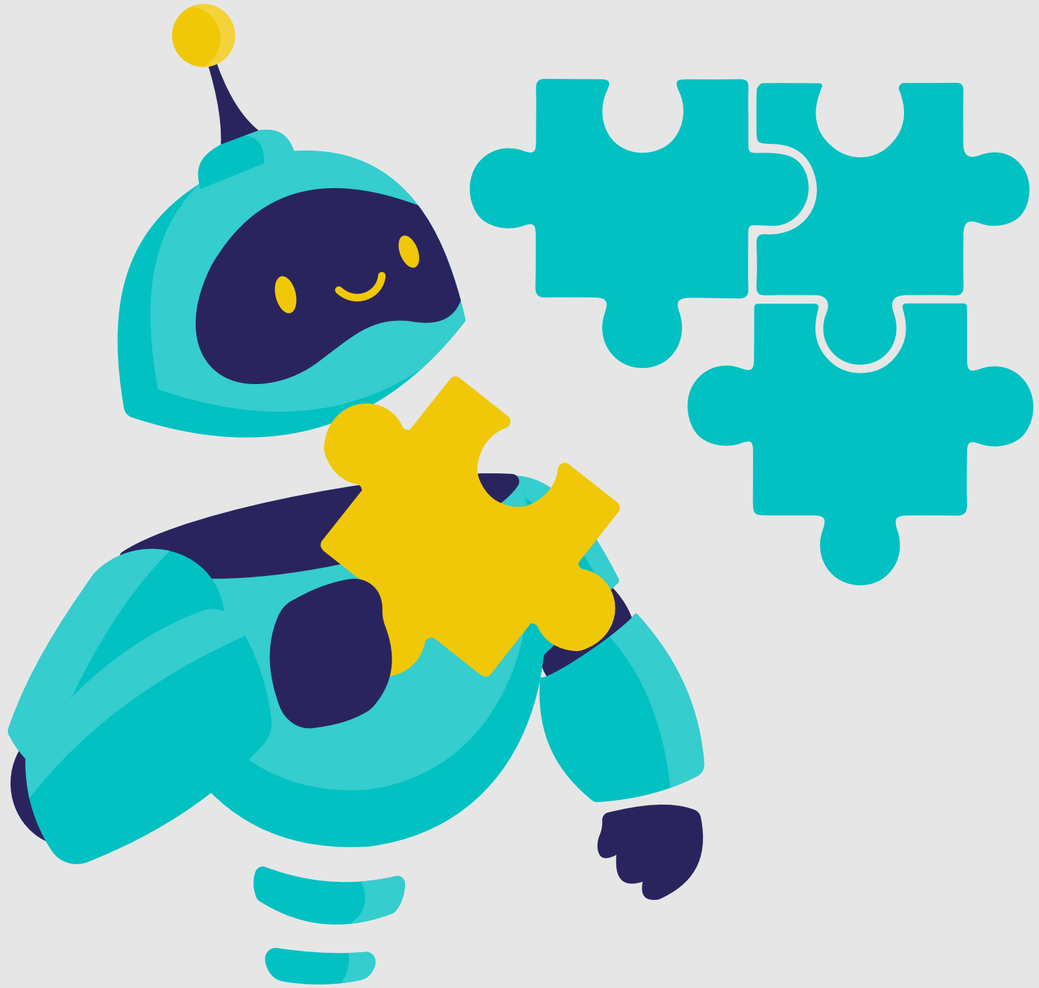
BUT OUR CODE CRASHED AFTER 13 HOURS OF RUNTIME



OPTIMISE FILE STORAGE

PICKLES





BIT MASKING ❌

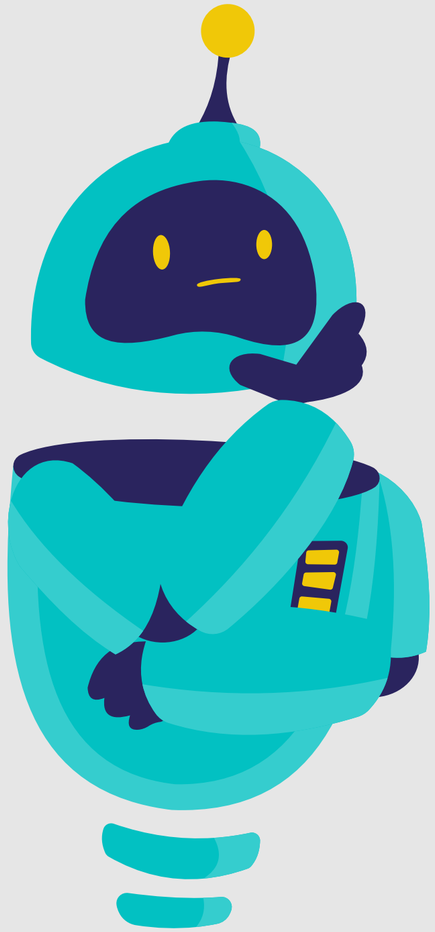
TUPLE → INT ✅

PROS:

- FAST EQUALITY CHECK
- SAVES RAM AND MEMORY

CONS:

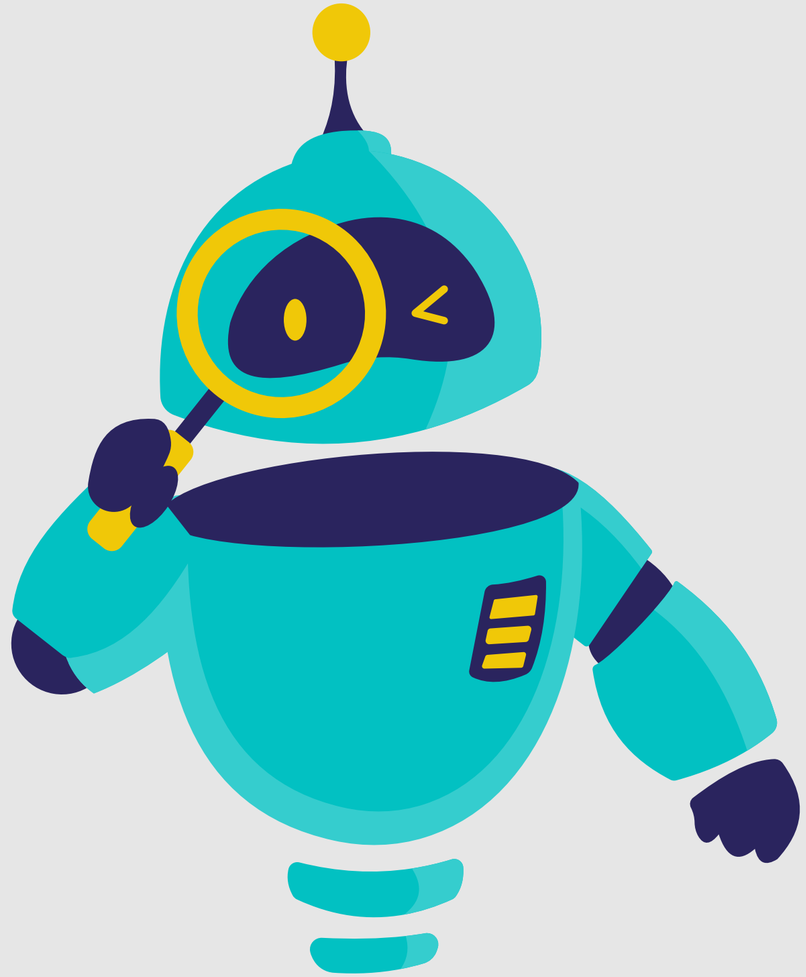
- DEBUGGING IS HARD



COMPUTE LABEL

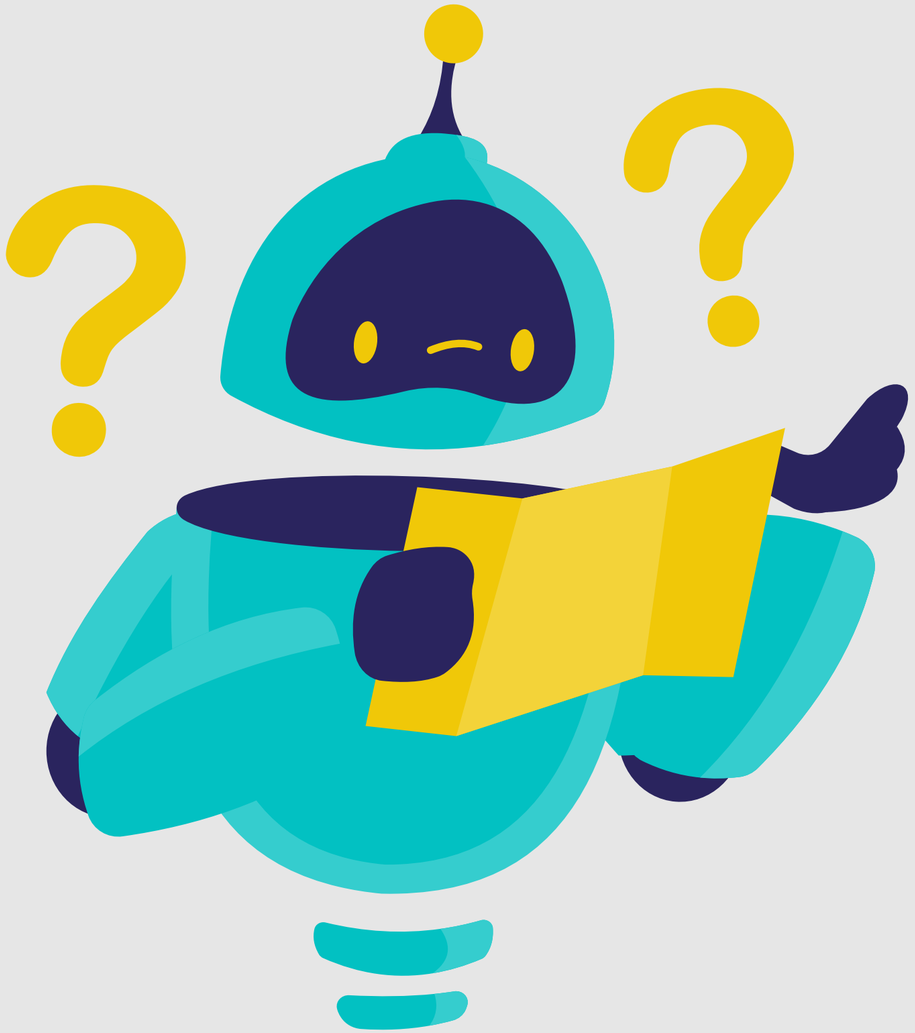
COMPUTE LABEL EACH TIME ❌

USE A LABEL GRID ✅



LOW DISCRETISATION

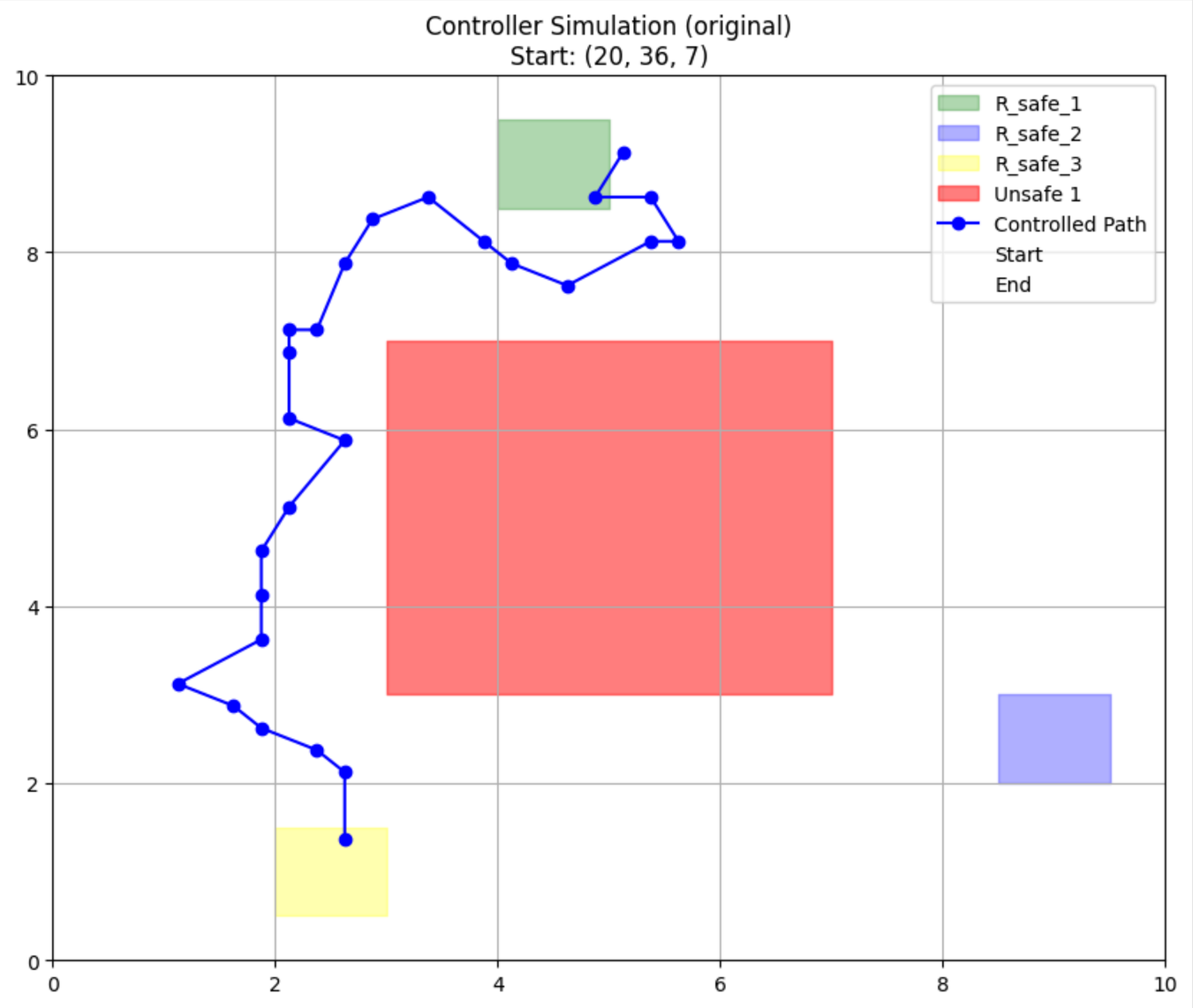
UNFORTUNATELY HERE WE DON’T HAVE A CHOICE, WE DECREASE THE DISCRETISATION



FINAL BENCHMARK

NOT COMPILING AFTER 13 HOURS ❌

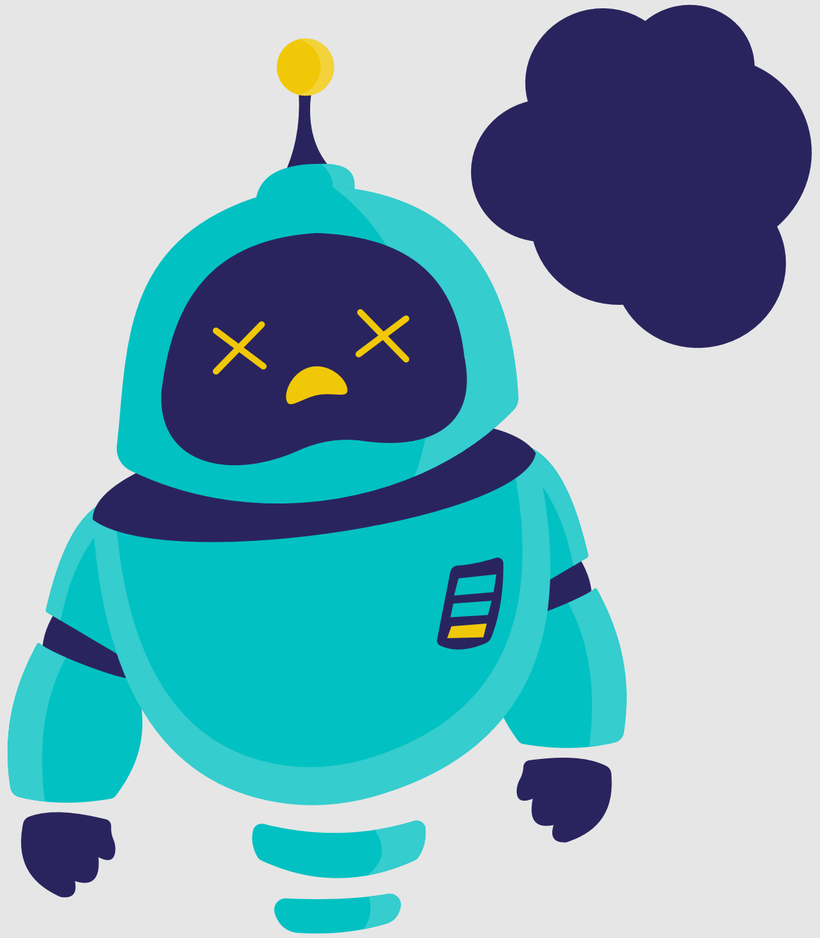
FINISHING WITHIN 90 MINUTES ✅


Controller Simulation (original)
Start: (20, 36, 7)
R_safe_1
R_safe_2
R_safe_3
Unsafe 1
Controlled Path
Start
End
0
2
4
6
8
10

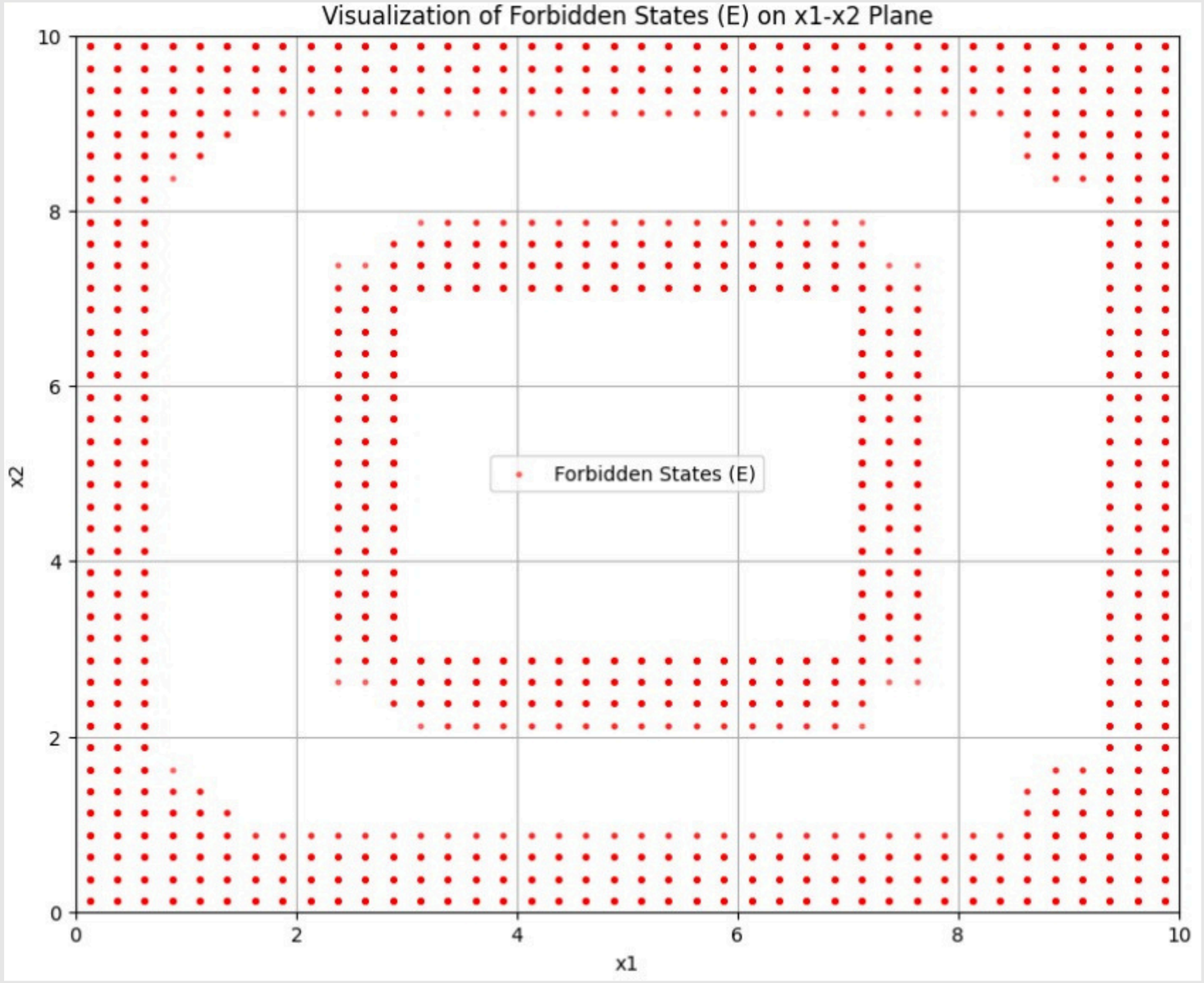
CONTINUING WITH STRUGGLES

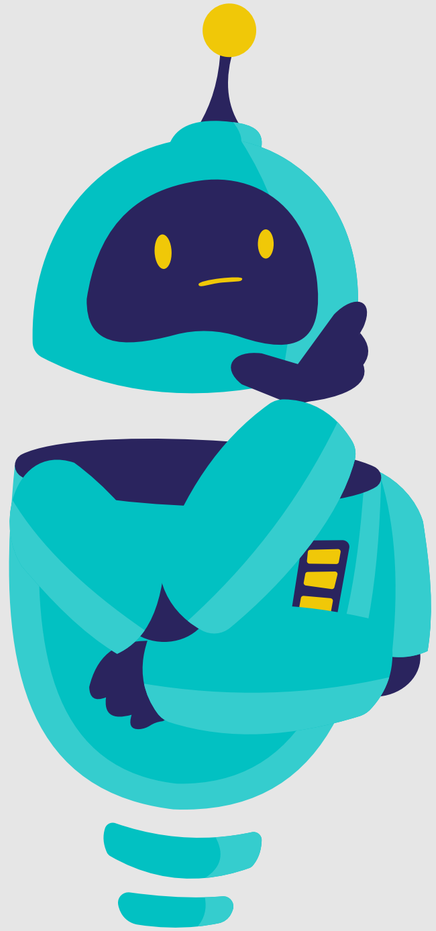
ANY VS ALL IN

REACHABILITY REQUIRES ALL,WHY?

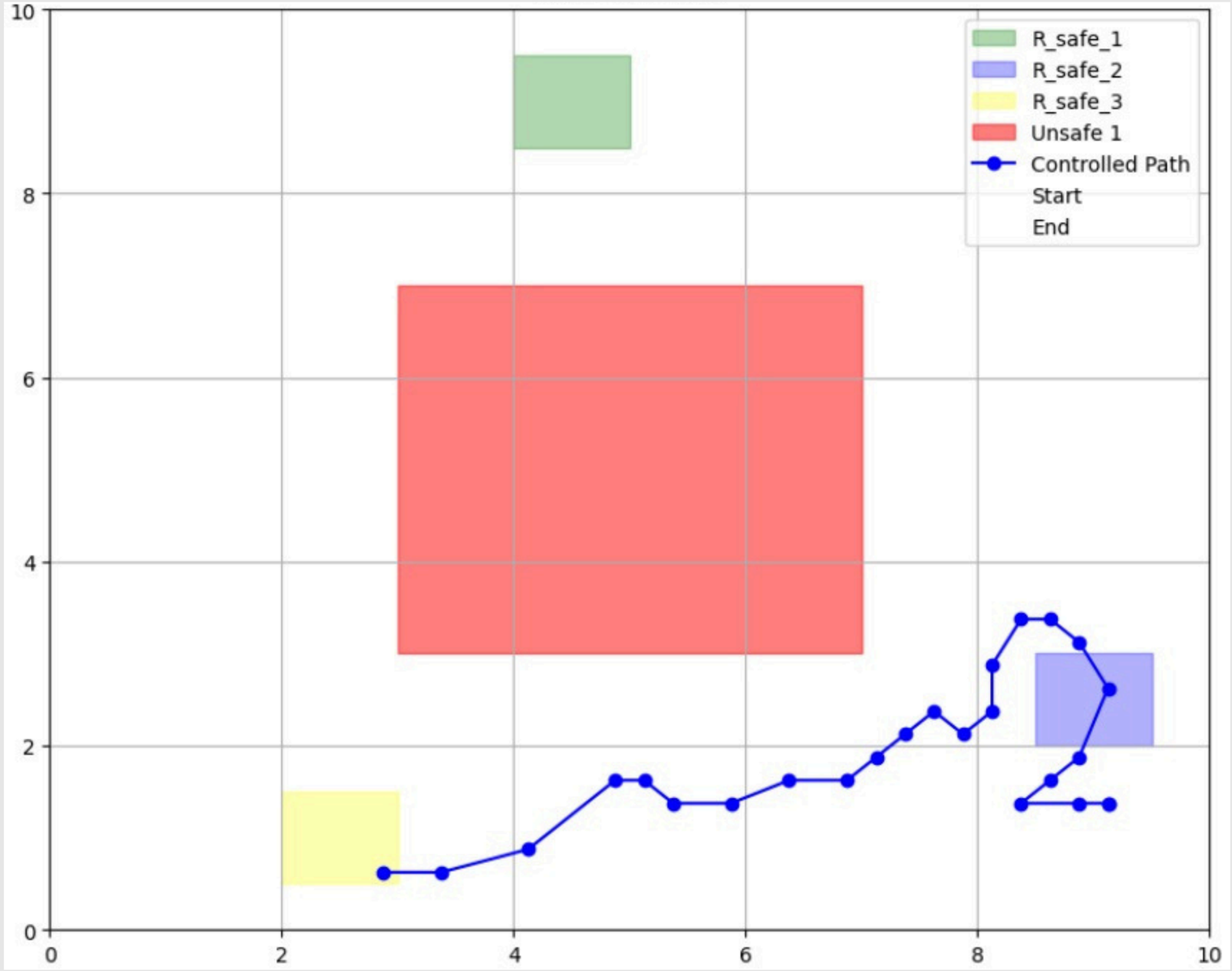


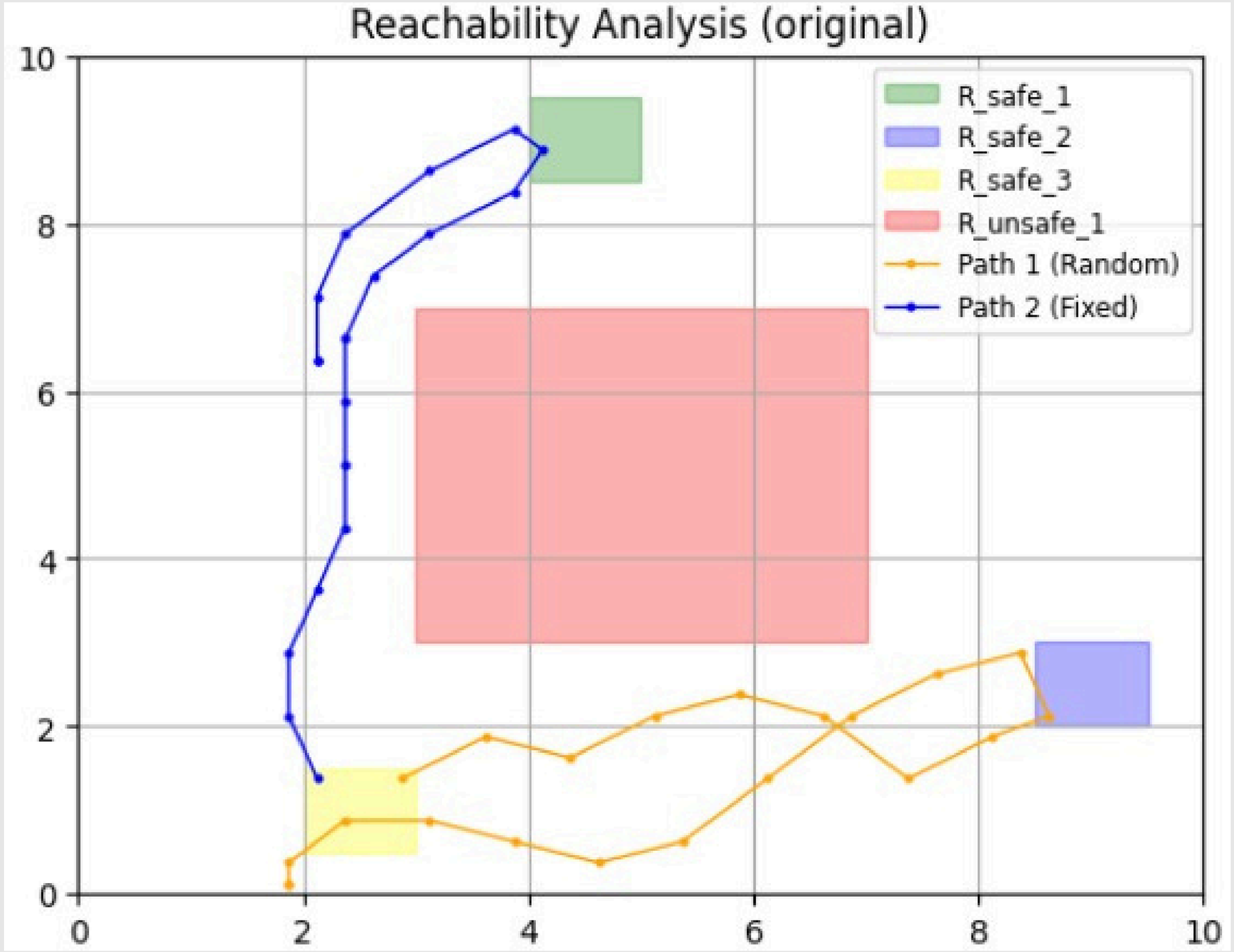
SAFETY ISSUE:PROBLEM OF CLAMPING



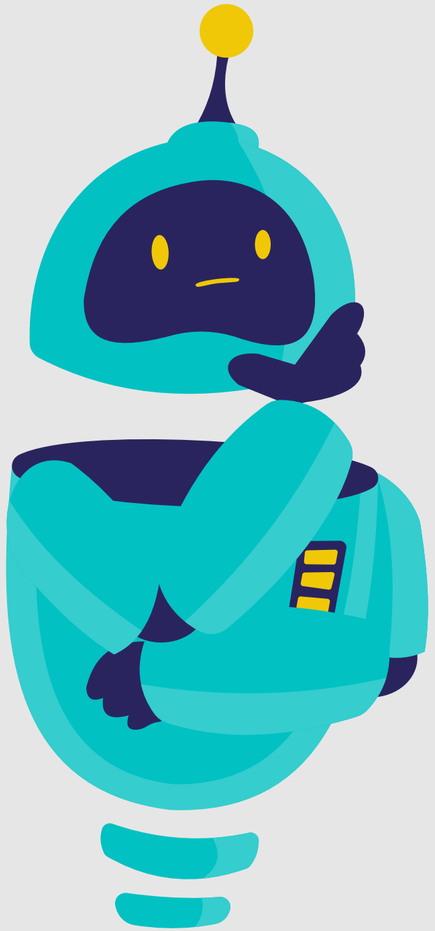


HOW TO MAKE SIMULATION LOOK BETTER?




Reachability Analysis (original)
R_safe_1
R_safe_2
R_safe_3
R_unsafe_1
Path 1 (Random)
Path 2 (Fixed)
0
2
4
6
8
10

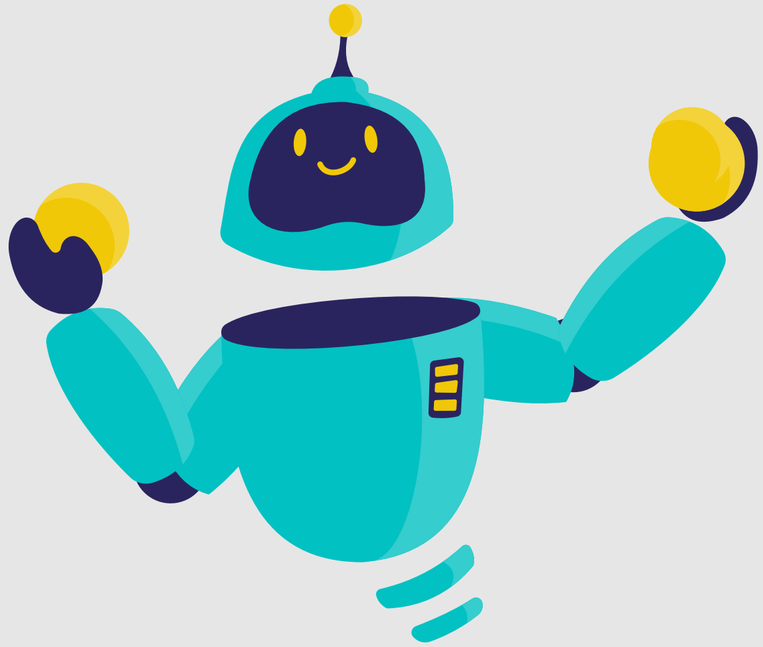
MAYBE BFS??



MULTIPLE SPECIFICATION AUTOMATA

Why multiple specs?

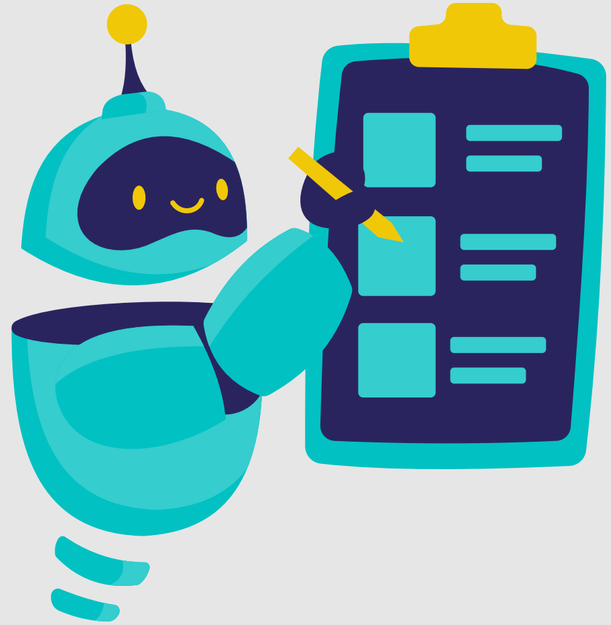
To evaluate how the controller behaves under different logical constraints,
we worked with several specification modes rather than just one.



Specification Families

Specification modes we implemented:

- **Original spec**
- **Fairness spec**
- **Visit-count spec**
- **Custom JSON-defined spec**



API Integration for Custom Automata

How it works:

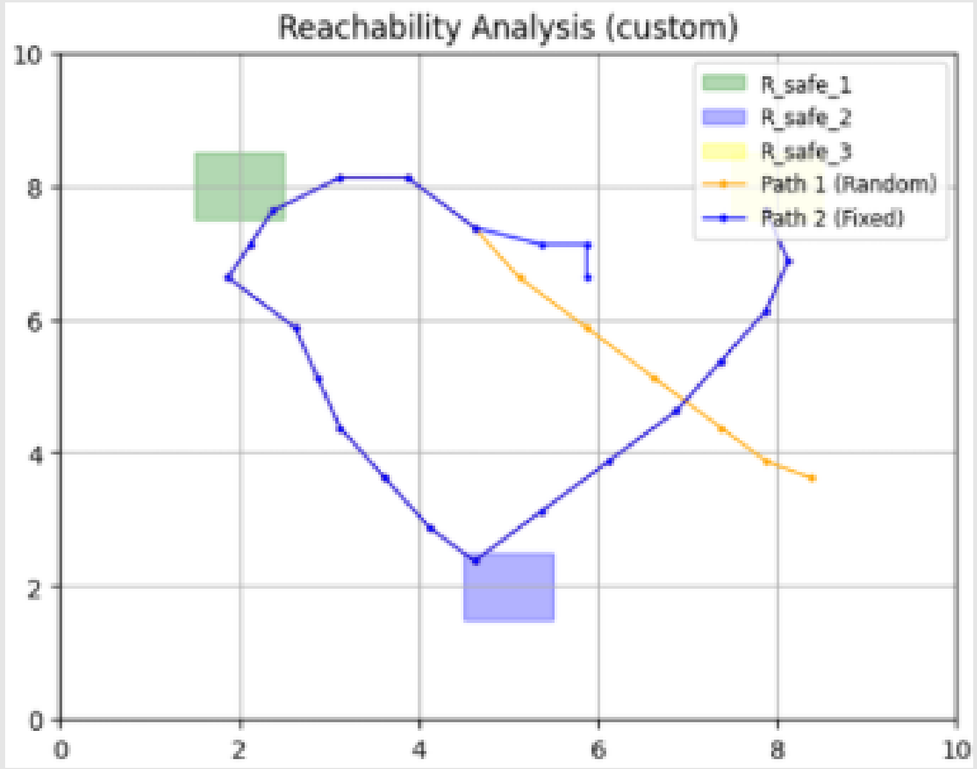
- **The user can optionally define regions or let the AI infer them.**
- **We built a superprompt for the model.**
- **The model outputs a JSON automaton matching the request.**
- **Our code processes that JSON and converts it into a controller.**

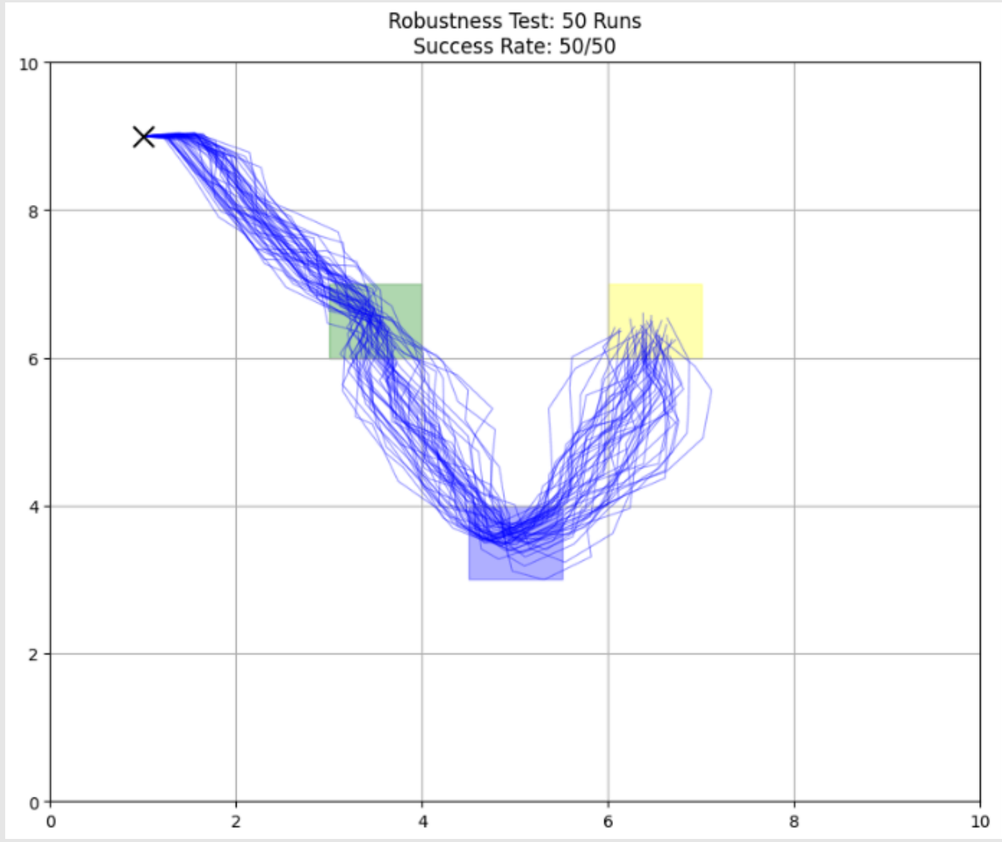
JSON format the AI outputs

```
{
  "regions": {
     "R_safe_1": [[xmin,xmax],[ymin,ymax]],
     ...
  },
  "danger_zones": {
     "R_unsafe_1": [[xmin,xmax],[ymin,ymax]],
     ...
  },
  "alphabet": ["none", ...],
  "automaton": {
      "states": [...],
      "initial": "q0",
      "accepting": [...],
      "transitions": [
         {"from": "...", "label": "...", "to": "..."},
         ...
      ]
  }
}
```

Example:

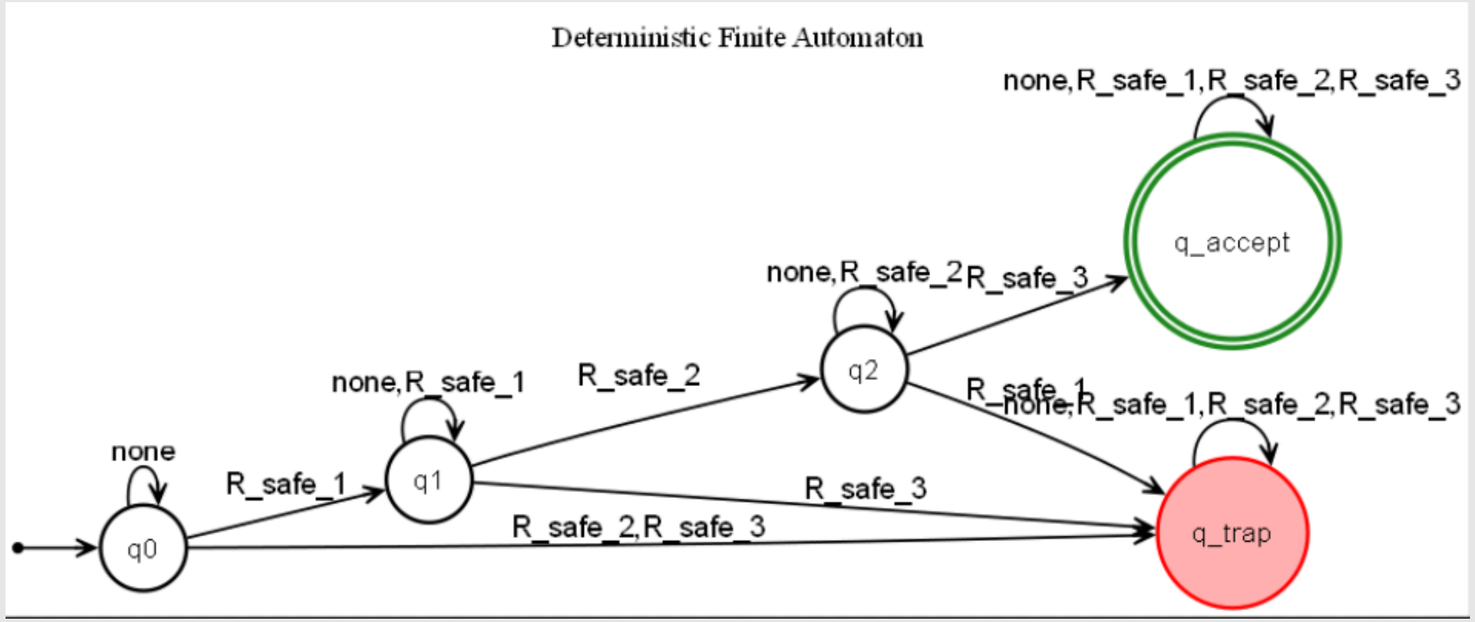
AI-generated automaton for “draw a V shape in the center of the grid”.





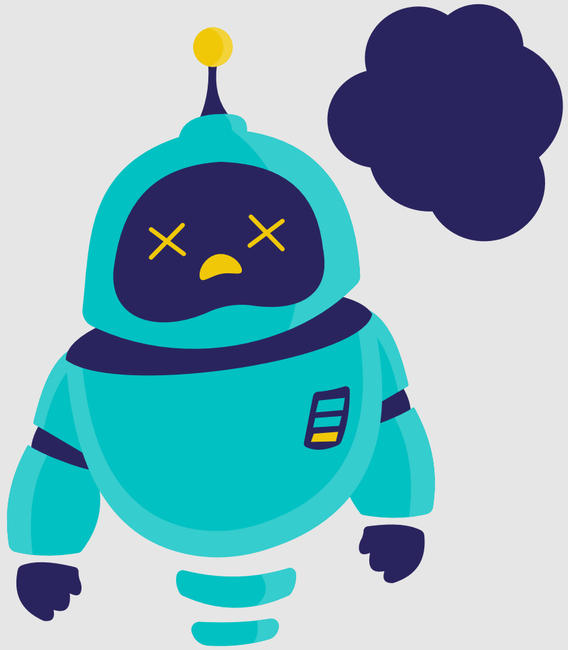
Code to visualize automaton transition table and diagram

```
=== AUTOMATON TRANSITION TABLE ===
STATE     | none      | R_safe_1 | R_safe_2 | R_safe_3
-----------------------------------------------------
q0        | q0        | q1       | q_trap   | q_trap
q1        | q1        | q1       | q2       | q_trap
q2        | q2        | q_trap   | q2       | q_accept
q_accept  | q_accept  | q_accept | q_accept | q_accept
q_trap    | q_trap    | q_trap   | q_trap   | q_trap
=====================================================
```



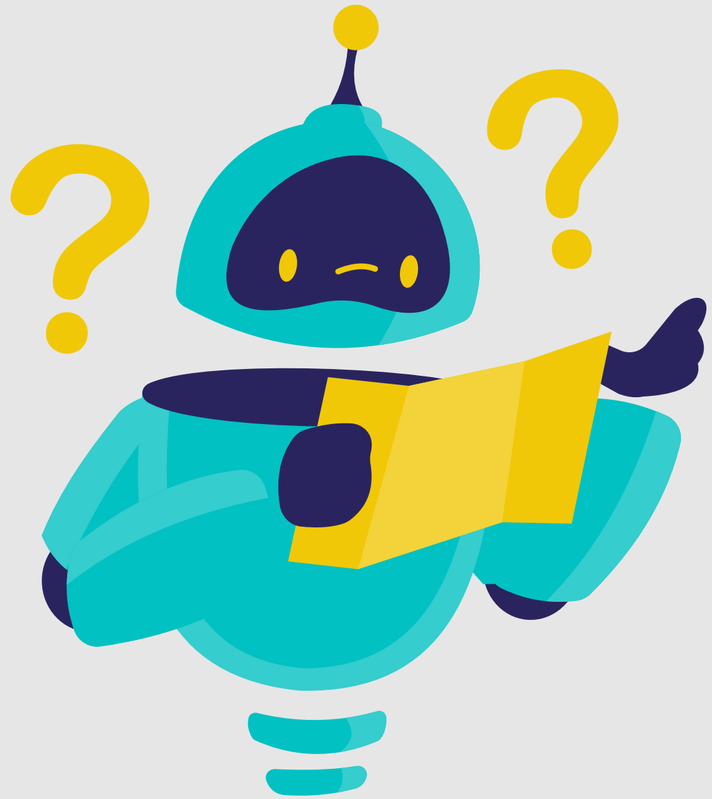
Problems faced:

- **Some automata are too strict and require extremely fine discretization, which makes them slow or even non-computable.**
- **Example of strict ordering that dramatically increase computation time in 3D but compiles in 2D.**



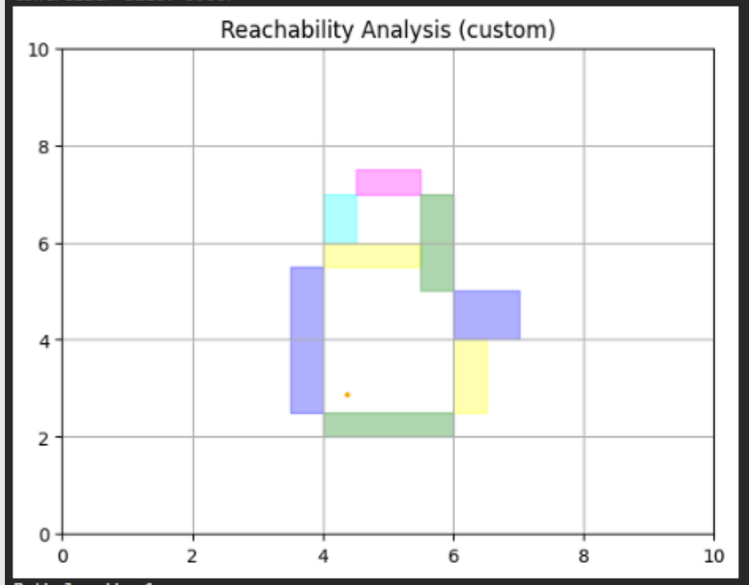
Problems faced:

- **The AI can behave unpredictably if the prompt isn’t precise enough.**
- **We built a super-prompt to limit errors, but the AI can still make mistakes.**

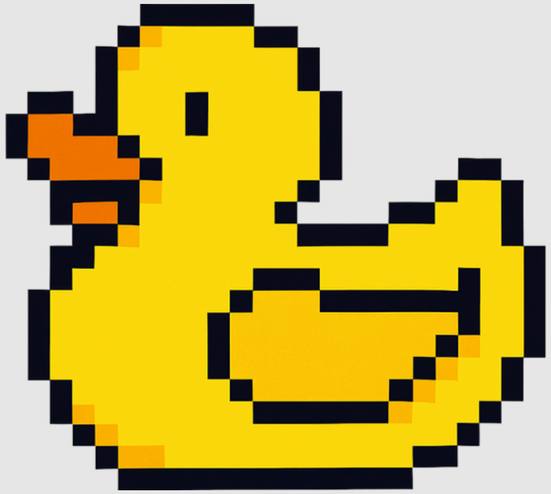


Some other spec automata just don’t work because they need too much movement

```
Picked RANDOM valid start state from controller: (np.float64(4.375), np.float64(2.875), np.float64(1.780235837034216))
Start state in winning set? False
Searching for shortest path to {'q_accept'} using BFS...
No BFS path found! Attempting greedy simulation...
No robust strategy guaranteed from (np.float64(9.625), np.float64(9.125), np.float64(-2.617993877991494))
Controller Size: 35667
```



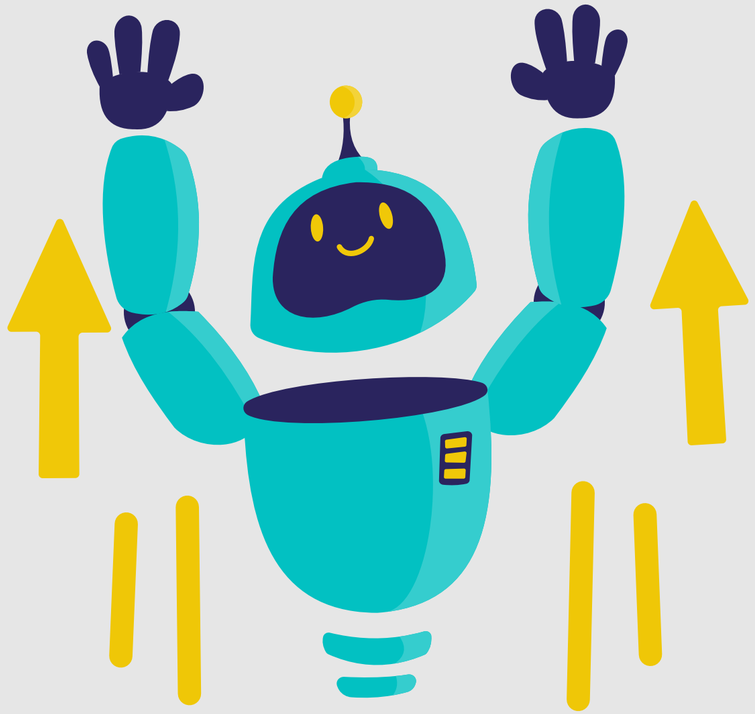

```
Path length: 1
Final state: q0
No winning states found starting at q0.
No valid start states found for q0.
```

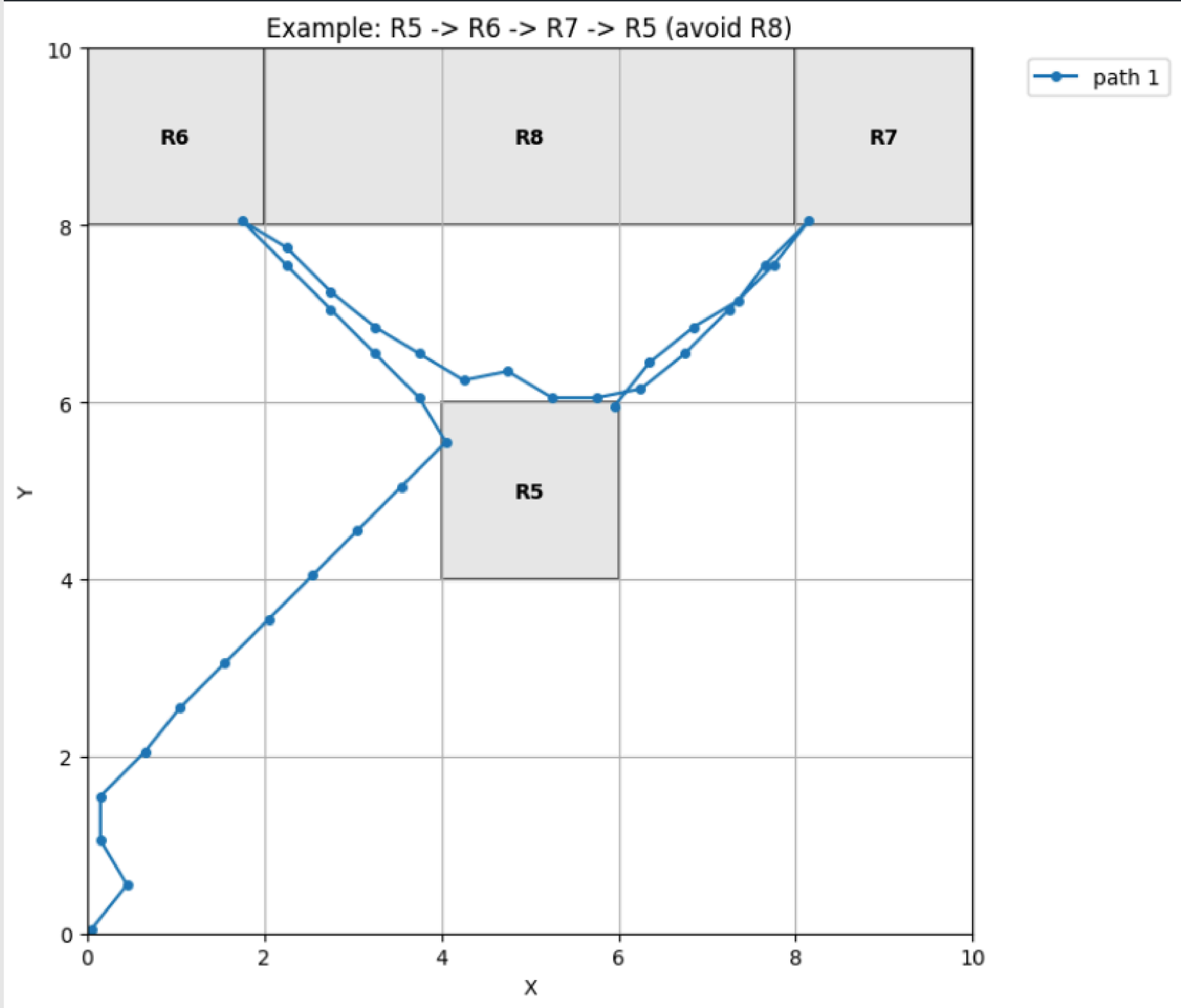


2D

DYNAMIC REGIONS

INSTEAD OF WORKING WITH R1, R2, R3 AND R4 AS THEY ARE DEFINED IN THE PAPER, LET'S GIVE THE USER THE FREEDOM TO CHOOSE ANY REGION HE WANTS TO CONTROL

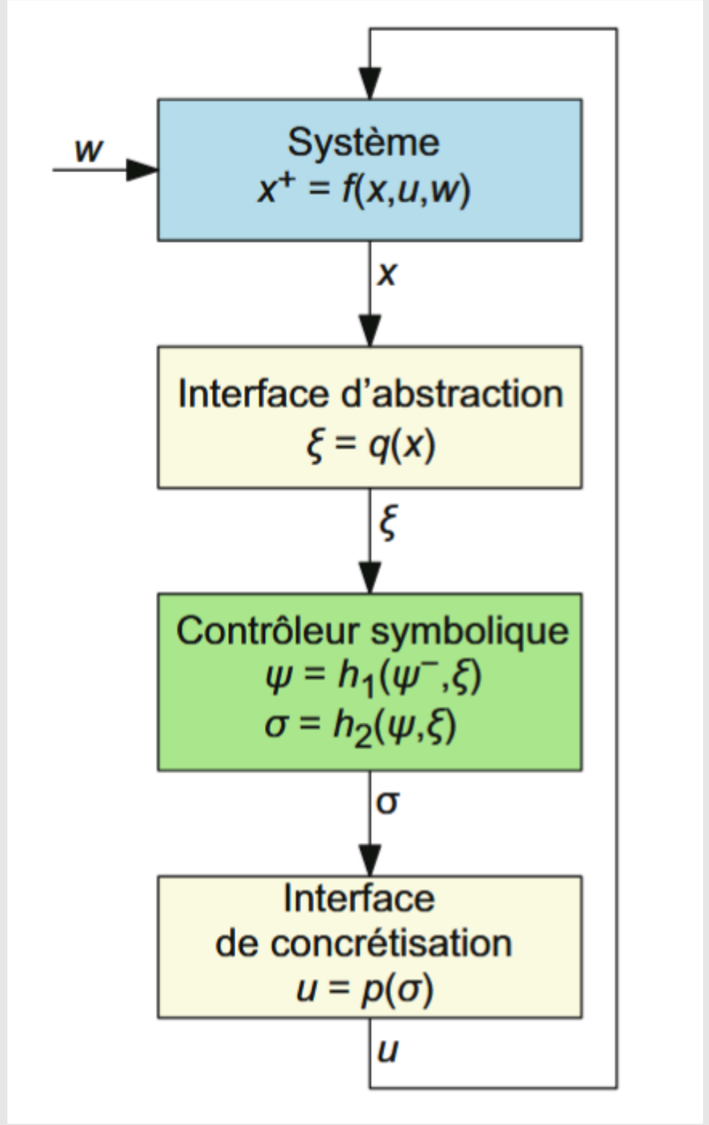


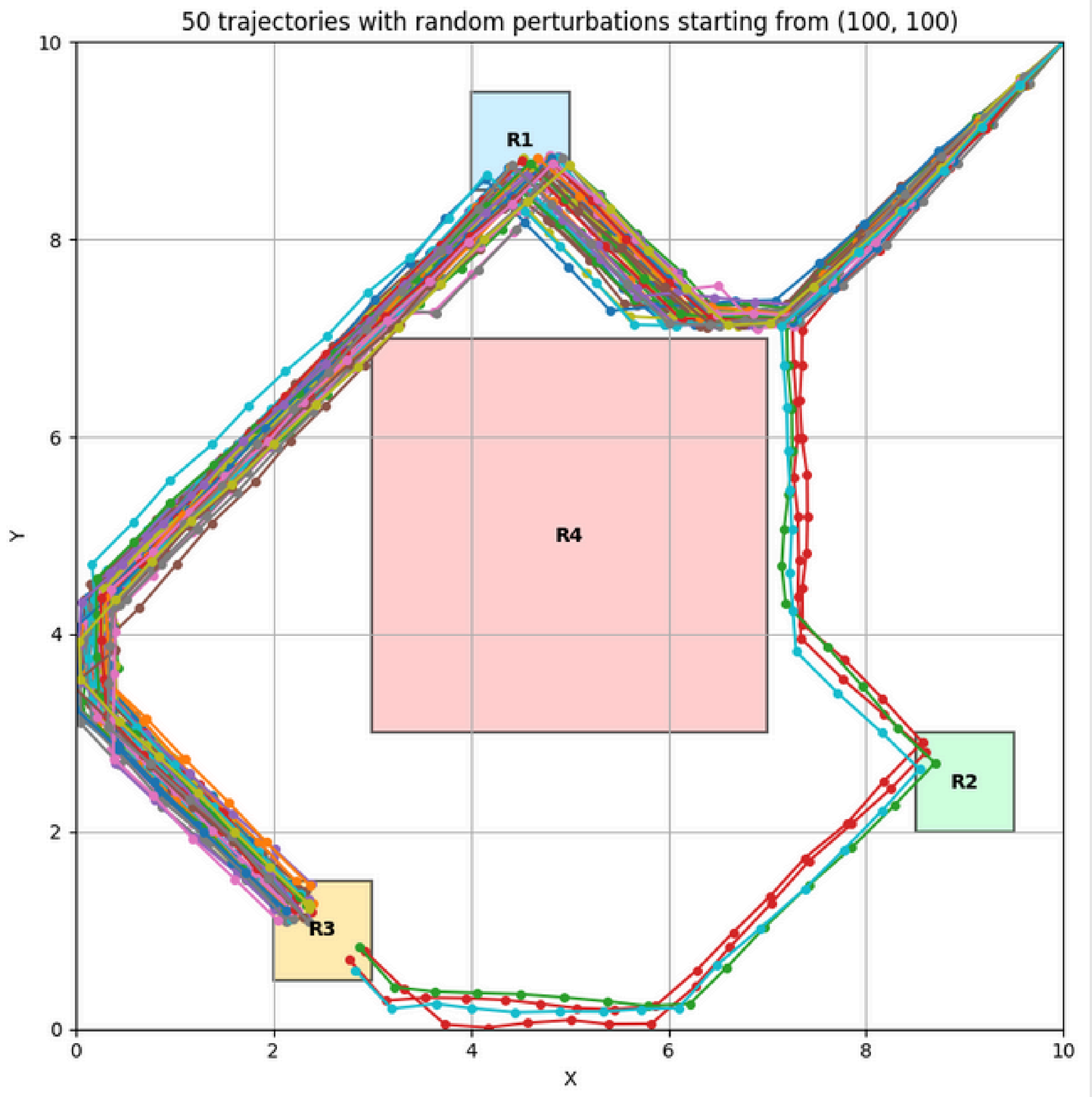

Example: R5 -> R6 -> R7 -> R5 (avoid R8)
R6
R8
R7
R5
path 1
X
Y
0
2
4
6
8
10

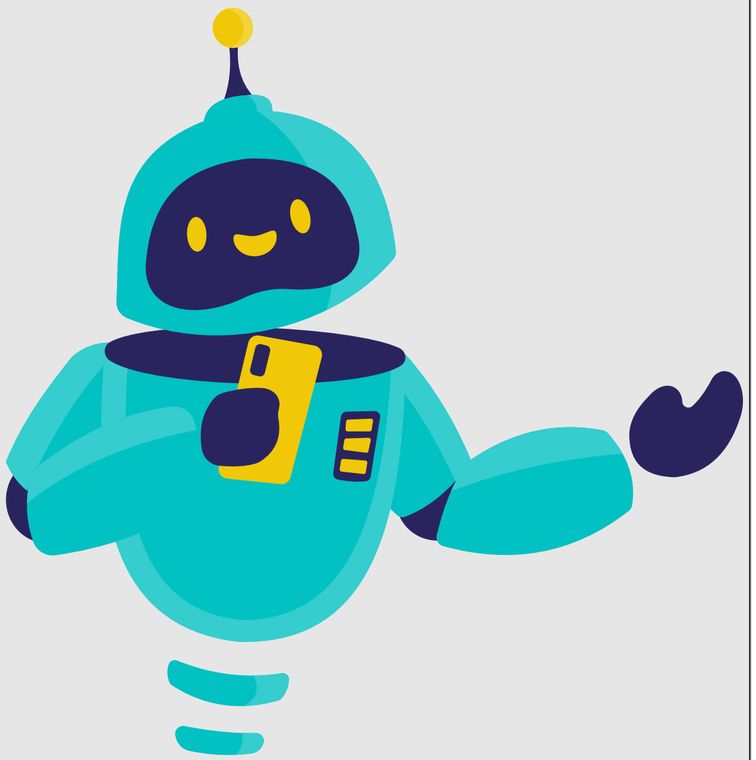
SIMULATIONS

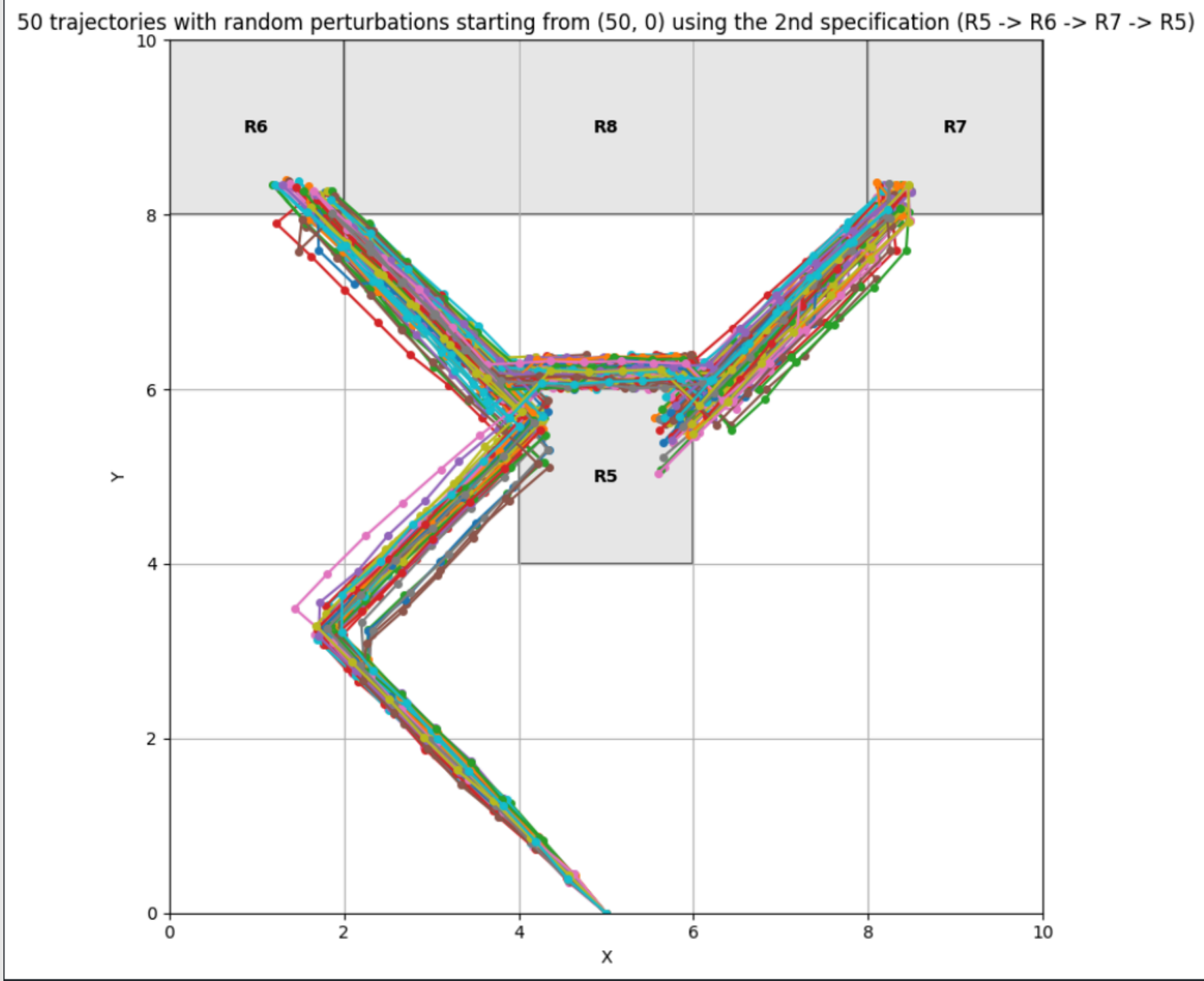
2D SIMULATIONS

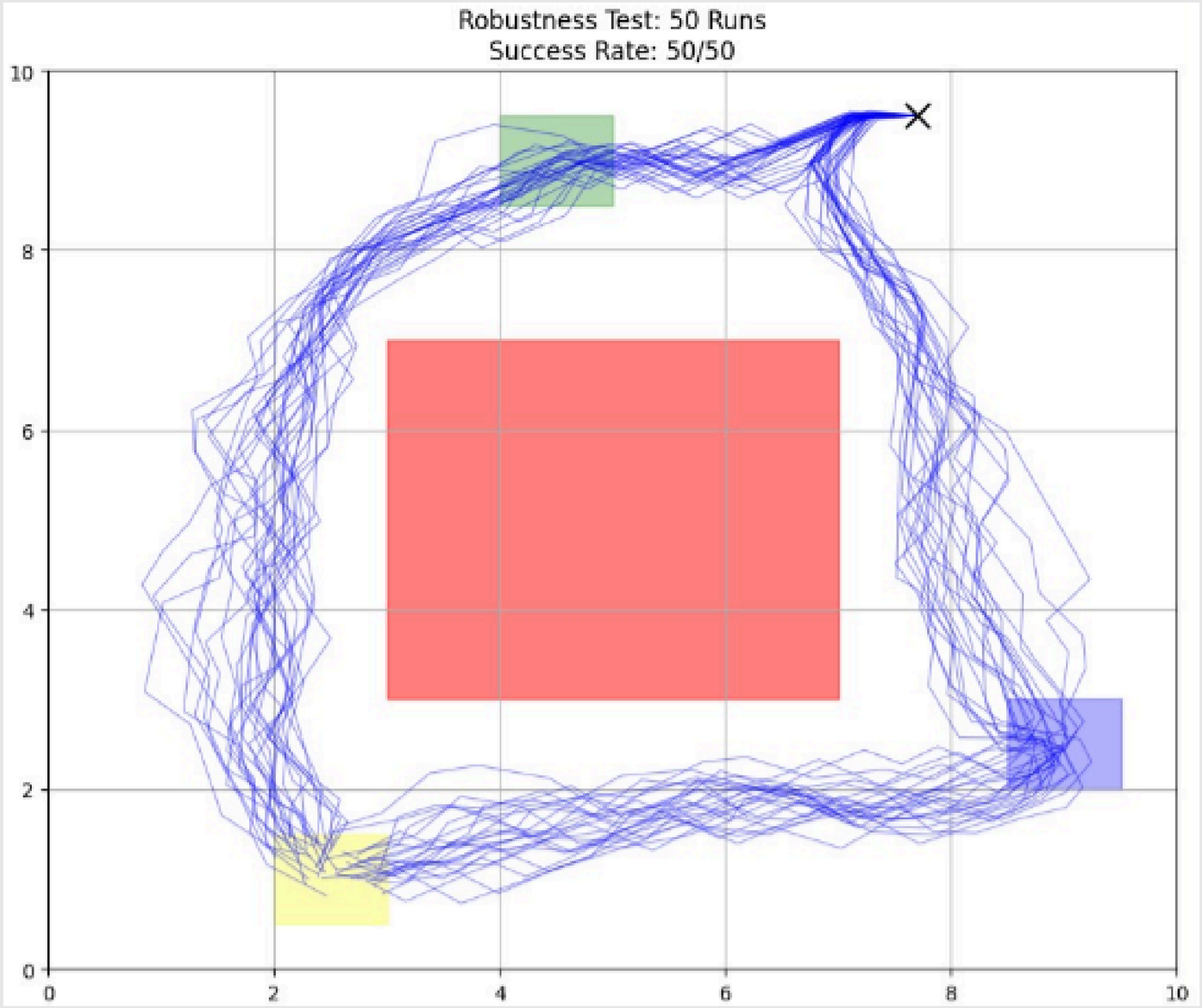
LET'S SEE HOW IT LOOKS LIKE IN 2D, USING RANDOM PERTURBATION




50 trajectories with random perturbations starting from (100, 100)
R1
R2
R3
R4
X
Y
0
2
4
6
8
10

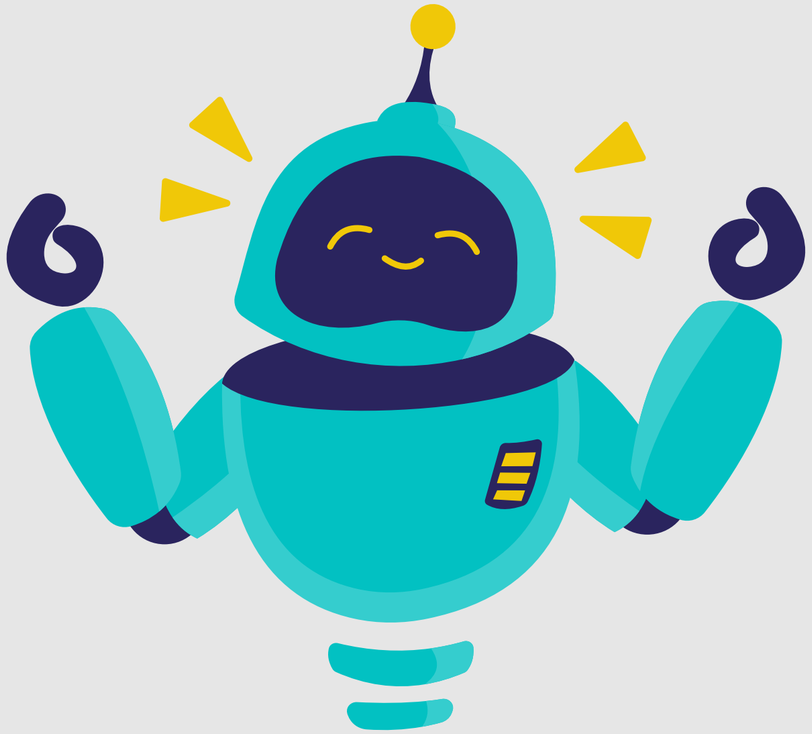



50 trajectories with random perturbations starting from (50, 0) using the 2nd specification (R5 -> R6 -> R7 -> R5)
R6
R8
R7
R5
10
8
6
4
2
0
Y
0
2
4
6
8
10
X

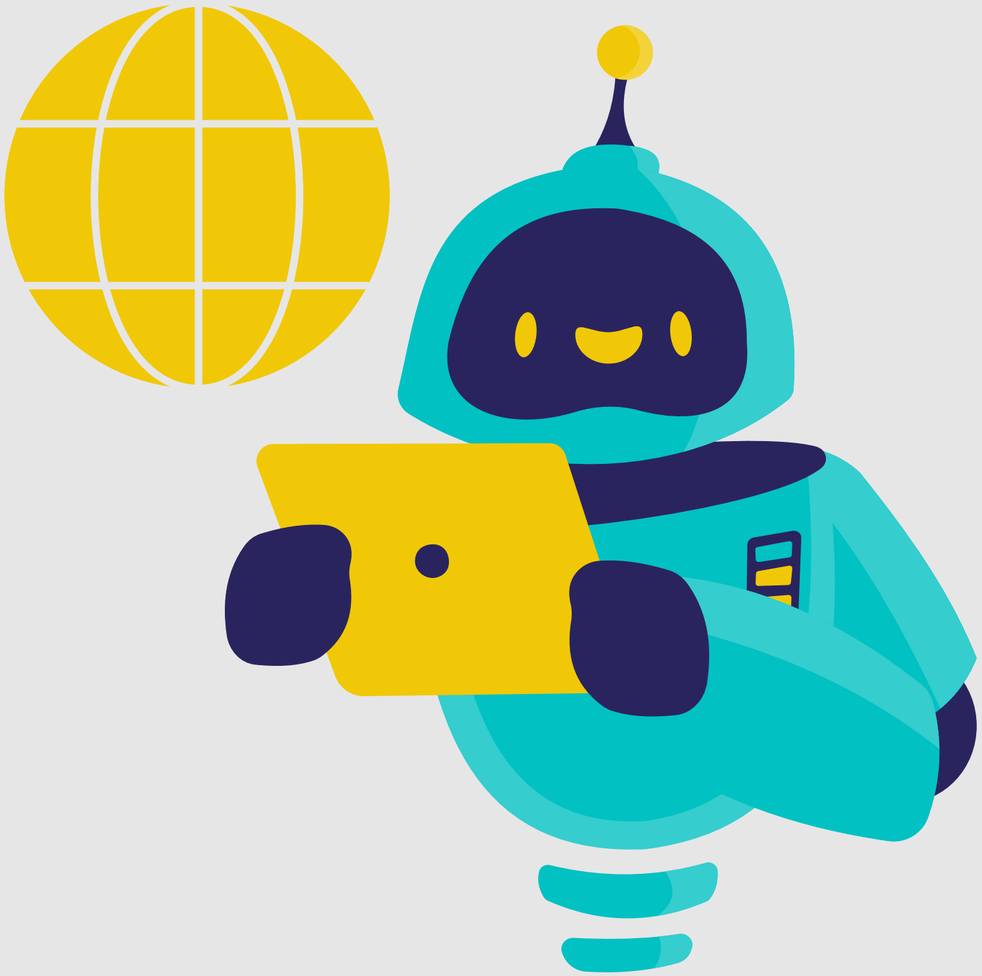

Robustness Test: 50 Runs
Success Rate: 50/50
10
8
6
4
2
0
0
2
4
6
8
10

PYBULLET SIMULATION

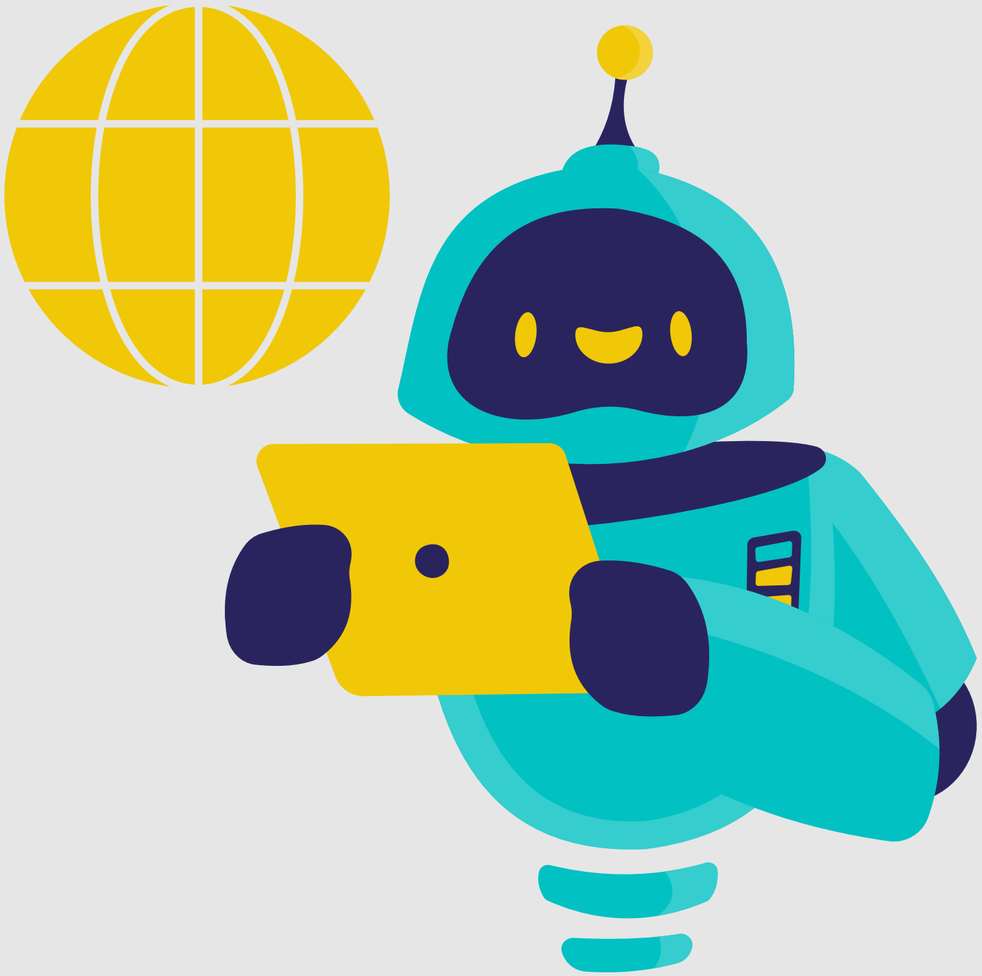
3D SIMULATION



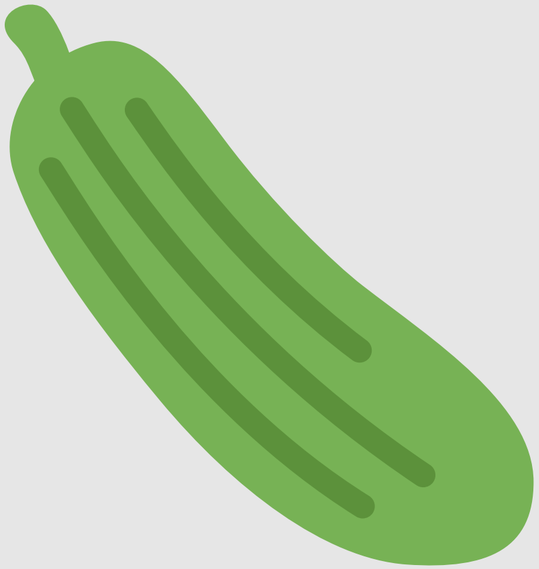
COMMUNICATION



COMMUNICATION



PICKLES



ANY QUESTIONS?