6-Axis Robot

5-Step-SARSA

26.01.2023

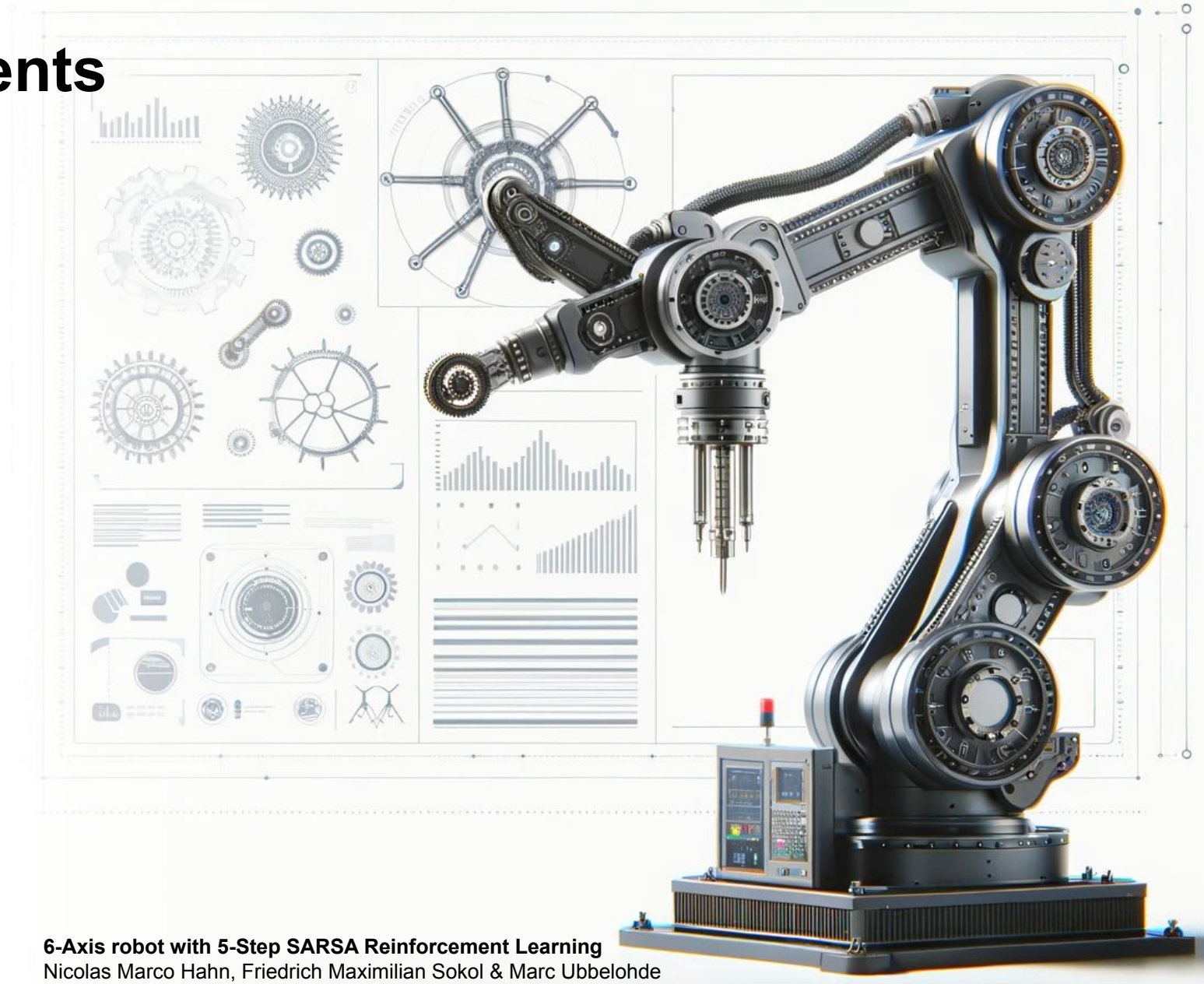
Friedrich Maximilian Sokol,
Nicolas Marco Hahn &
Marc Ubbelohde

BHT
Berliner Hochschule
für Technik

Studiere Zukunft

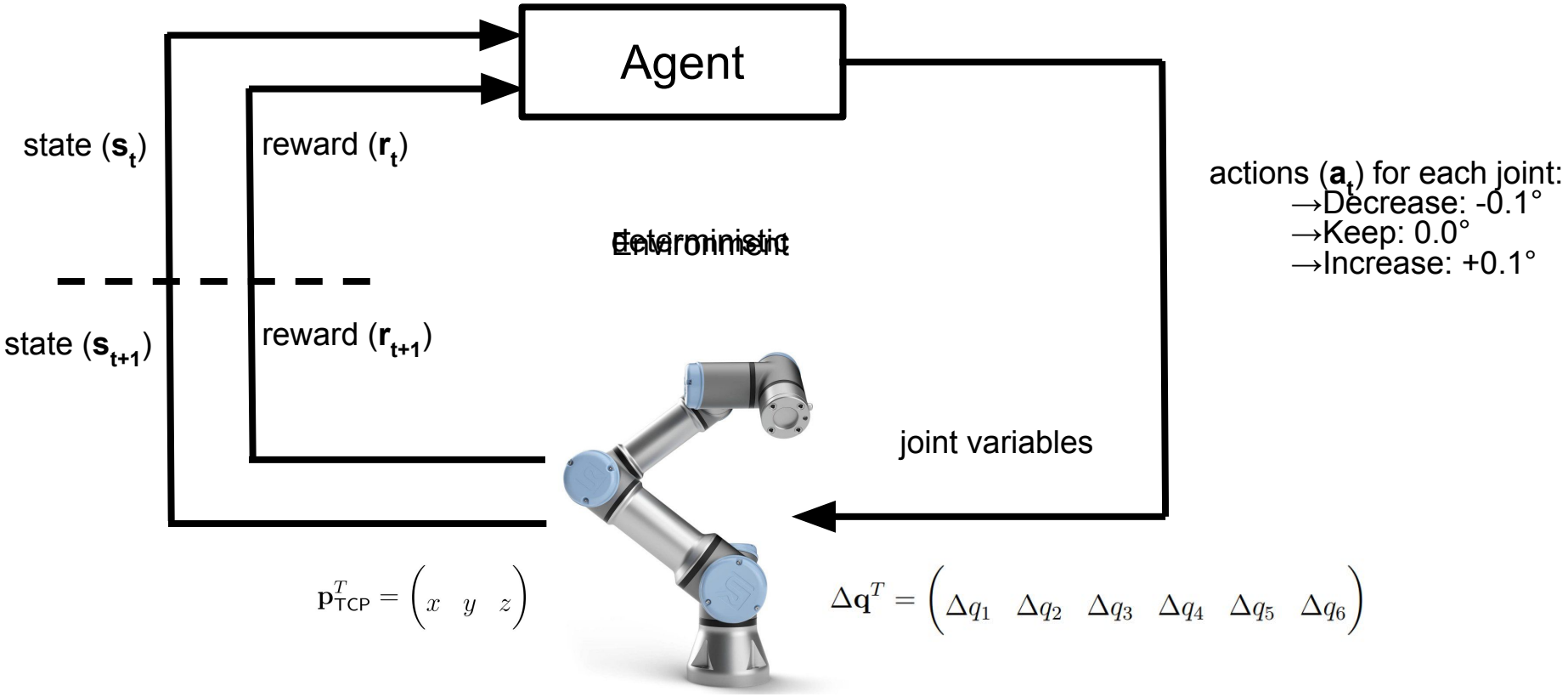
Table of Contents

- Task
- Environment
- Trajectory
- Algorithm
- Approaches
- Results
- Future Work



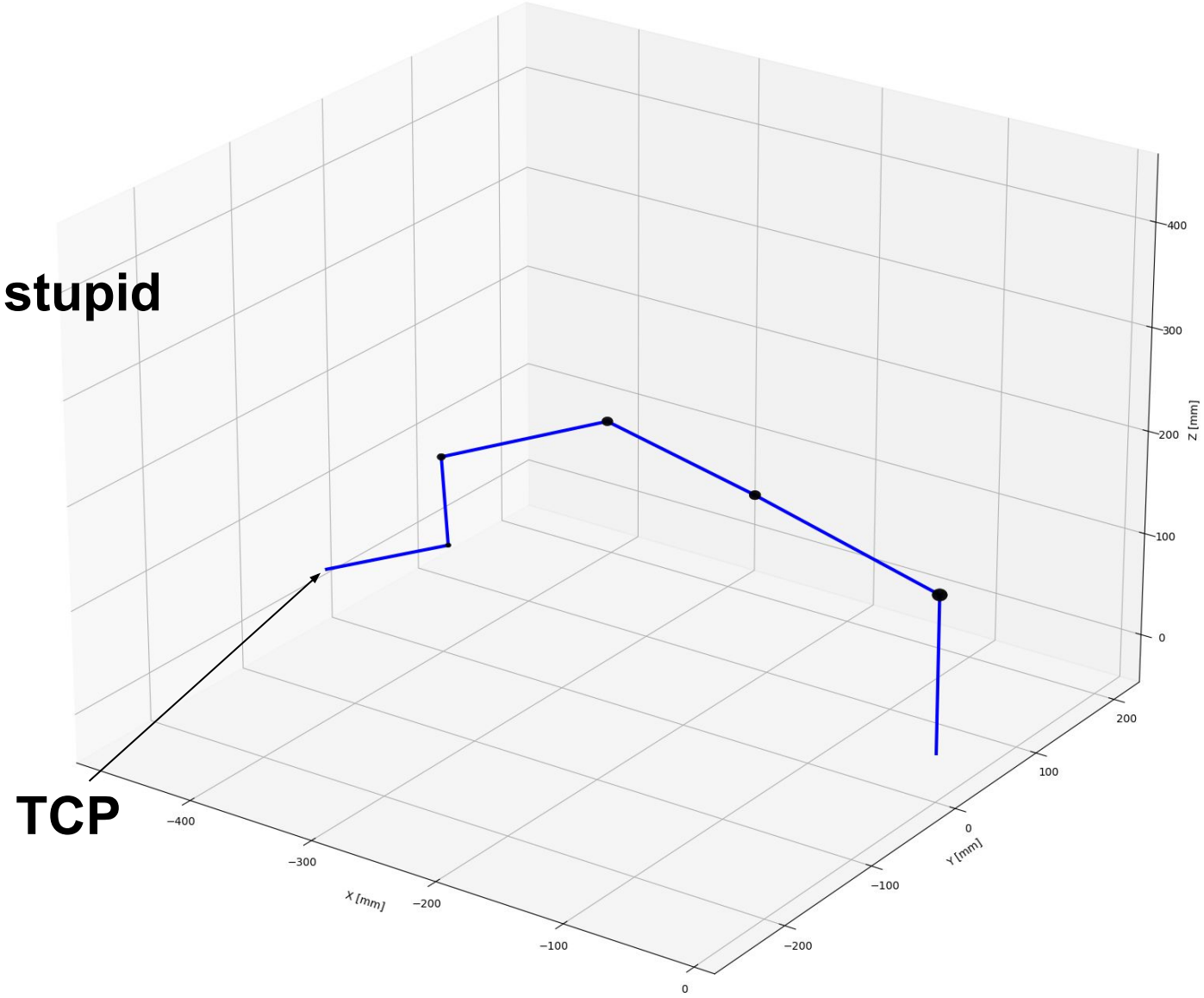
Source: Dall-E

Task


Agent
state ($\mathbf{s}_t$)
reward ($\mathbf{r}_t$)
Environment
deterministic
state ($\mathbf{s}_{t+1}$)
reward ($\mathbf{r}_{t+1}$)
actions ($\mathbf{a}_t$) for each joint:
→Decrease: -0.1°
→Keep: 0.0°
→Increase: +0.1°
joint variables
$\mathbf{p}_{\mathrm{TCP}}^T = \begin{pmatrix} x & y & z \end{pmatrix}$
$\Delta\mathbf{q}^T = \begin{pmatrix} \Delta q_1 & \Delta q_2 & \Delta q_3 & \Delta q_4 & \Delta q_5 & \Delta q_6 \end{pmatrix}$

Environment

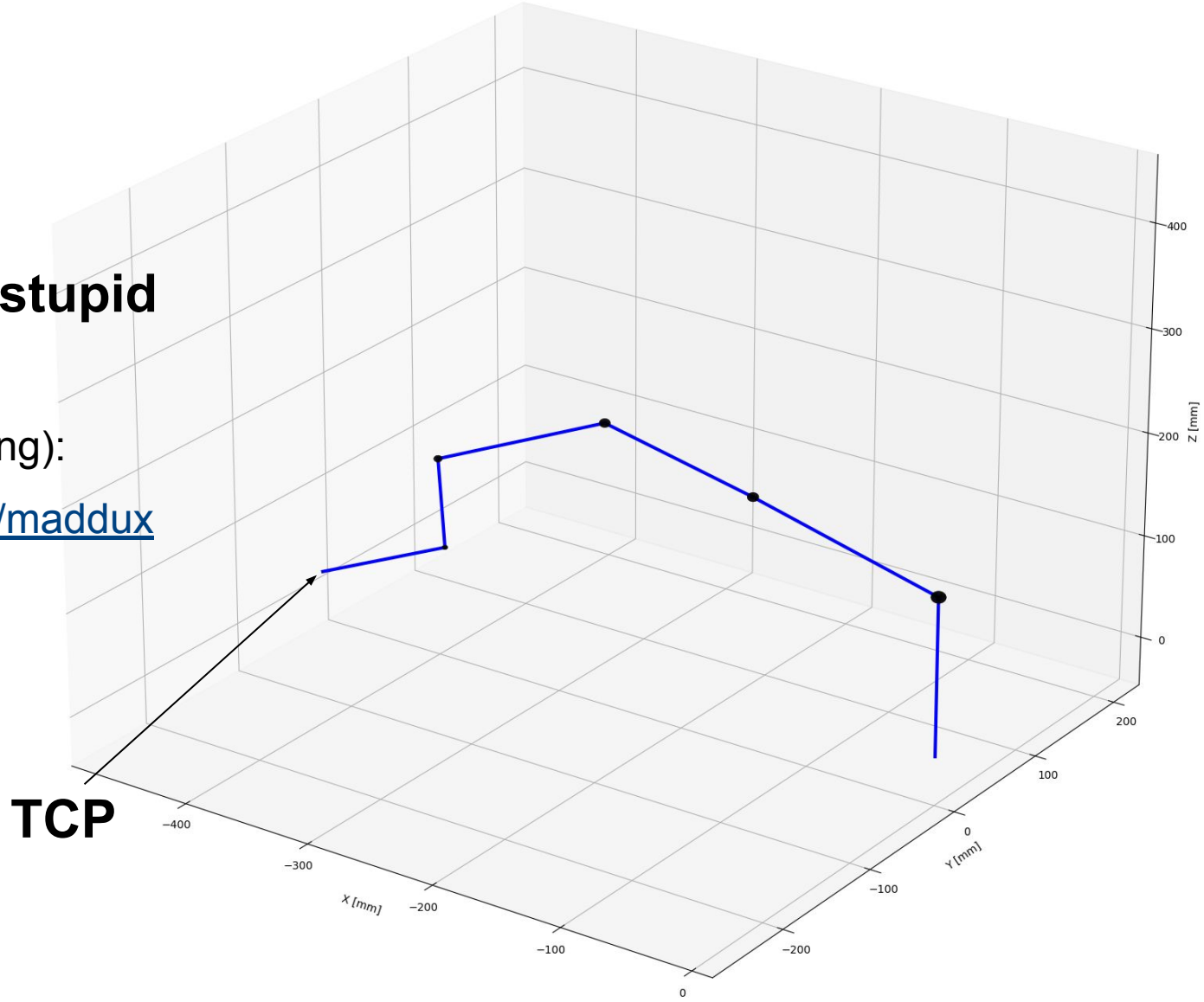
KISS-Principle: Keep it simple (and) stupid



Environment

KISS-Principle: Keep it simple (and) stupid

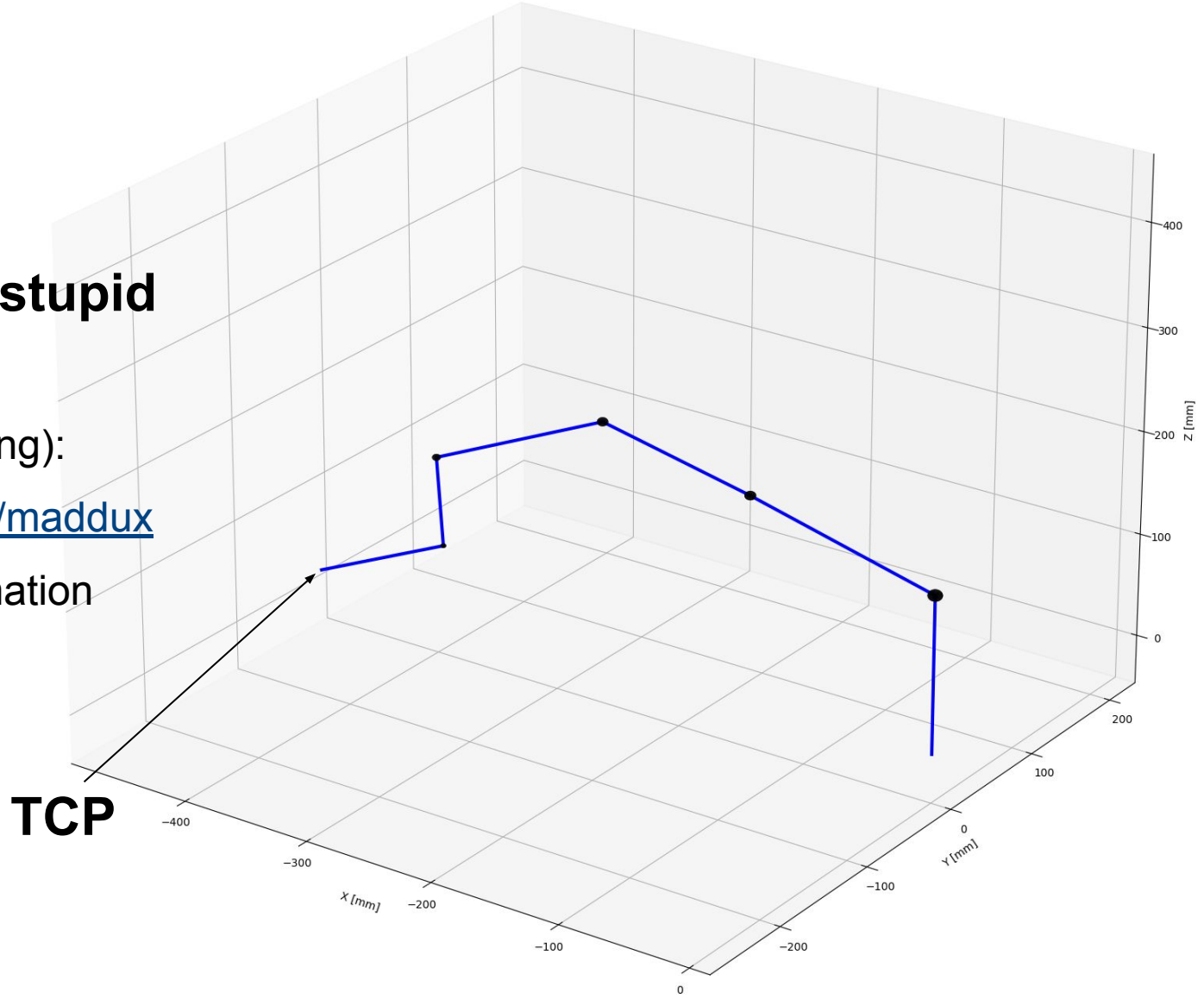
- Robot simulation (forward kinematics and plotting):
  - adapted version of https://github.com/bcaine/maddux



Environment

KISS-Principle: Keep it simple (and) stupid

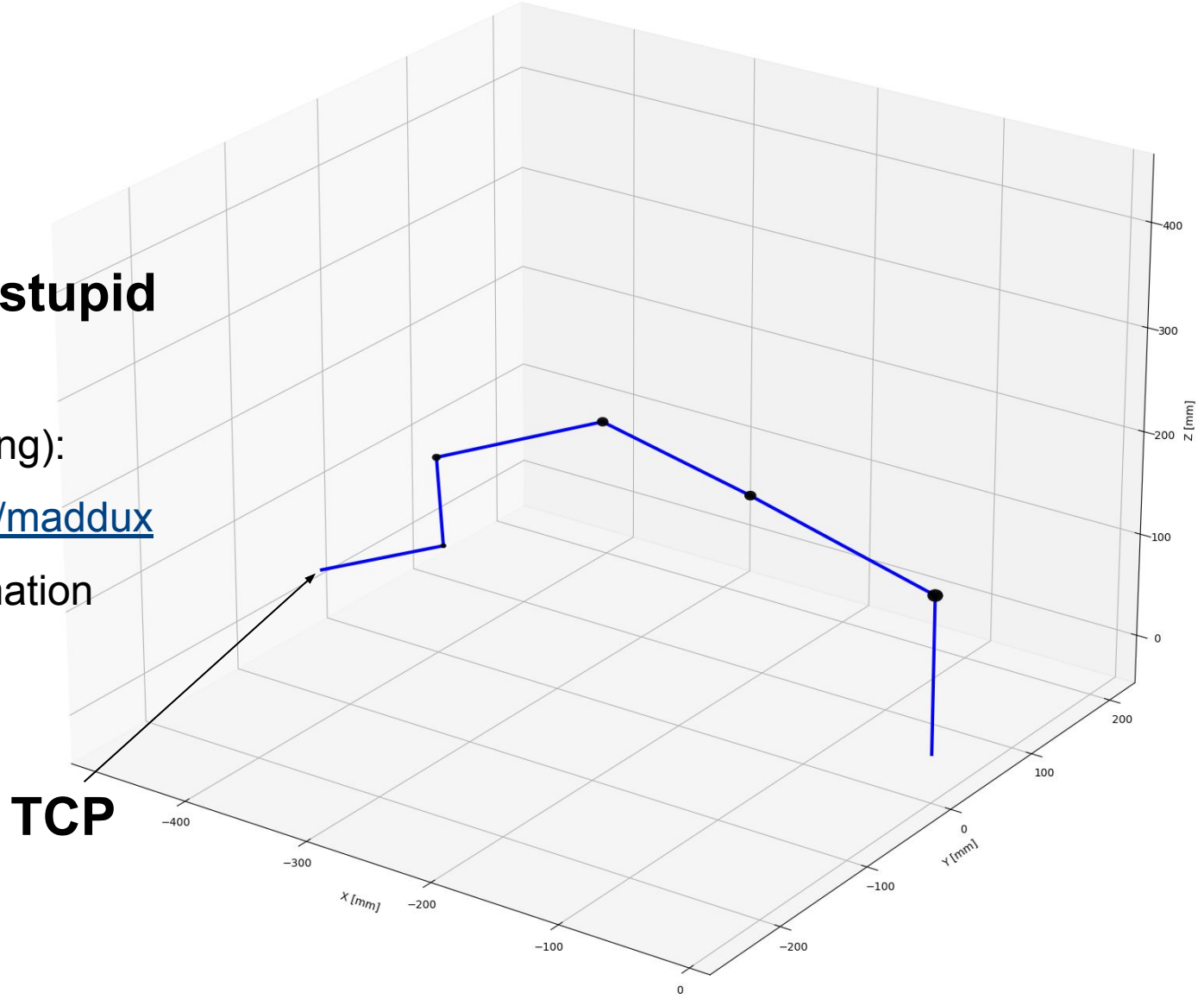
- Robot simulation (forward kinematics and plotting):
  - adapted version of https://github.com/bcaine/maddux
- Use matplotlib for light weight plotting and animation



Environment

KISS-Principle: Keep it simple (and) stupid

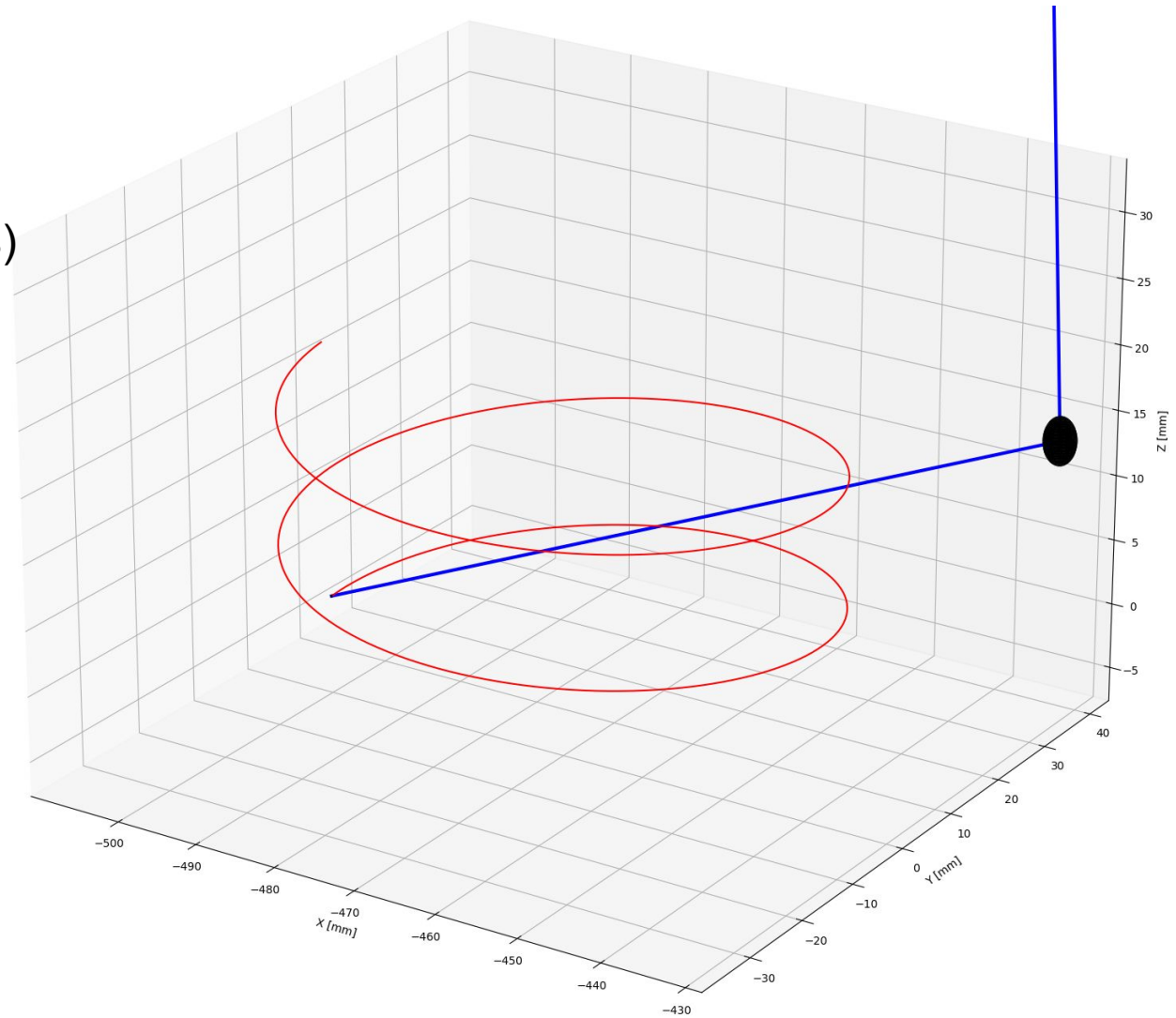
- Robot simulation (forward kinematics and plotting):
  - adapted version of https://github.com/bcaine/maddux
- Use matplotlib for light weight plotting and animation
- Wrapper around robot simulation:
  - get reward, get TCP
  - do move, check win etc.



Trajectory

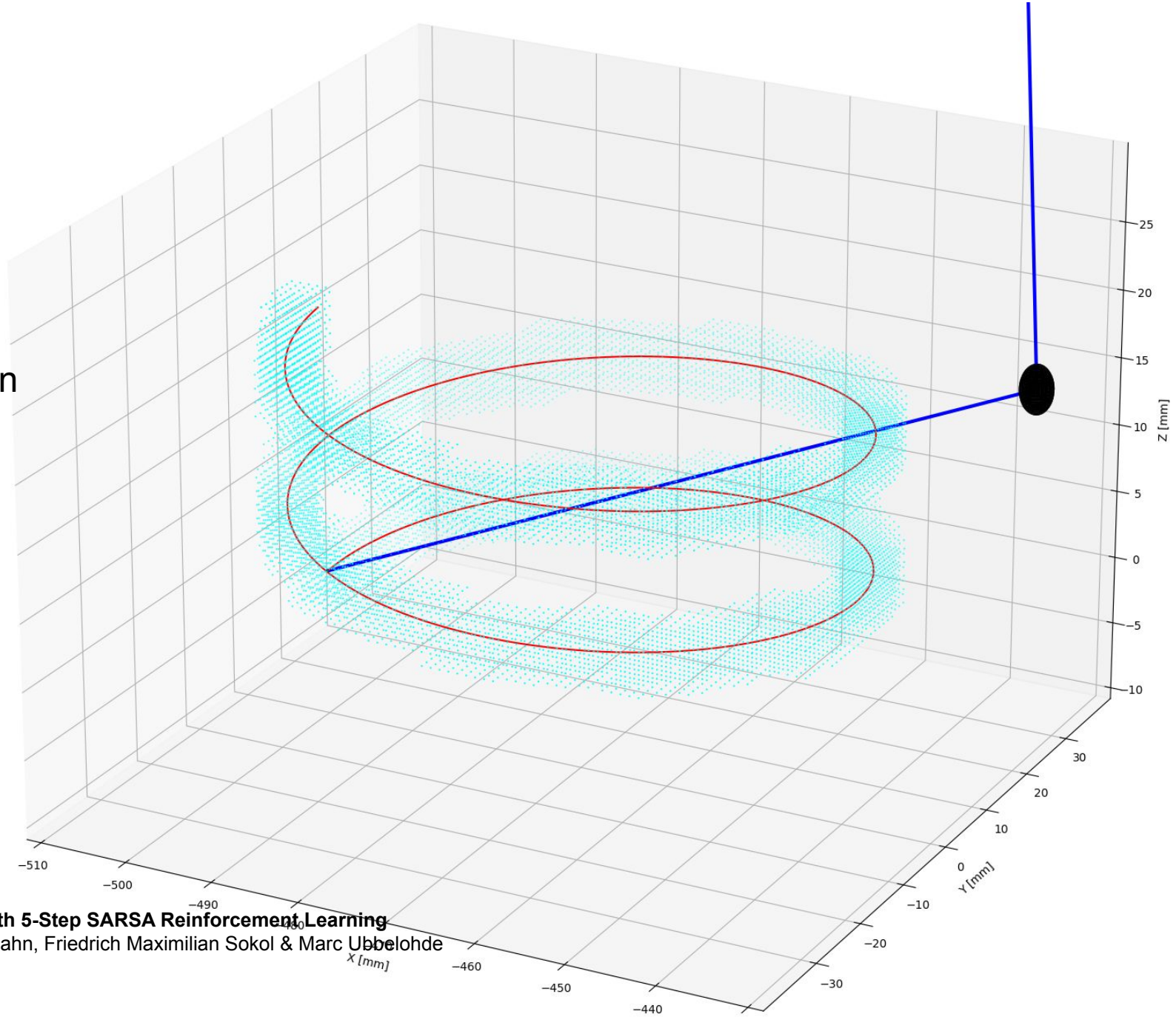
- Helix with radius 3 cm and a height of 2 cm (2 turns)
- Generated as a set of 300 x, y, z coordinates

$$\begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix} = \begin{pmatrix} -3 \cdot \cos(4\pi t) + 3 + x_{\text{start}} \\ 3 \cdot \sin(4\pi t) + y_{\text{start}} \\ 2 \cdot t + z_{\text{start}} \end{pmatrix}$$



Trajectory

- Voxellated space needed:
  - Q values
  - Reward specific to voxel position
  - “Winning Voxels”



Trajectory

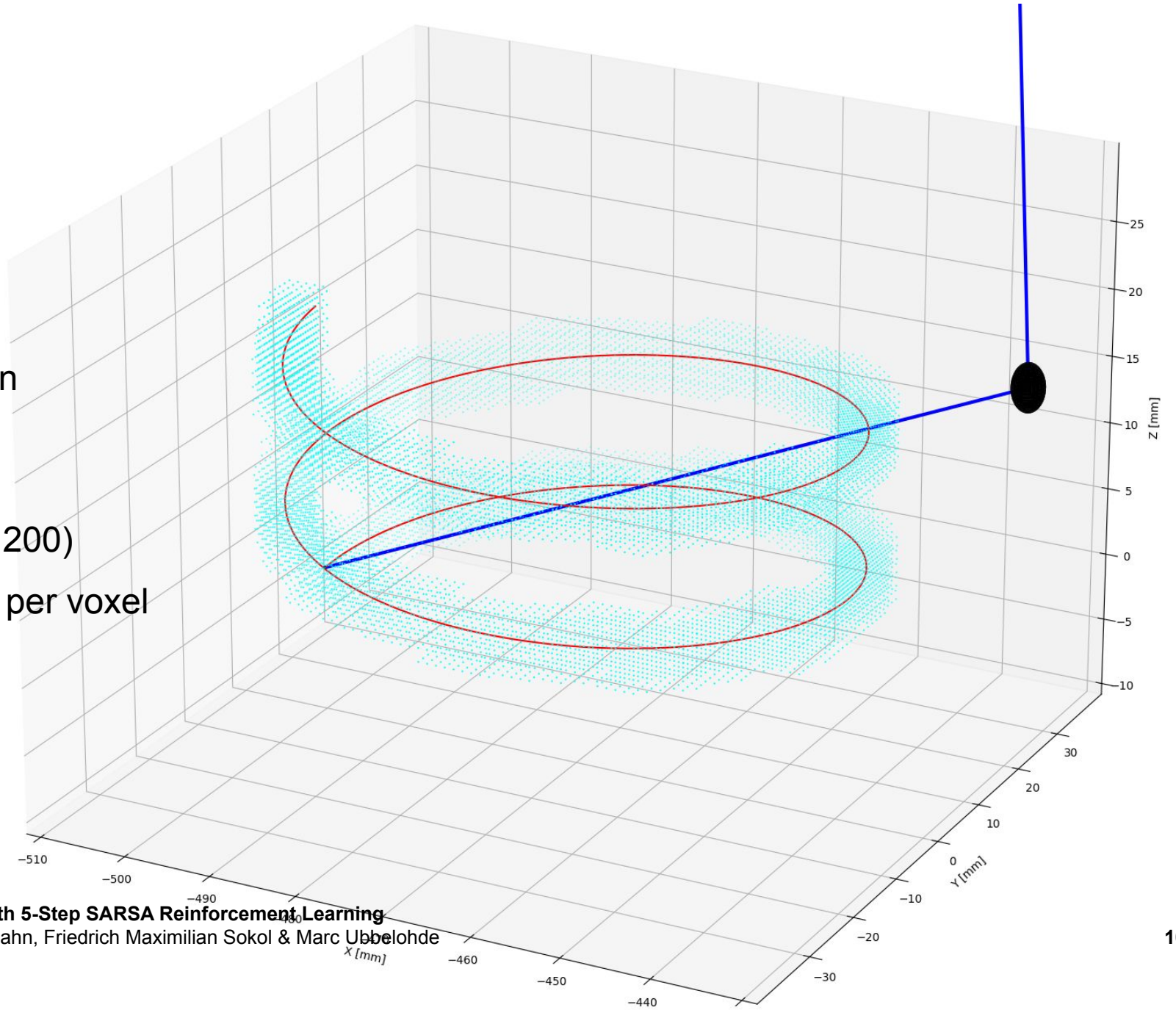
- Voxellated space needed:
  - Q values
  - Reward specific to voxel position
  - “Winning Voxels”

- Cube of voxels? (e.g. 600 x 600 x 200)

  = 72 × 10^6 voxels × 729 actions per voxel

  = 52.488 × 10^9 fields × 4

  = 209.952 TB of Data



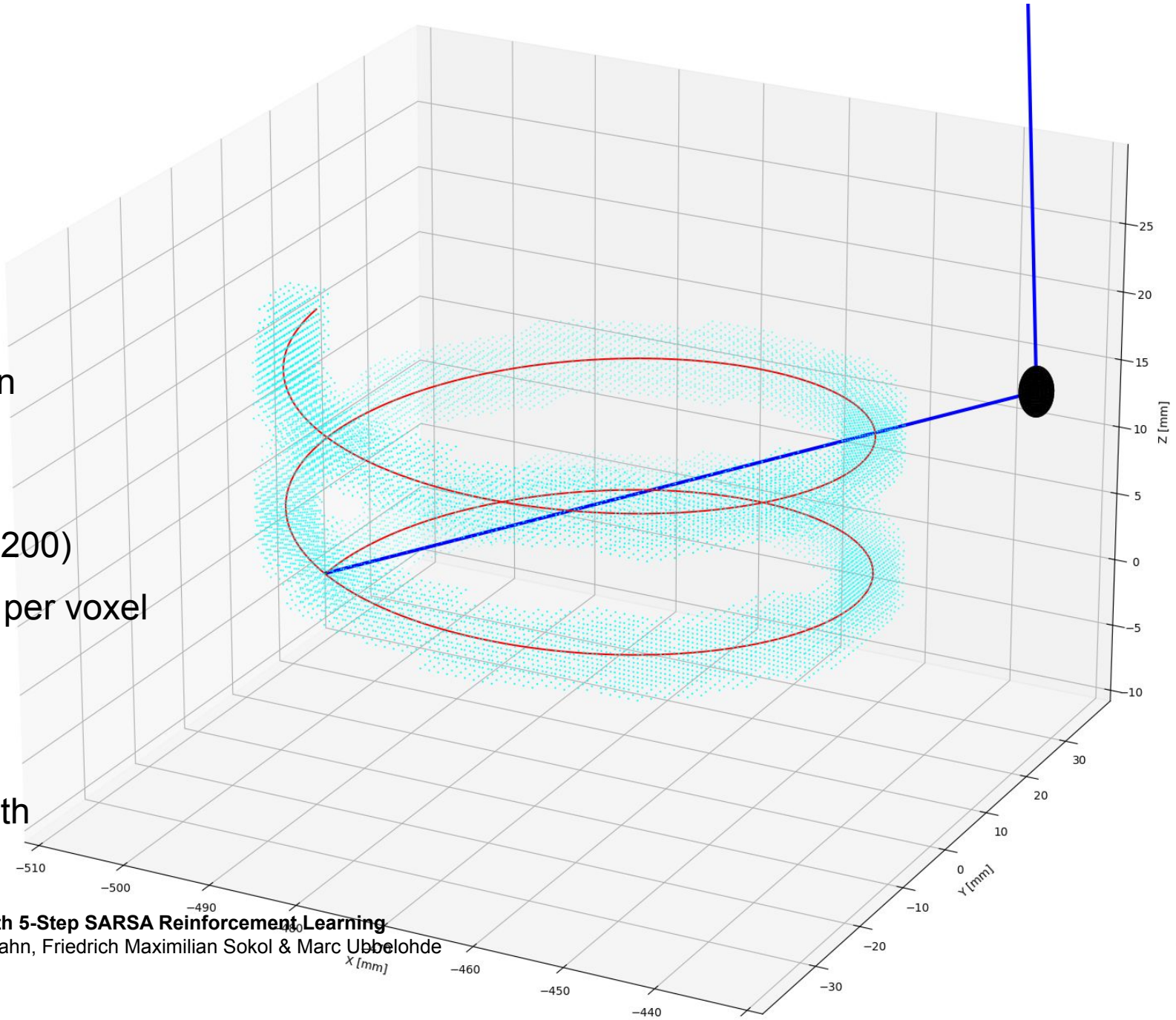
Trajectory

- Voxellated space needed:
  - Q values
  - Reward specific to voxel position
  - “Winning Voxels”
- Cube of voxels? (e.g. 600 x 600 x 200)

  = 72 × 10^6 voxels × 729 actions per voxel

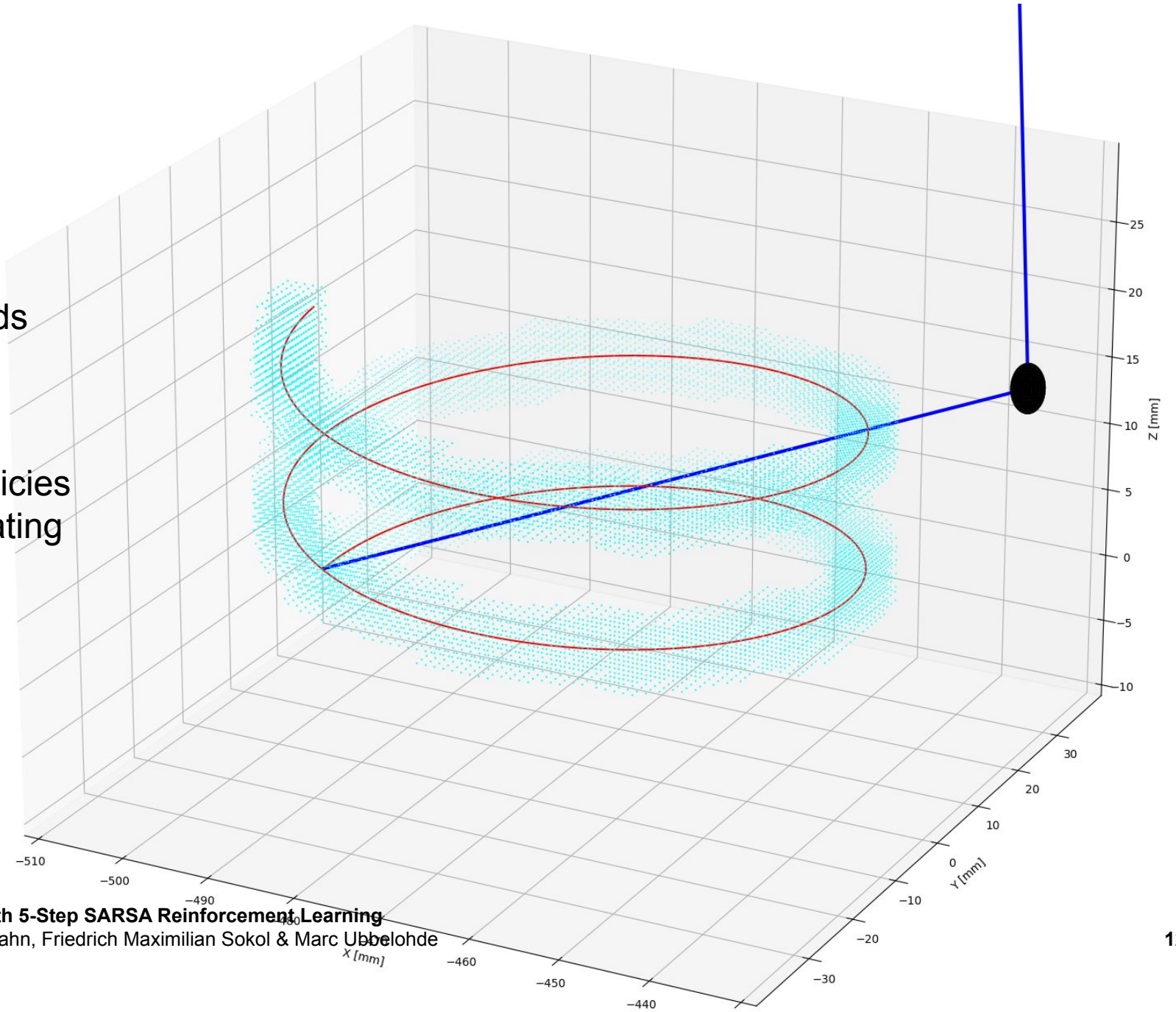
  = 52.488 × 10^9 fields × 4

  = 209.952 TB of Data
- →Only voxellate space around path



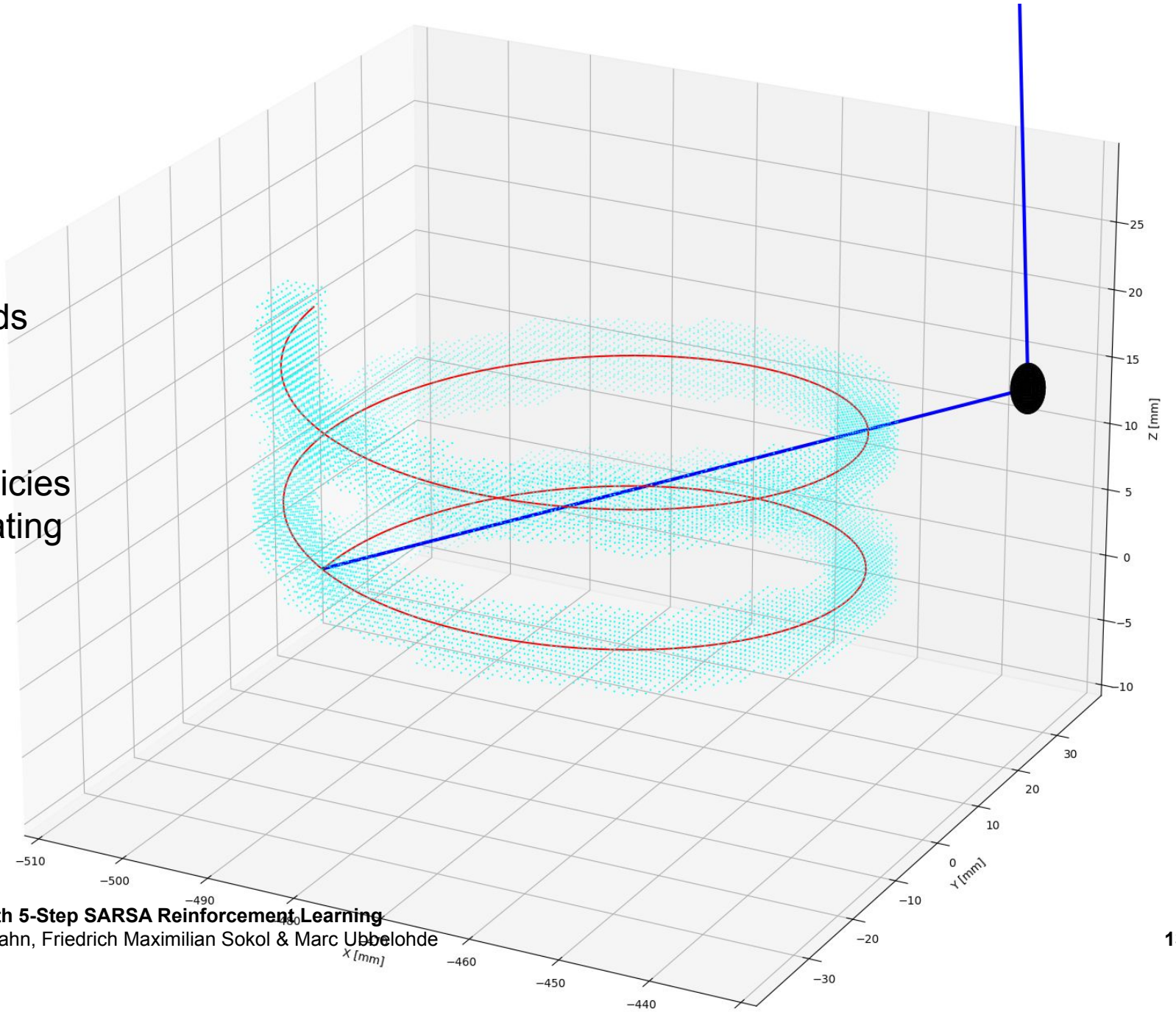
Trajectory

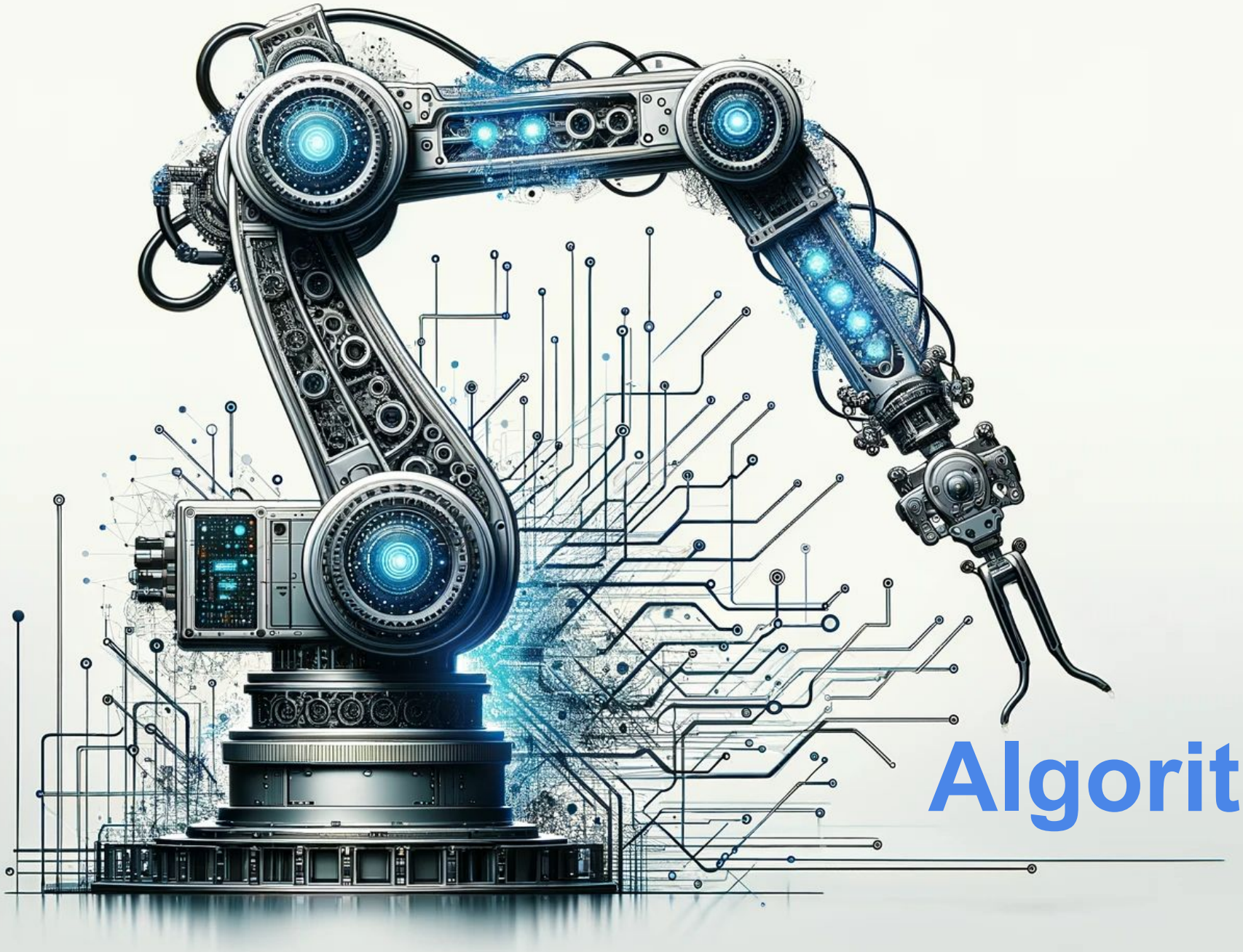
- Voxels generated before training
- Q values and voxel specific rewards in separate lists
- Voxel generates hash
  - used in hashmap containing indicies of each Q value / reward correlating to the Voxel



Trajectory

- Voxels generated before training
- Q values and voxel specific rewards in separate lists
- Voxel generates hash
  - used in hashmap containing indicies of each Q value / reward correlating to the Voxel
- No search algorithm needed
  - ⇒ Access time O(1)





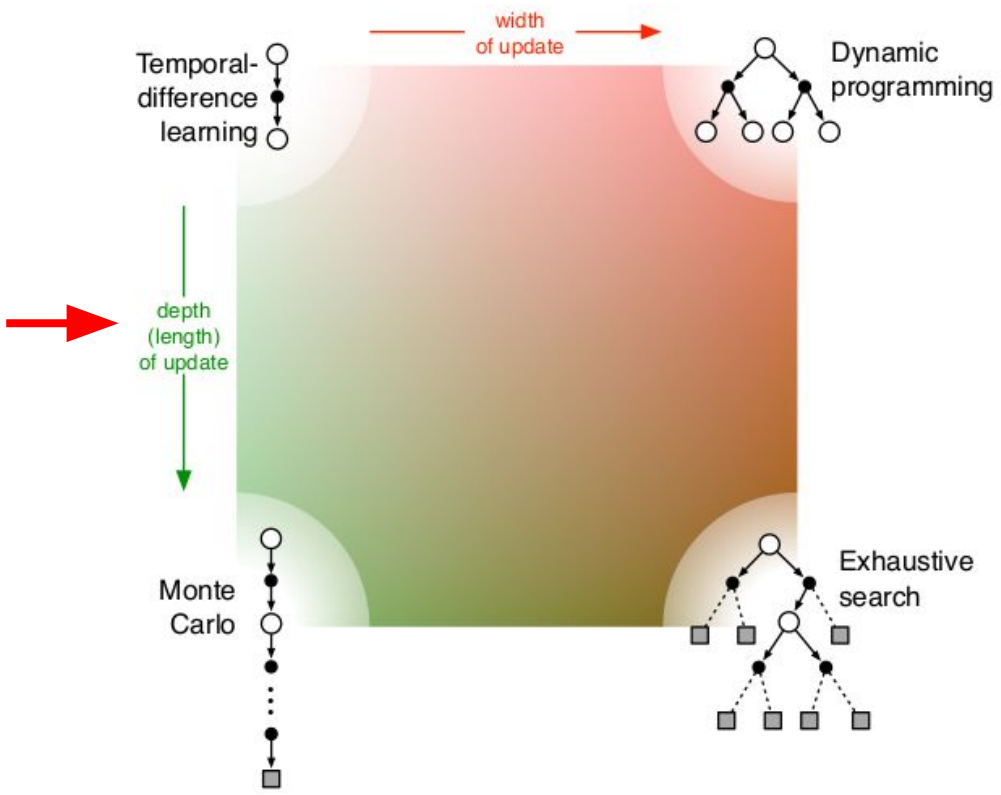
Algorithm

Source: Dall-E

Overview

n-step TD methods

- Different approaches to solve RL problems
- Graph have two axis which separate the algorithm
- Right: access to the model
- Left: looks at one trajectory, no model, sampling
- Upper: one step into the future
- Lower: until the end of the episode
- N-step-bootstrapping

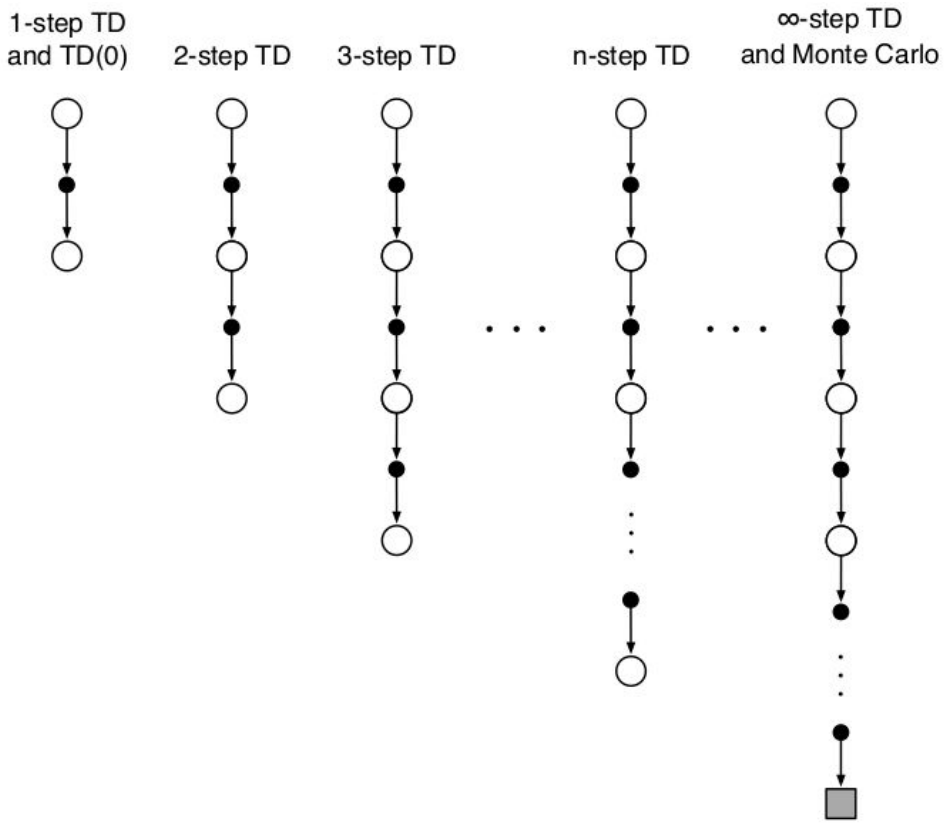


[1] Richard S. Sutton and Andrew g. Barto: Reinforcement Learning

Overview

n-step TD methods

- Two extremes: Temporal Difference and Monte Carlo
- Temporal Difference: information propagate slowly backwards
- Monte Carlo: updates are noisier
- Best methods are between: N-step-methods
- Bootstrapping works best with significant state change
- Look N-steps into the future
- Taking N-rewards into account (appropriately discounting)



[1] Richard S. Sutton and Andrew g. Barto: Reinforcement Learning

Implementation

5-Step-Sarsa

1) Input and output:

Transition-Probability-Function

Input: MDP $M = \langle S, s_0, A, P_a(s' \mid s), r(s, a, s') \rangle$, number of steps n

Output: Q-function Q

Initialise Q arbitrary; e.g., $Q(s, a) = 0$ for all s and a

Action-Value-Function

2) Iterate over episodes:

Repeat (for each episode)

agent acts greedy but we also need exploration

Select action a to apply in s using Q-values in Q and a multi-armed bandit algorithm such as ϵ-greedy

$ss = \langle s \rangle$

$as = \langle a \rangle$ store states, rewards and actions

$rs = \langle \rangle$

Implementation

5-Step-Sarsa

3) Interact with environment and store experience:

While ss is not empty

If s is not a terminal state then:

Execute action a in state s

Observe reward r and new state s'

$rs \leftarrow rs + \langle r \rangle$ add received reward to reward list

If s' is not a terminal state then:

Select action a' to apply in s' using Q and a multi-armed bandit algorithm

$ss \leftarrow ss + \langle s' \rangle$

$as \leftarrow ss + \langle a' \rangle$ add new state and actions to the lists

Implementation

5-Step-Sarsa

4) Learning from experience:

learning begins if reward list has a length of 5

If $|rs| = n$ or s is a terminal state then:

$G \leftarrow \sum_{i=0}^{|rs|-1} \gamma^i rs_i$

If s is not a terminal state then: $G \leftarrow G + \gamma^n Q(s', a')$

$Q(ss_0, as_0) \leftarrow Q(ss_0, as_0) + \alpha[G - Q(ss_0, as_0)]$ calculate the new action-value

$rs \leftarrow rs_{[1:n+1]}$

$ss \leftarrow ss_{[1:n+1]}$ delete everything which is older then 5 steps in the past

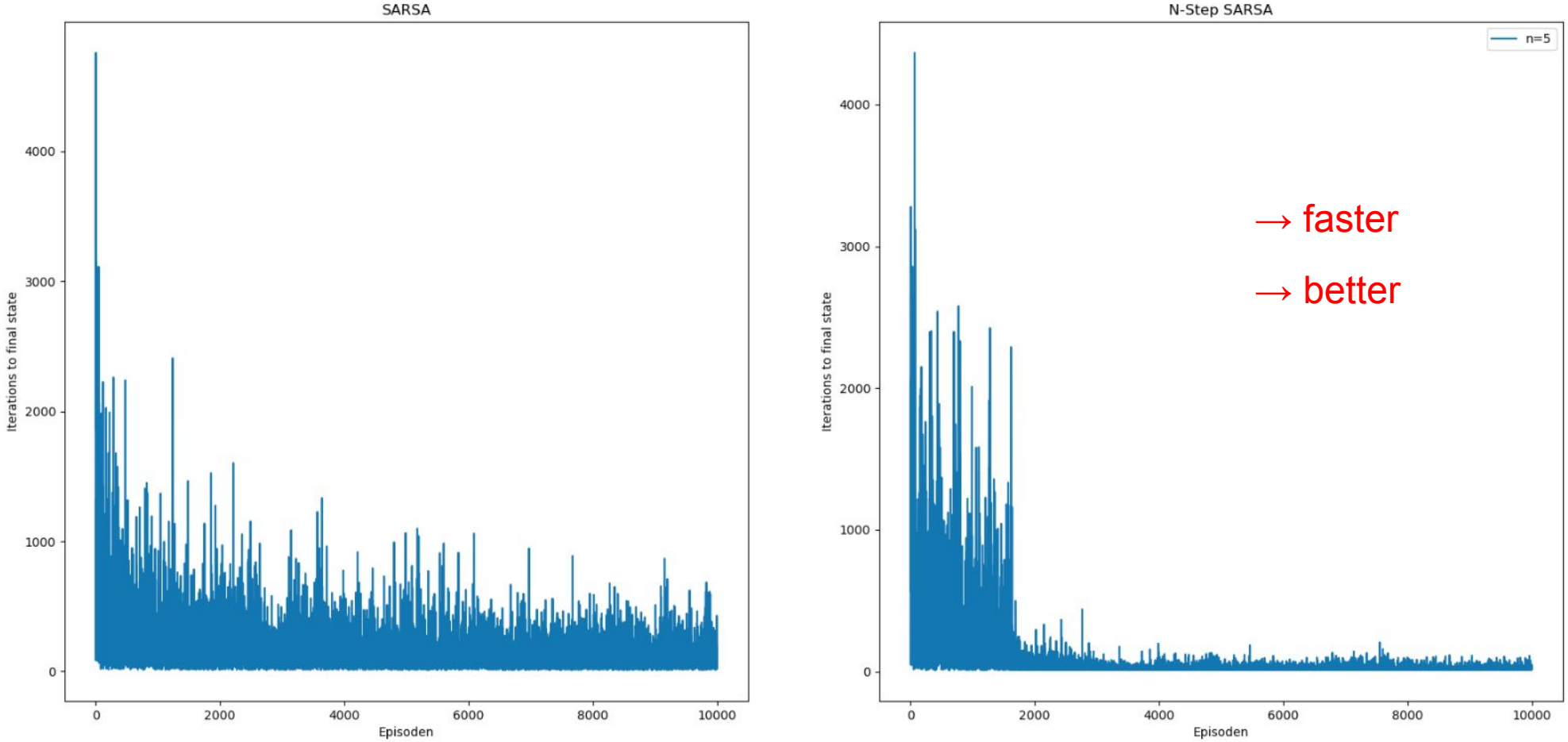
$as \leftarrow as_{[1:n+1]}$

$s \leftarrow s'$

$a \leftarrow a'$

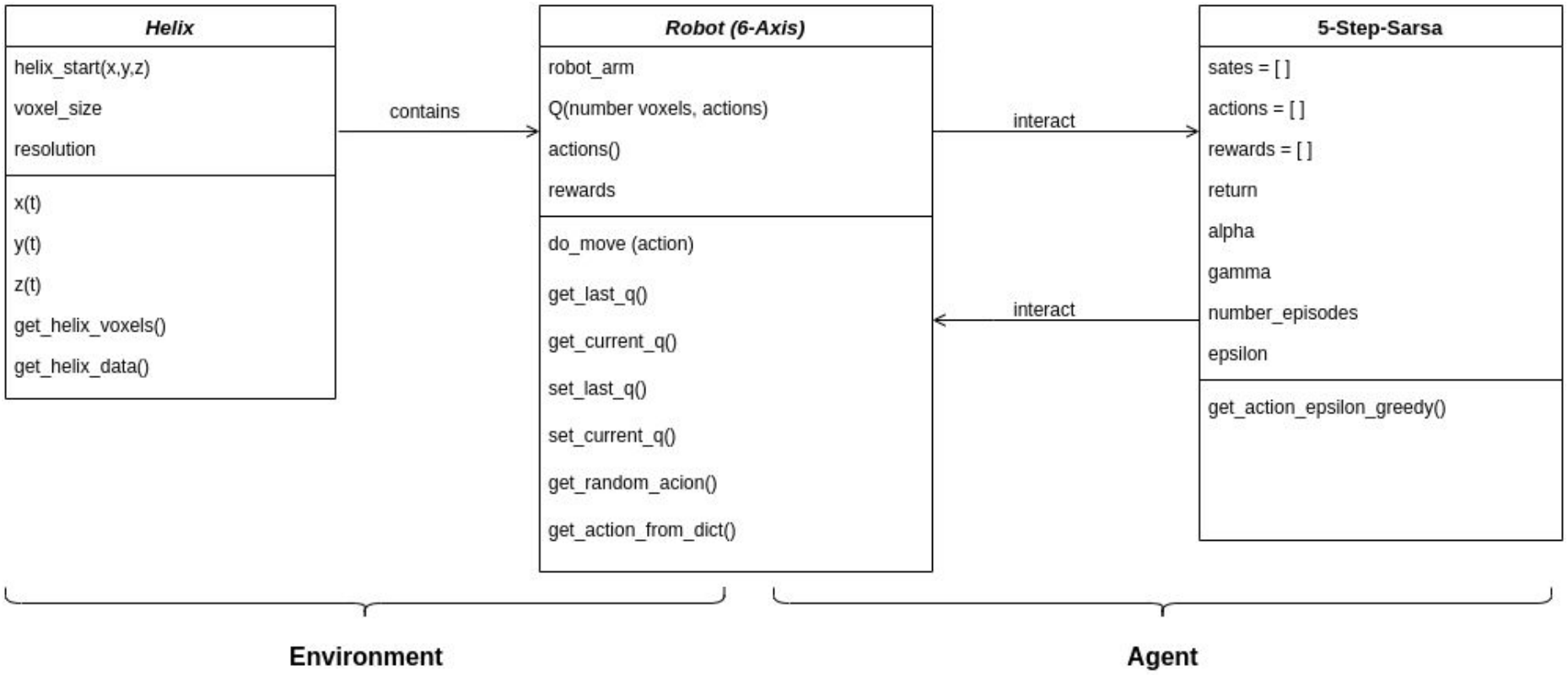
Validation

5-Step-Sarsa, race-track environment



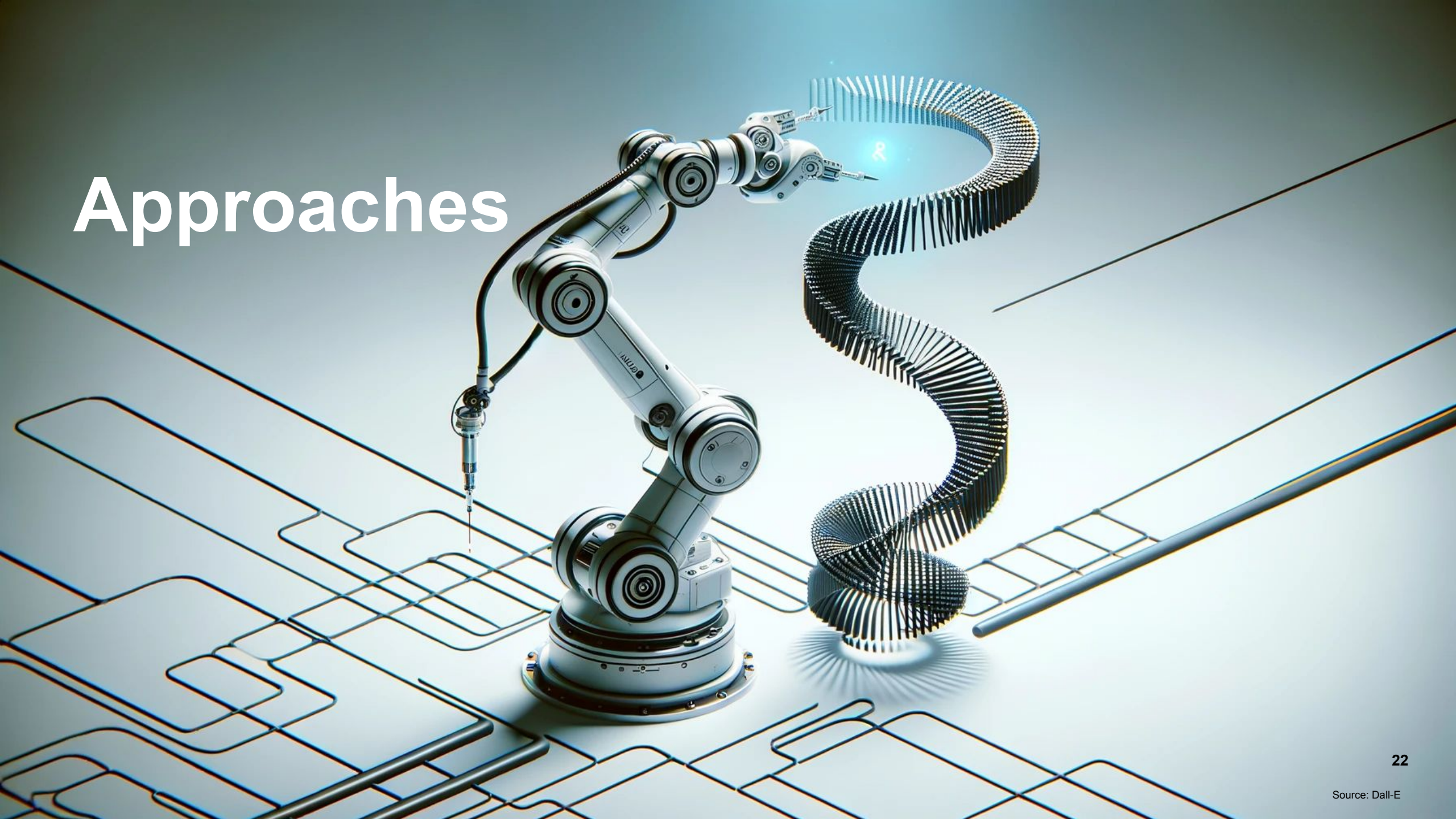
Implementation

Put all together, 6-Axis-Robot-Arm



* not all attributes and methods are shown

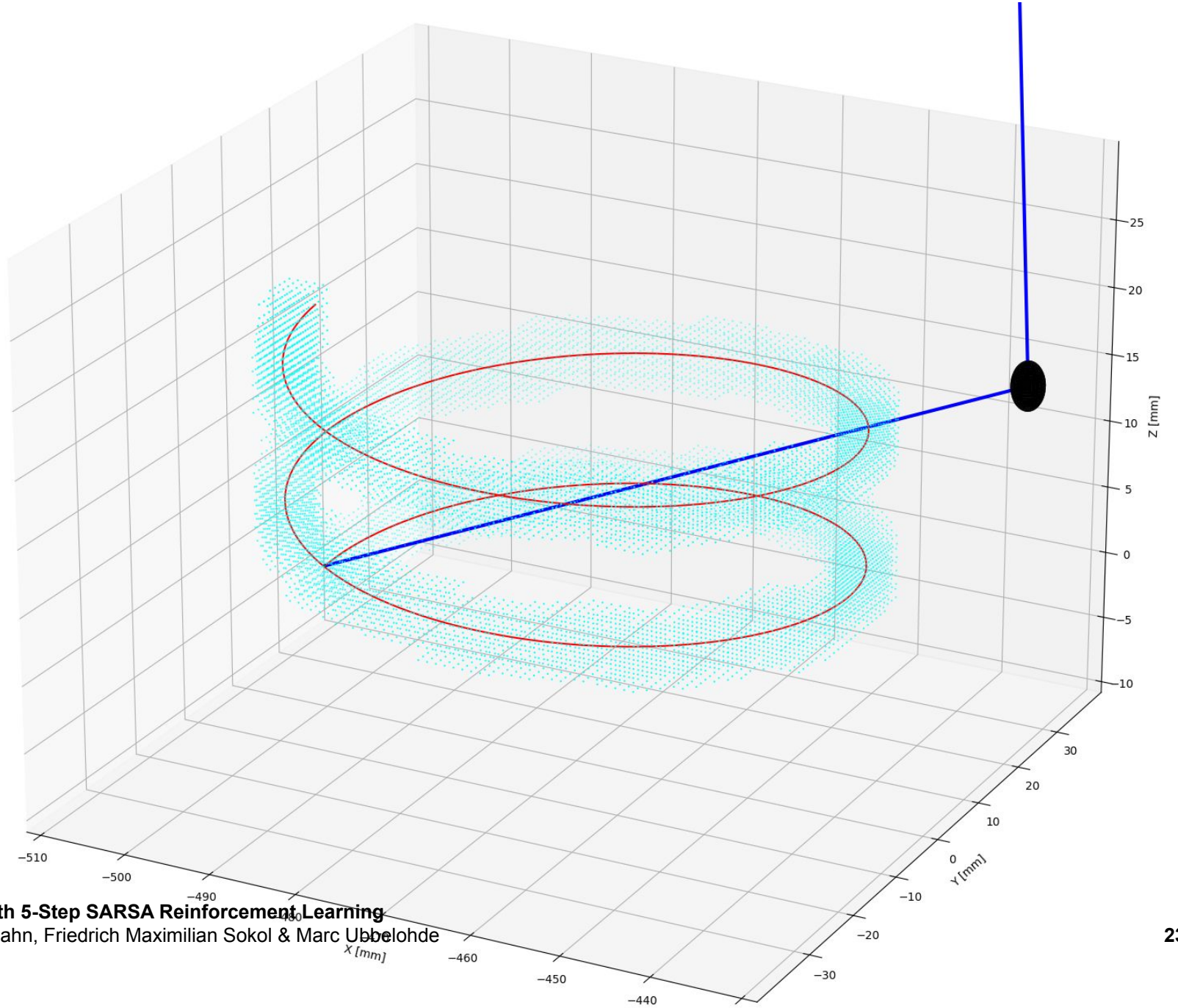
Approaches



Source: Dall-E

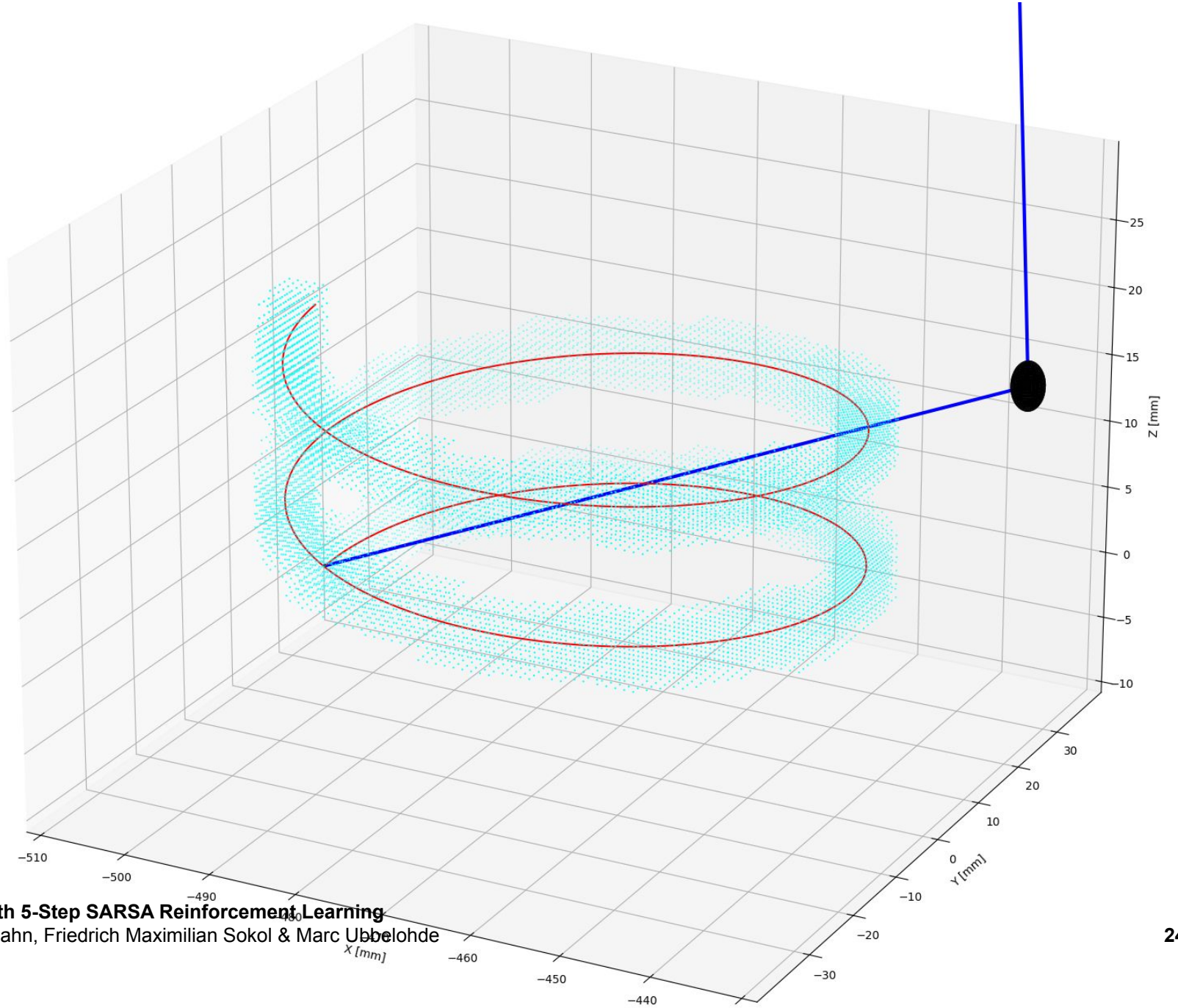
1st Approach

- Learn whole helix at once



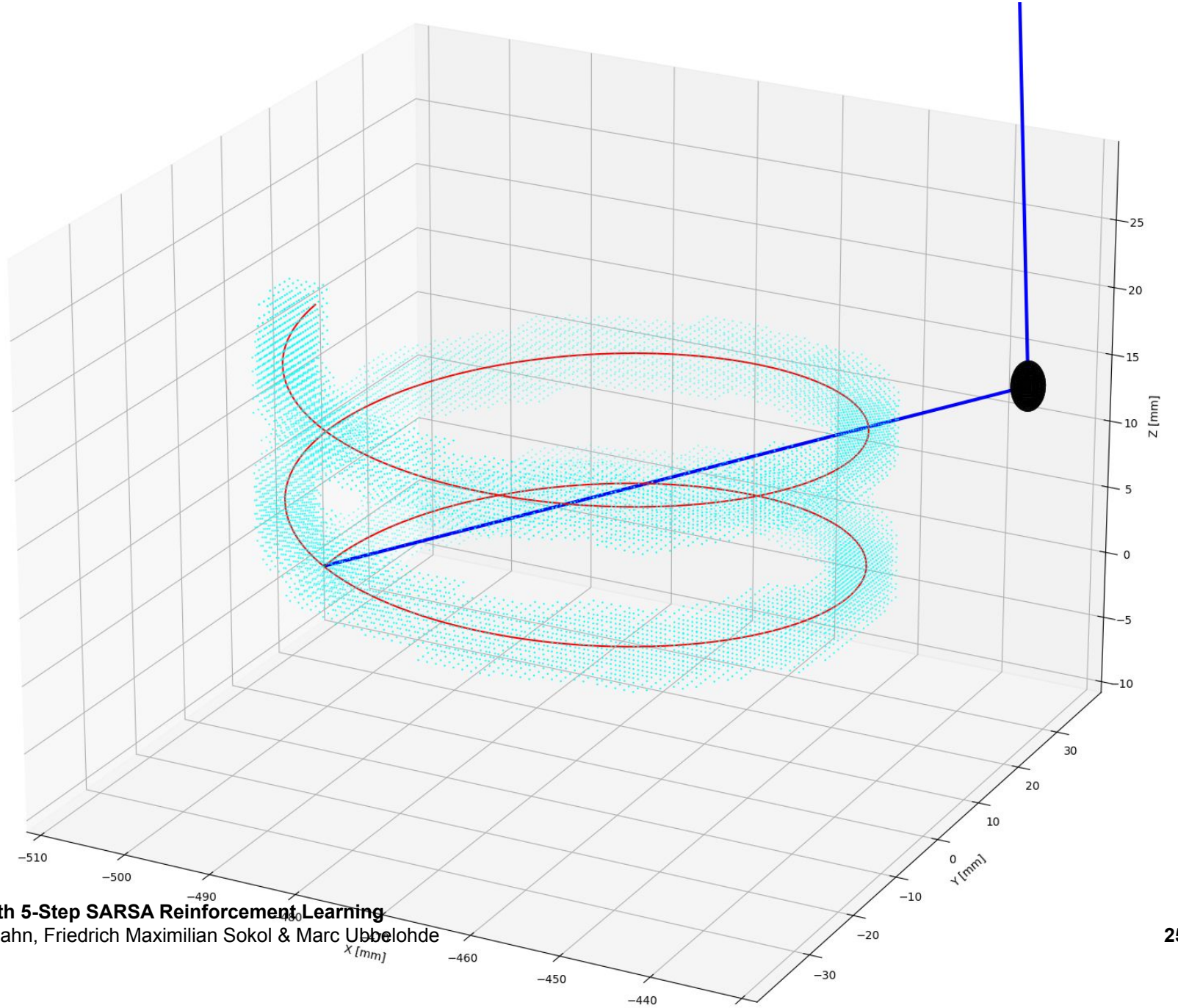
1st Approach

- Learn whole helix at once
- Reward system:
  - -1 every step
  - 0 when in finishing voxels



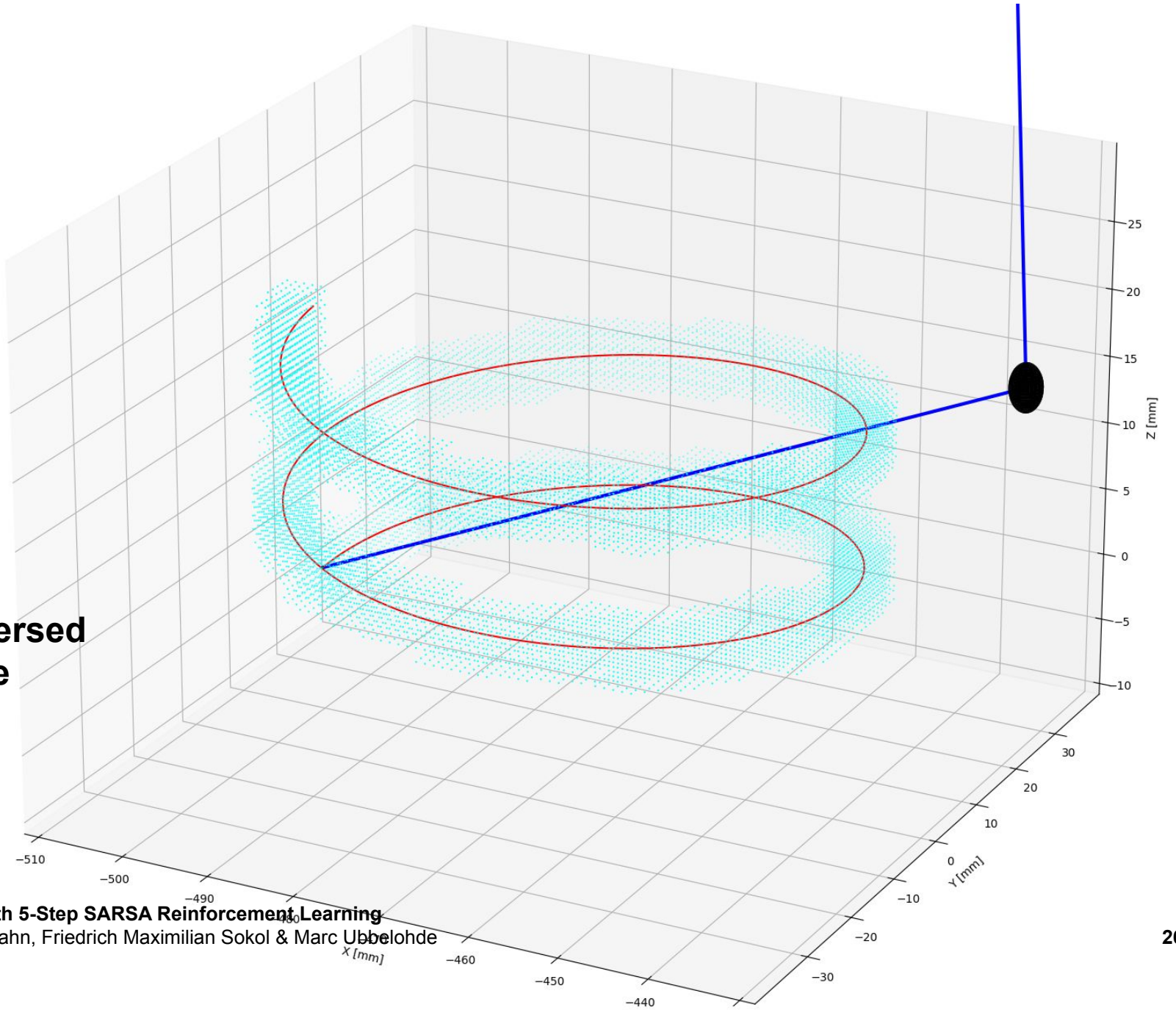
1st Approach

- Learn whole helix at once
- Reward system:
  - -1 every step
  - 0 when in finishing voxels
- … never even finished a single episode



1st Approach

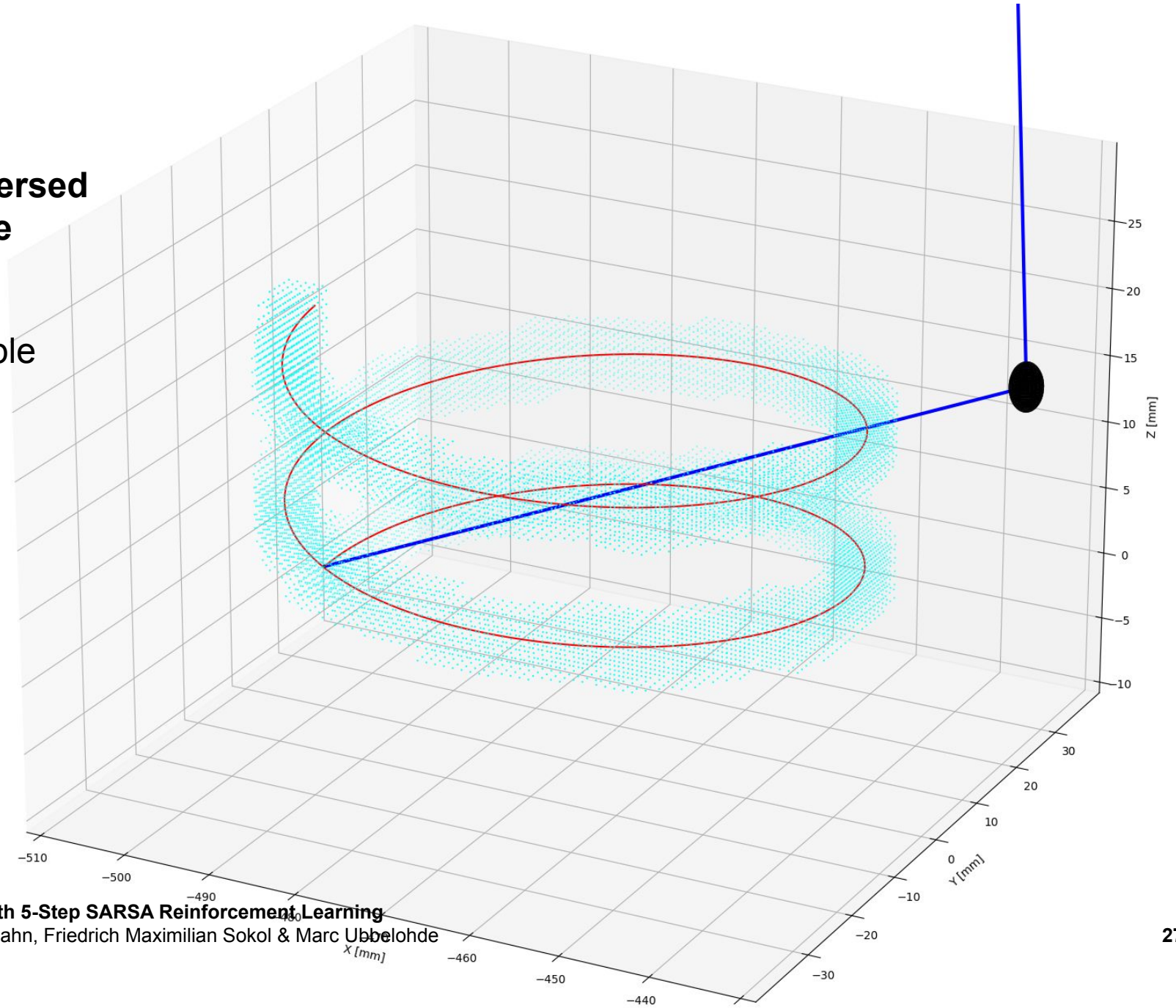
- Learn whole helix at once
- Reward system:
  - -1 every step
  - 0 when in finishing voxels
- … never even finished a single episode
- **Problem: Helix needs to be traversed by pure chance at least one time**



1st Approach

- **Problem: Helix needs to be traversed by pure chance at least one time**

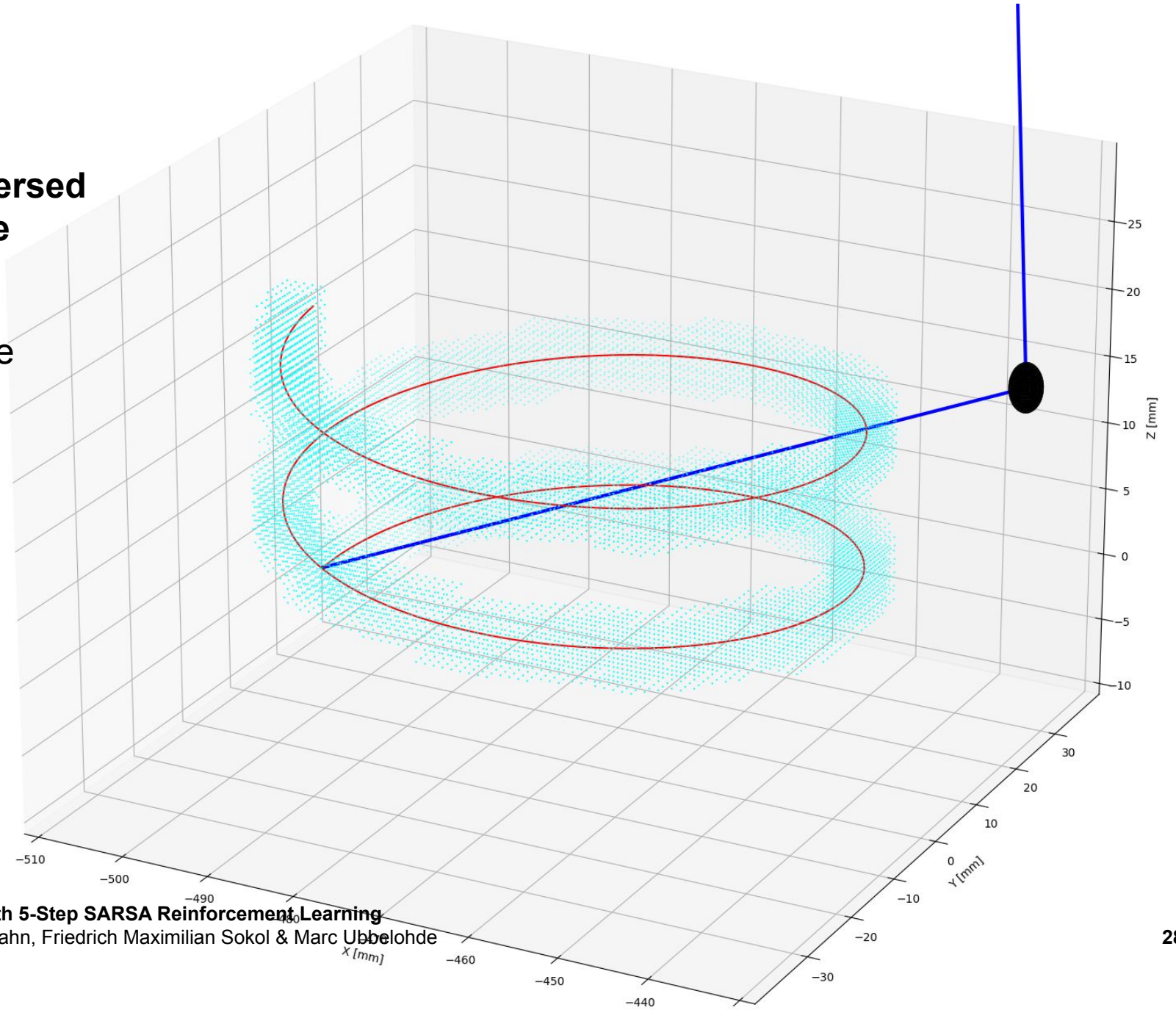
- Assumption: 100 out of 729 possible actions are good
- Probability to take good step: $\frac{100}{729}$



1st Approach

- **Problem: Helix needs to be traversed by pure chance at least one time**

- Assumption: 10 out of 729 possible actions are good
- Probability to take good step: $\frac{100}{729}$
- Assumption: >1000 steps needed to finish helix, max of 10^9 steps
- Probability to finish Helix: $\sum_{n=1000}^{1\cdot 10^9} \frac{100}{729}^n$

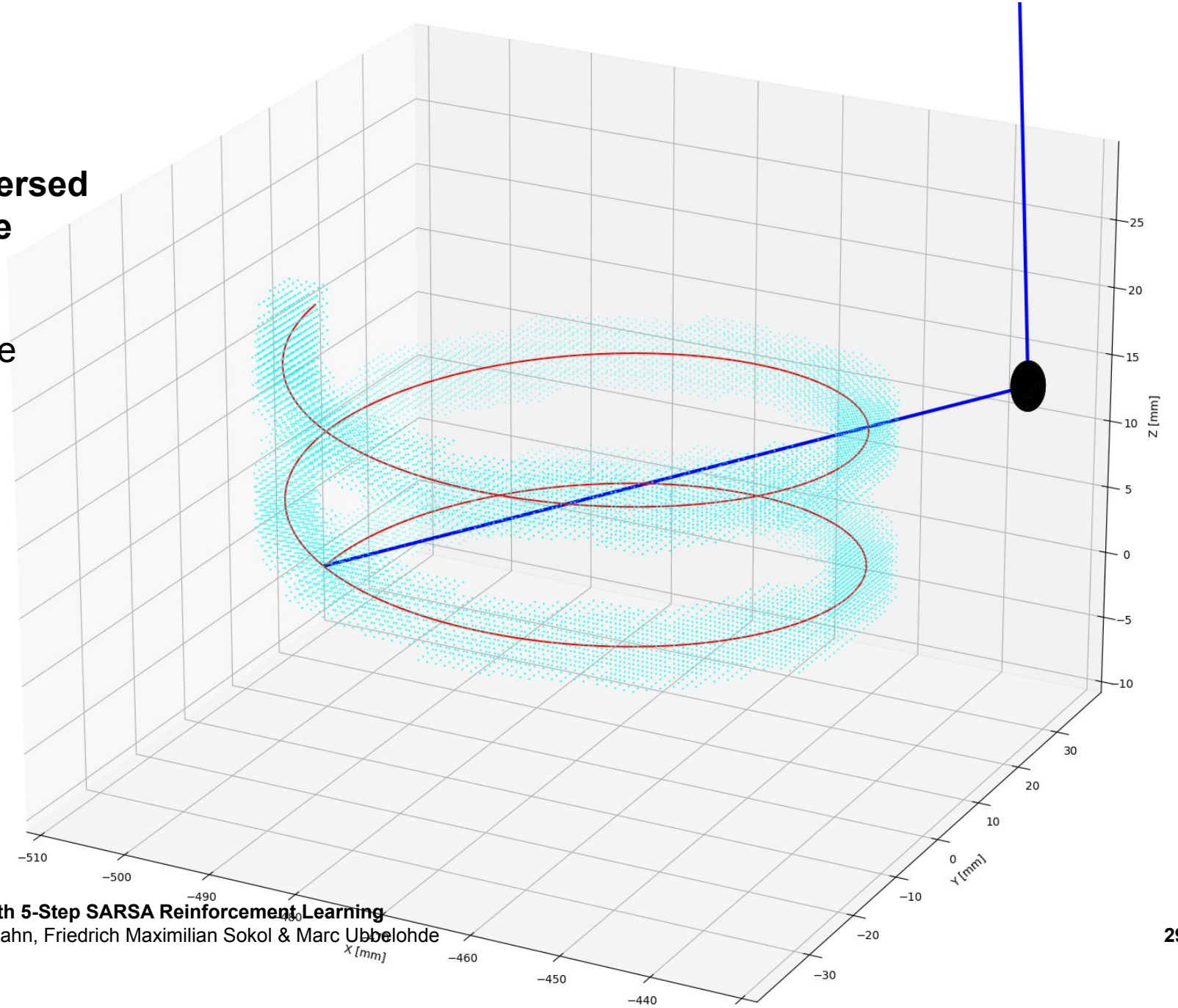


1st Approach

- **Problem: Helix needs to be traversed by pure chance at least one time**

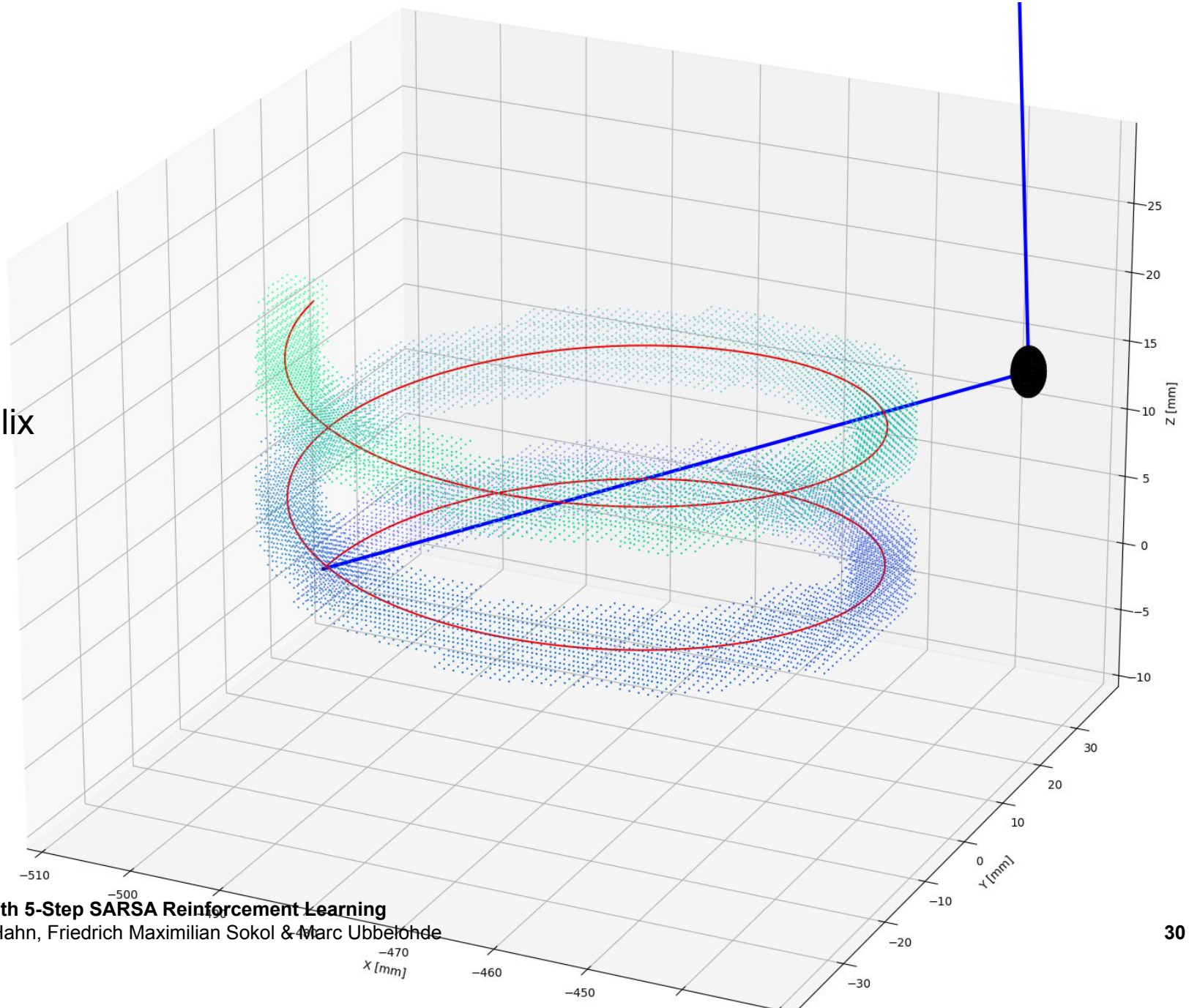
- Assumption: 10 out of 729 possible actions are good
- Probability to take good step: $\frac{100}{729}$
- Assumption: >1000 steps needed to finish helix, max of 10^9 steps
- Probability to finish Helix: $\sum_{n=1000}^{1\cdot 10^9} \frac{100}{729}^n$

practically zero



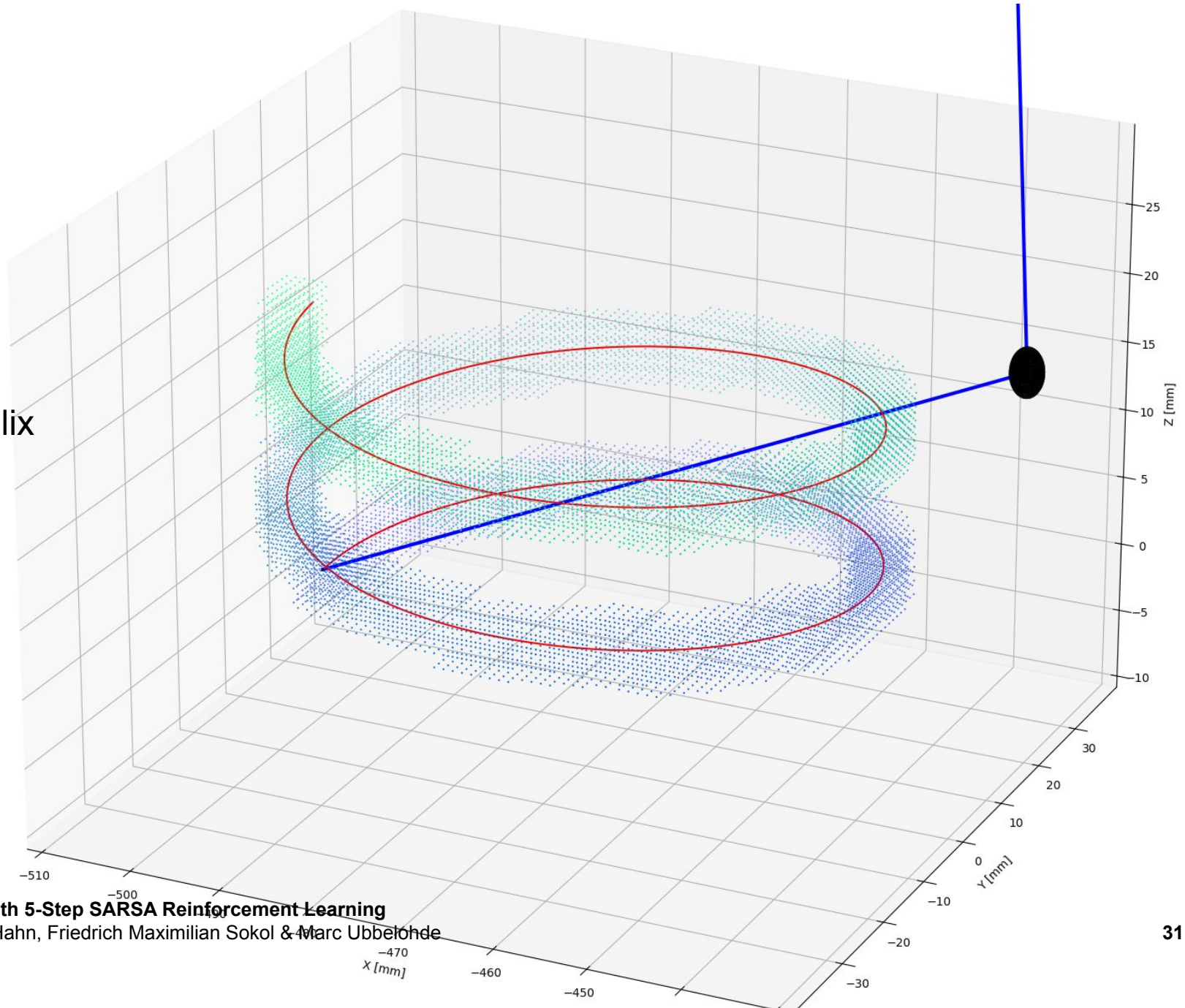
2nd Approach

- Learn whole helix at once
- Revised reward system:
  - -5 when out of bounds
  - -1 for step at beginning of helix
  - increase reward linearly until helix is finished
  - 0 when in finishing voxels



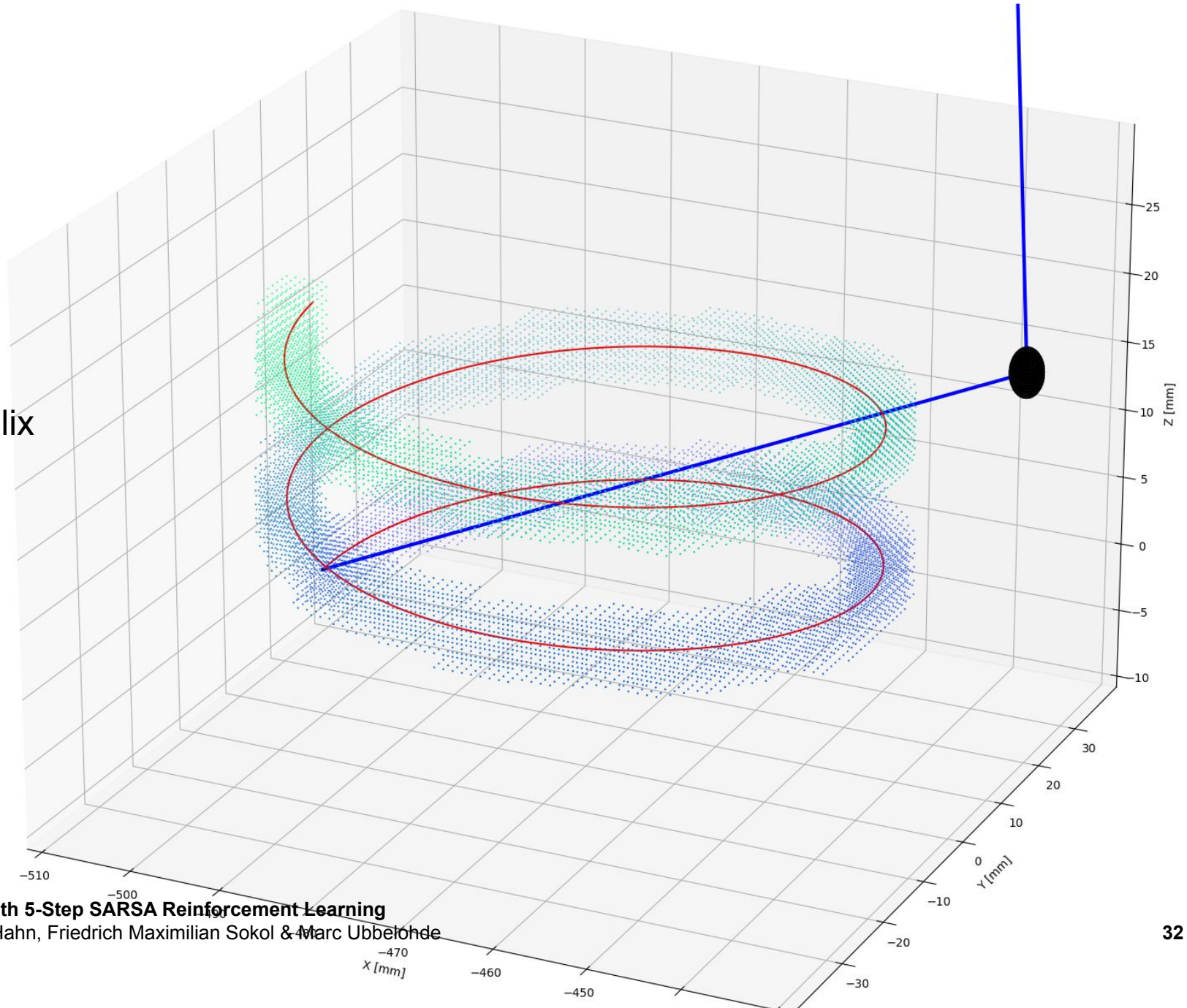
2nd Approach

- Learn whole helix at once
- Revised reward system:
  - -5 when out of bounds
  - -1 for step at beginning of helix
  - increase reward linearly until helix is finished
  - 0 when in finishing voxels
- … also never finished a single episode



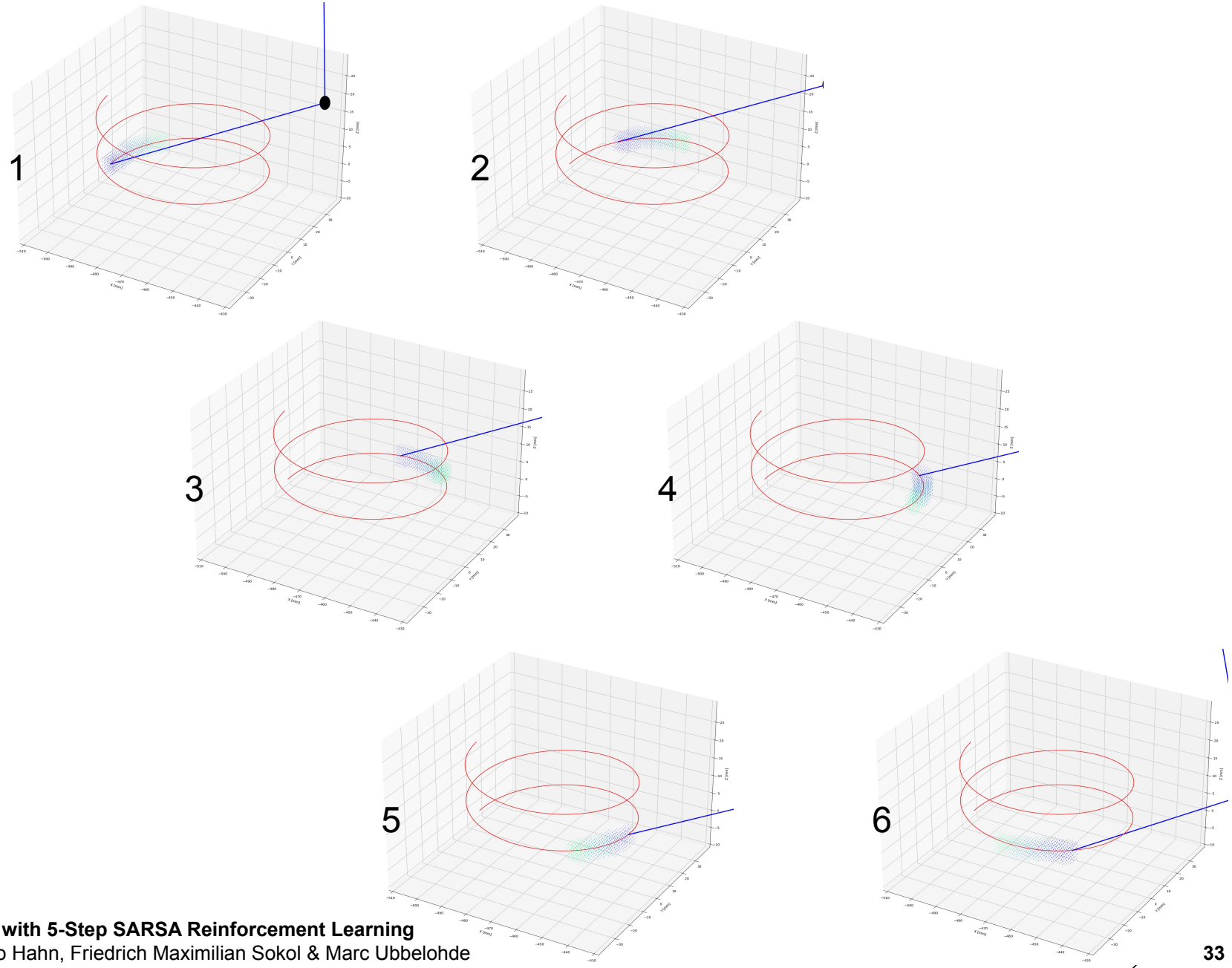
2nd Approach

- Learn whole helix at once
- Revised reward system:
  - -5 when out of bounds
  - -1 for step at beginning of helix
  - increase reward linearly until helix is finished
  - 0 when in finishing voxels
- … also never finished a single episode
- **But: it got further.**
- **Finished about 1/32 of the helix after 2 hours**



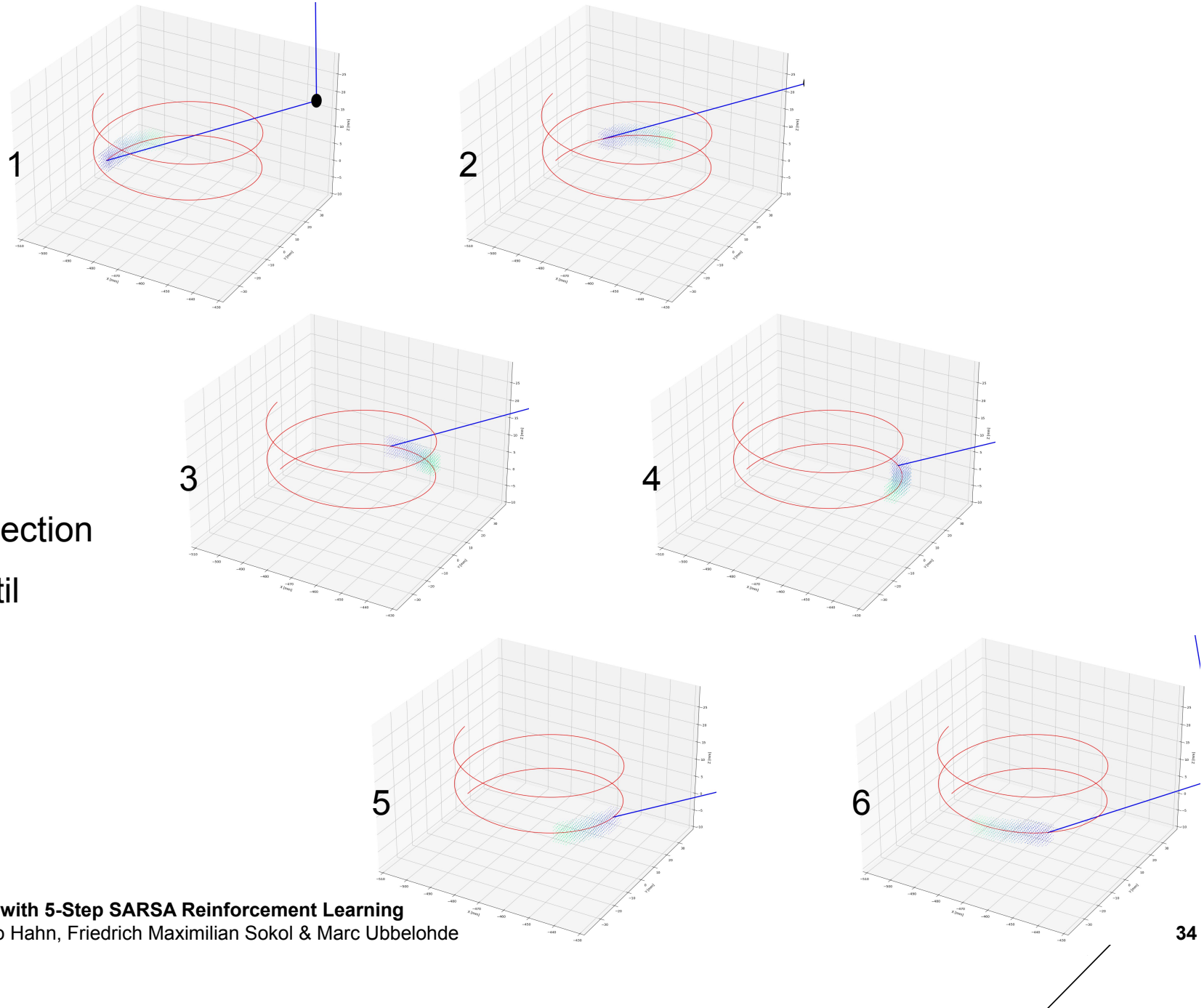
3rd Approach

- **Parallelization!**



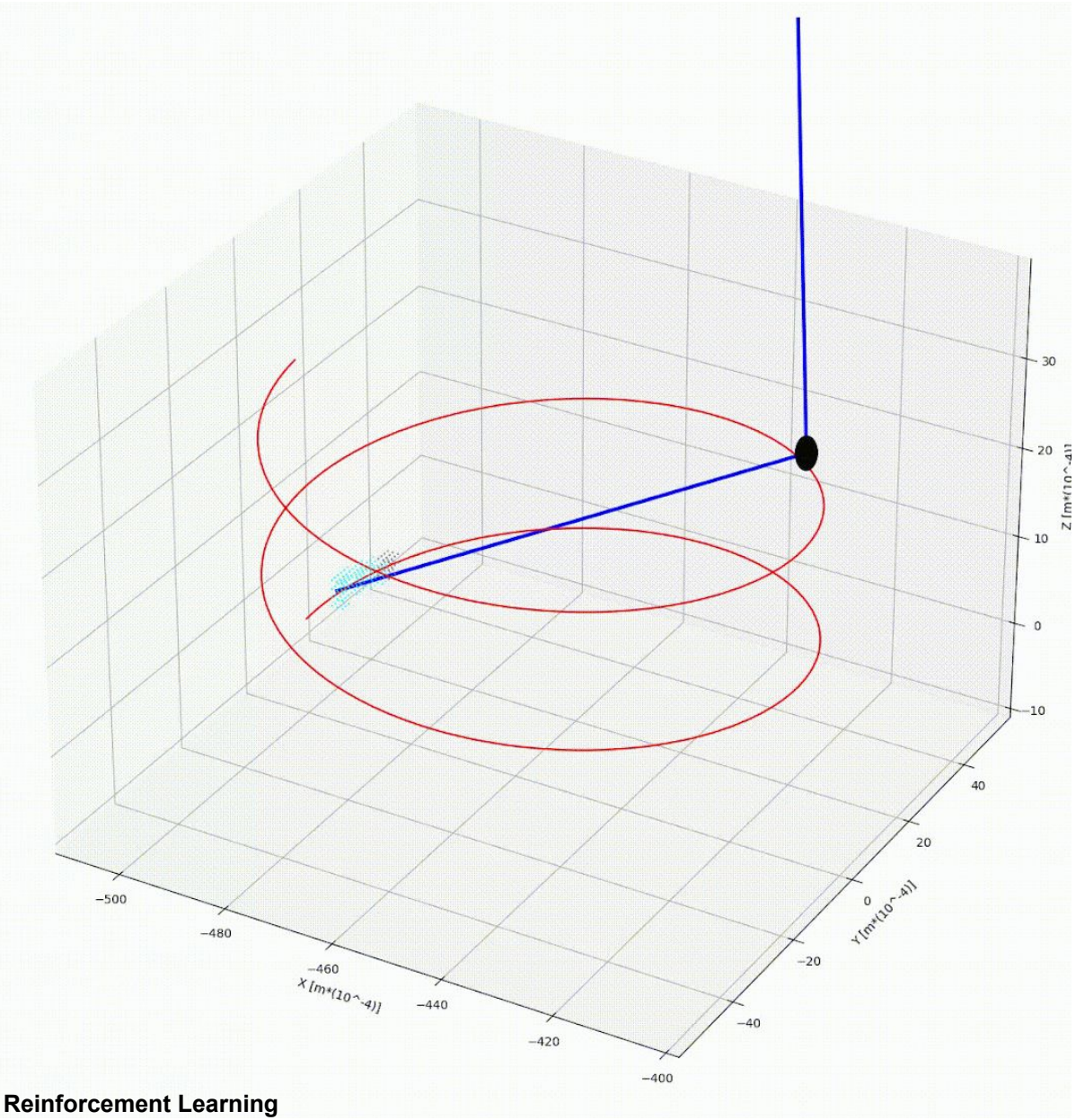
3rd Approach

- **Parallelization!**
- Split helix into sections
- Reward system:
  - -5 for going out of bounds
  - -1 for step at beginning of section
  - increase reward linearly until section is finished
  - 0 when in finishing voxels



3rd Approach

- **Parallelization!**
- Split helix into sections
- Reward system:
  - -5 for going out of bounds
  - -1 for step at beginning of section
  - increase reward linearly until section is finished
  - 0 when in finishing voxels
- **Sections are getting learned!**



3rd Approach

- **Parallelization!**
- What now?

```
                          Sections
          000  001  002  003  004  005  006  007
==================================================
Cycle   | 000  000  000  000  000  000  000  000
Episode| done 0299 0003 0019 0001 0000 0000 0000
```

```
max@Toaster: ~/Documents/Studium/WiSe2324/Reinforcement_Learning/rl_team_03/code 91x26

  0[||||||      28.6%]   3[||||||||||79.9%]   6[||||||||||68.3%]   9[||||||||||100.0%]
  1[||||||||||88.5%]   4[||||||||||100.0%]   7[||||||||    44.7%]  10[||||||||    44.1%]
  2[||||||||    43.4%]   5[||||||||||100.0%]   8[||||||||||100.0%]  11[||||||||||77.4%]
Mem[||||||||||||||||||||||||||6.39G/13.3G] Tasks: 176, 1209 thr, 207 kthr; 9 running
Swp[                            0K/4.00G] Load average: 6.43 2.92 1.43
                                          Uptime: 08:46:37

 Main  I/O
   PID USER      PRI  NI  VIRT   RES   SHR S CPU%▽MEM%   TIME+  Command
 30170 max      20   0  575M 62848 21376 R 102.1  0.4  0:50.14 python3 reinforce.py
 30184 max      20   0  577M 52604  9616 R 101.7  0.4  0:50.33 python3 reinforce.py
 30186 max      20   0  577M 52744  9616 R 101.7  0.4  0:49.86 python3 reinforce.py
 30201 max      20   0  577M 52616  9616 R 101.7  0.4  0:49.26 python3 reinforce.py
 30204 max      20   0  577M 52620  9616 R 101.7  0.4  0:49.03 python3 reinforce.py
 30185 max      20   0  577M 52604  9616 R 101.3  0.4  0:50.04 python3 reinforce.py
 30188 max      20   0  577M 52744  9744 R 101.3  0.4  0:49.66 python3 reinforce.py
 30189 max      20   0  577M 52360  9488 R 101.3  0.4  0:49.45 python3 reinforce.py
```

3rd Approach

- **Parallelization!**
- What now?
- Stitch sections together to receive the full policy
- Finish training based on the stitched Q values

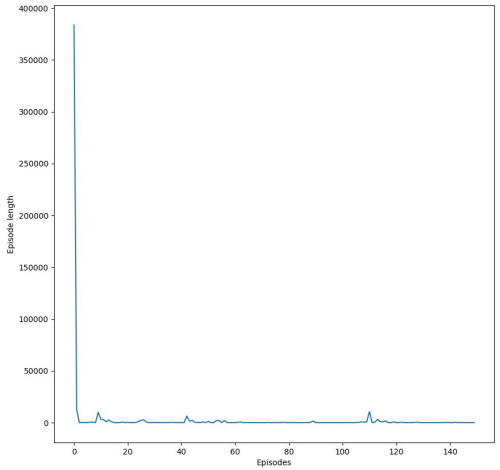
```
                          Sections
          000   001   002   003   004   005   006   007
=================================================
Cycle   | 000   000   000   000   000   000   000   000
Episode| done 0299 0003 0019 0001 0000 0000 0000
```

```
max@Toaster: ~/Documents/Studium/WiSe2324/Reinforcement_Learning/rl_team_03/code 91x26

  0[||||||      28.6%]   3[||||||||||79.9%]   6[||||||||||68.3%]   9[||||||||||100.0%]
  1[||||||||||88.5%]   4[||||||||||100.0%]   7[||||||||    44.7%]  10[||||||||    44.1%]
  2[||||||||    43.4%]   5[||||||||||100.0%]   8[||||||||||100.0%]  11[||||||||||77.4%]
Mem[||||||||||||||||||||||||||6.39G/13.3G] Tasks: 176, 1209 thr, 207 kthr; 9 running
Swp[                            0K/4.00G] Load average: 6.43 2.92 1.43
                                          Uptime: 08:46:37

 Main   I/O
   PID USER      PRI  NI  VIRT   RES   SHR S CPU%▽MEM%   TIME+  Command
 30170 max      20   0  575M 62848 21376 R 102.1  0.4  0:50.14 python3 reinforce.py
 30184 max      20   0  577M 52604  9616 R 101.7  0.4  0:50.33 python3 reinforce.py
 30186 max      20   0  577M 52744  9616 R 101.7  0.4  0:49.86 python3 reinforce.py
 30201 max      20   0  577M 52616  9616 R 101.7  0.4  0:49.26 python3 reinforce.py
 30204 max      20   0  577M 52620  9616 R 101.7  0.4  0:49.03 python3 reinforce.py
 30185 max      20   0  577M 52604  9616 R 101.3  0.4  0:50.04 python3 reinforce.py
 30188 max      20   0  577M 52744  9744 R 101.3  0.4  0:49.66 python3 reinforce.py
 30189 max      20   0  577M 52360  9488 R 101.3  0.4  0:49.45 python3 reinforce.py
```

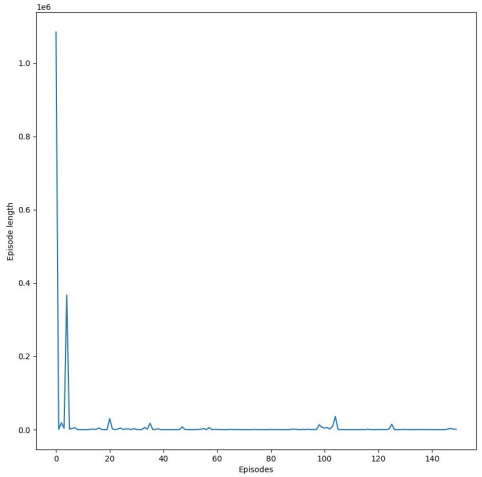
3rd Approach

- **Parallelization!**

Section 1



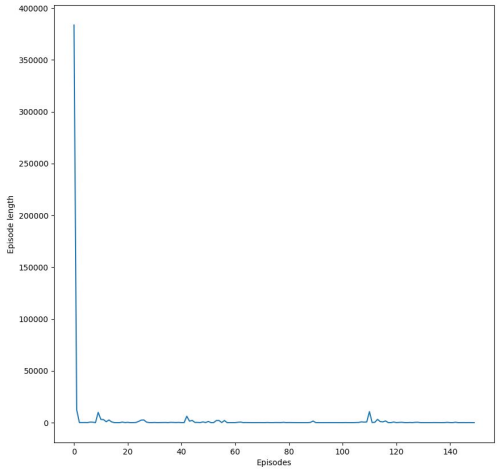
Section 2



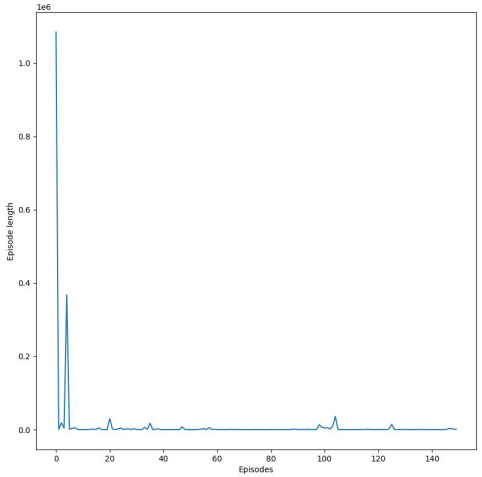
3rd Approach

- **Parallelization!**
- After stitching sections together:
  - good start in episode lengths
  - episode length increases with training time

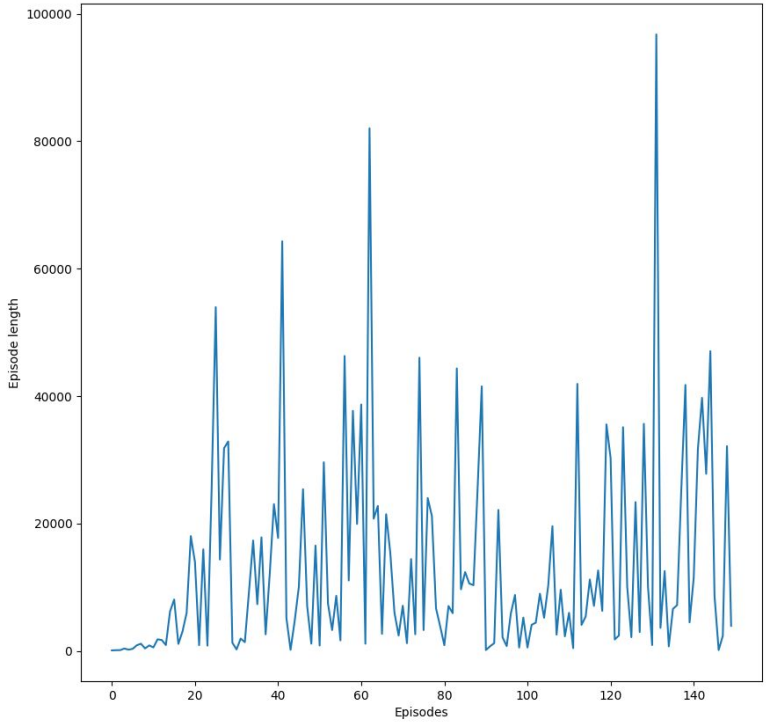
Section 1



Section 2



Section 1 + 2

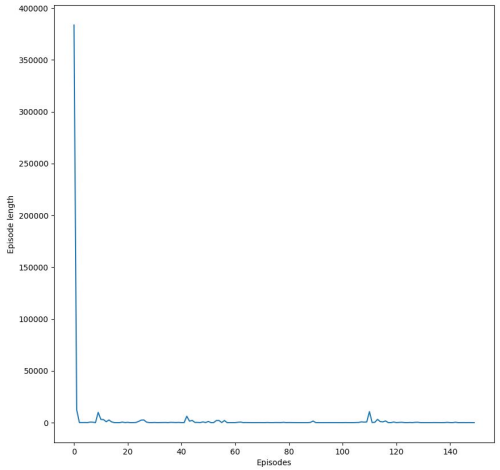


3rd Approach

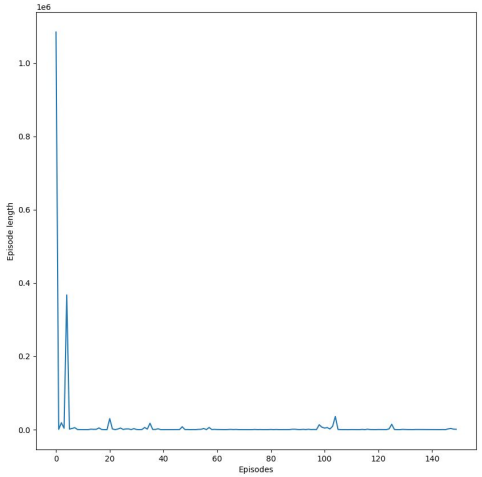
- **Parallelization!**
- After stitching sections together:
  - good start in episode lengths
  - episode length increases with training time

WHY?

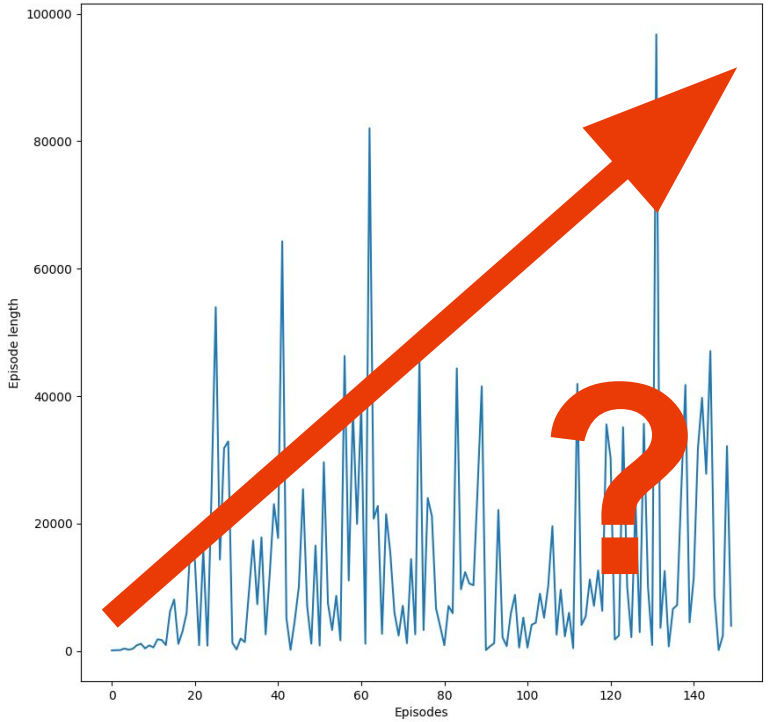
Section 1



Section 2

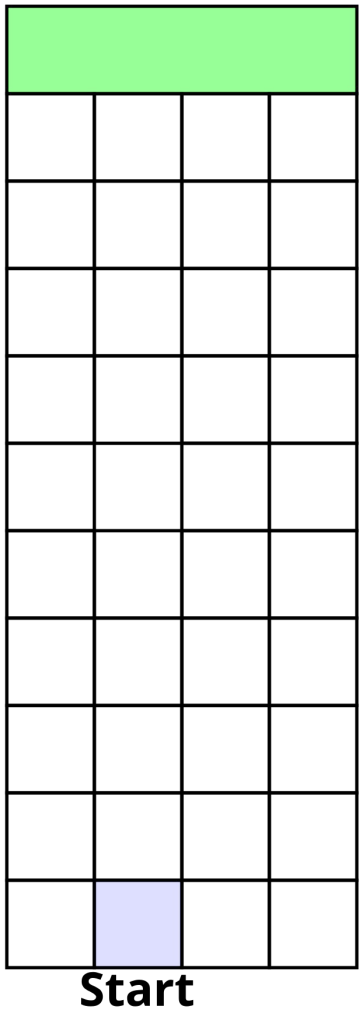


Section 1 + 2



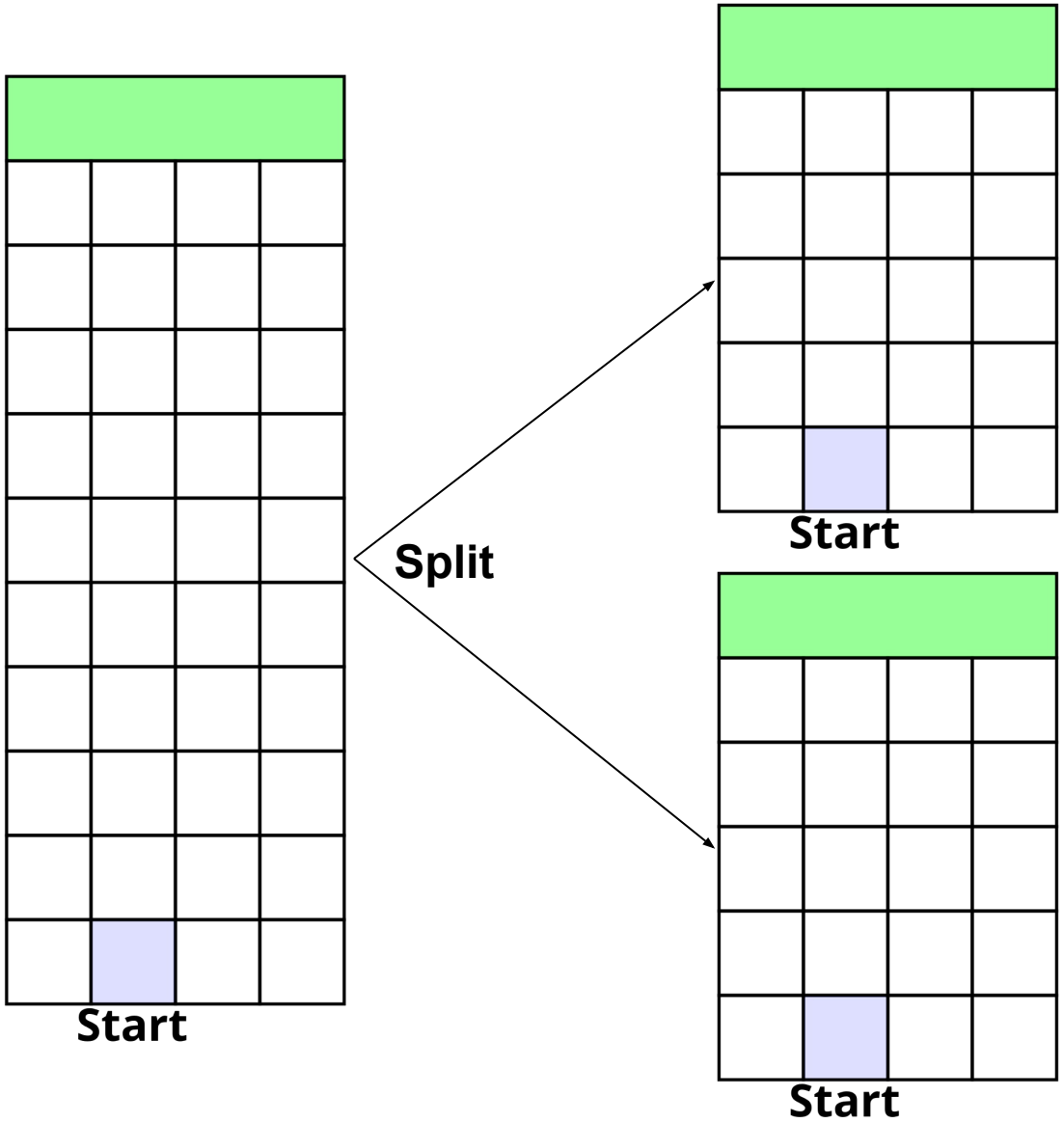
3rd Approach

- **Let' simplify…**
- Environment:
  - Gridworld
  - Possible Moves:
    - North, South, East, West
- Reward System:
  - -1 every step
  - 0 when in winning pixels



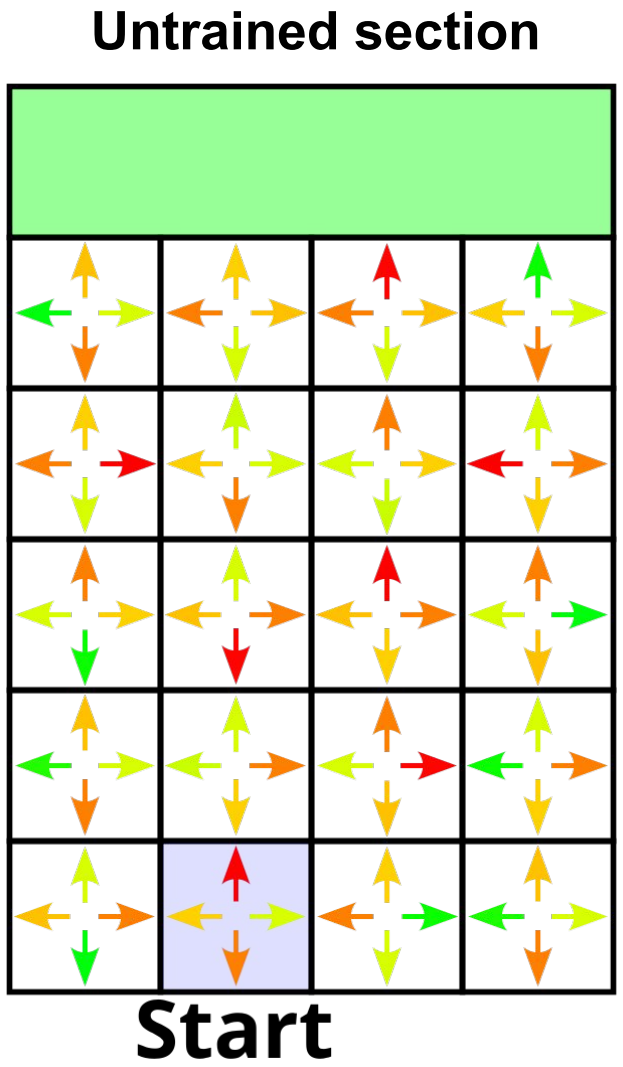
3rd Approach

- **Let' simplify…**
- Environment:
  - Gridworld
  - Possible Moves:
    - North, South, East, West
- Reward System:
  - -1 every step
  - 0 when in winning pixels



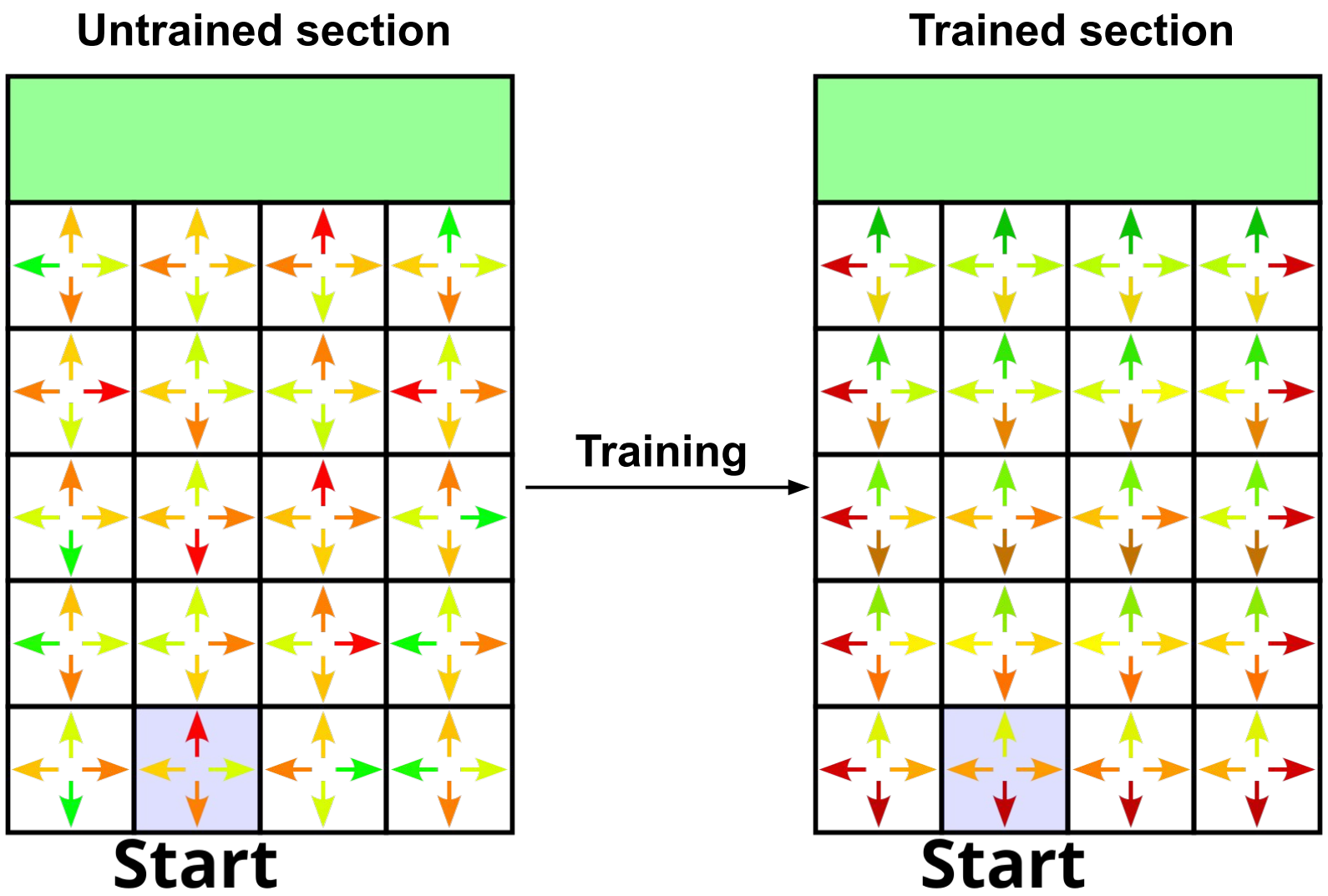
3rd Approach

- Green arrow = high Q value
- Red arrow = low Q value



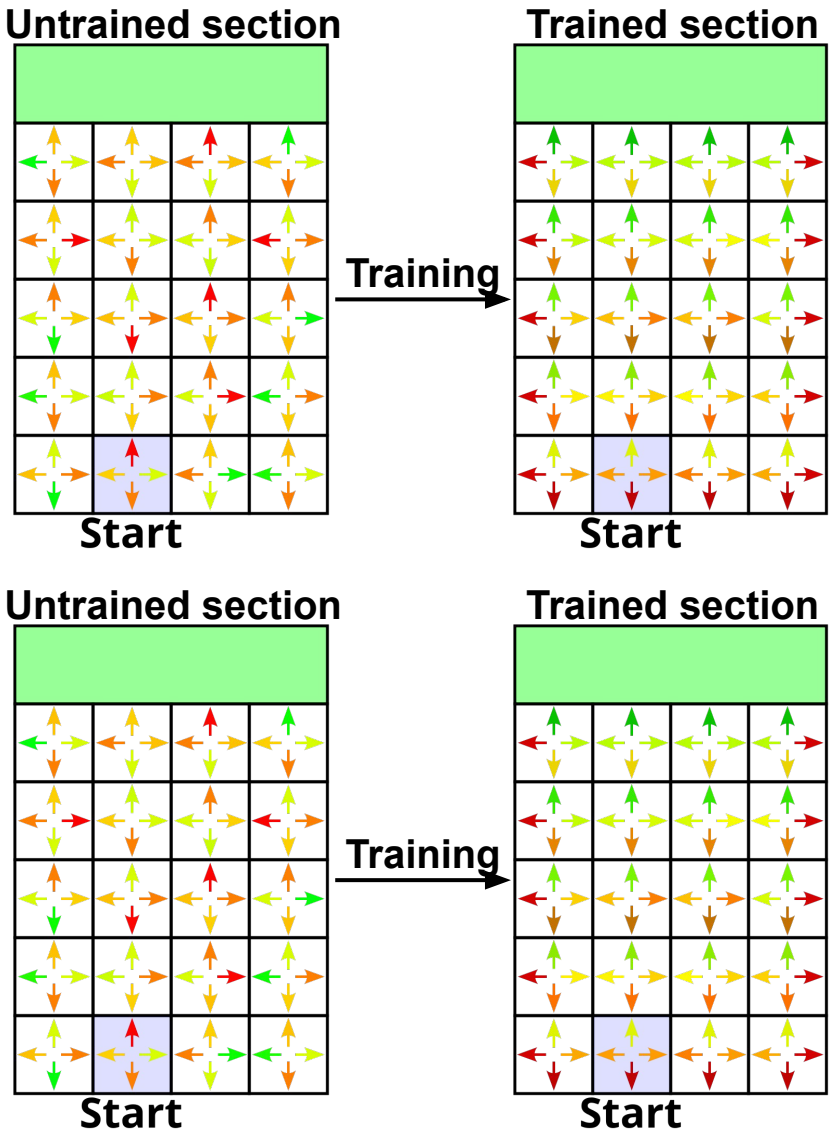
3rd Approach

- Green arrow = high Q value
- Red arrow = low Q value

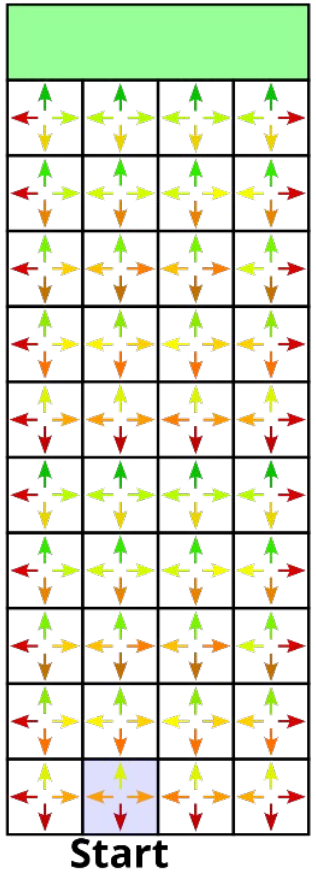


3rd Approach

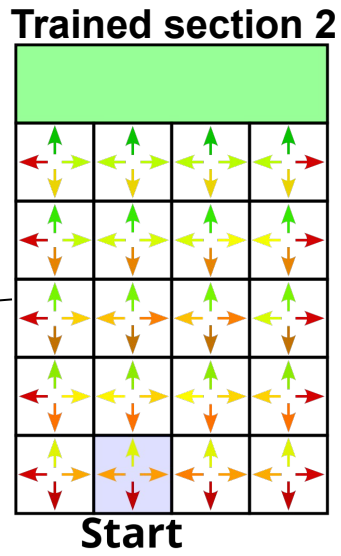
- Green arrow = high Q value
- Red arrow = low Q value
- **Train both sections!**

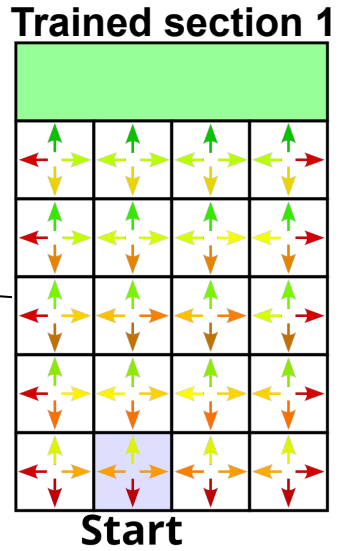


3rd Approach

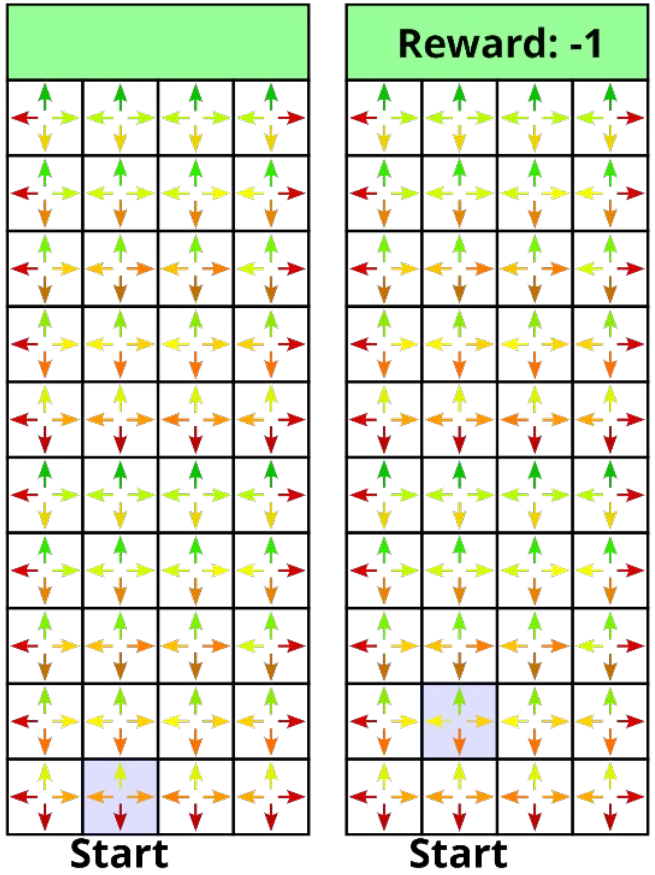


- … stitch sections together and train even more

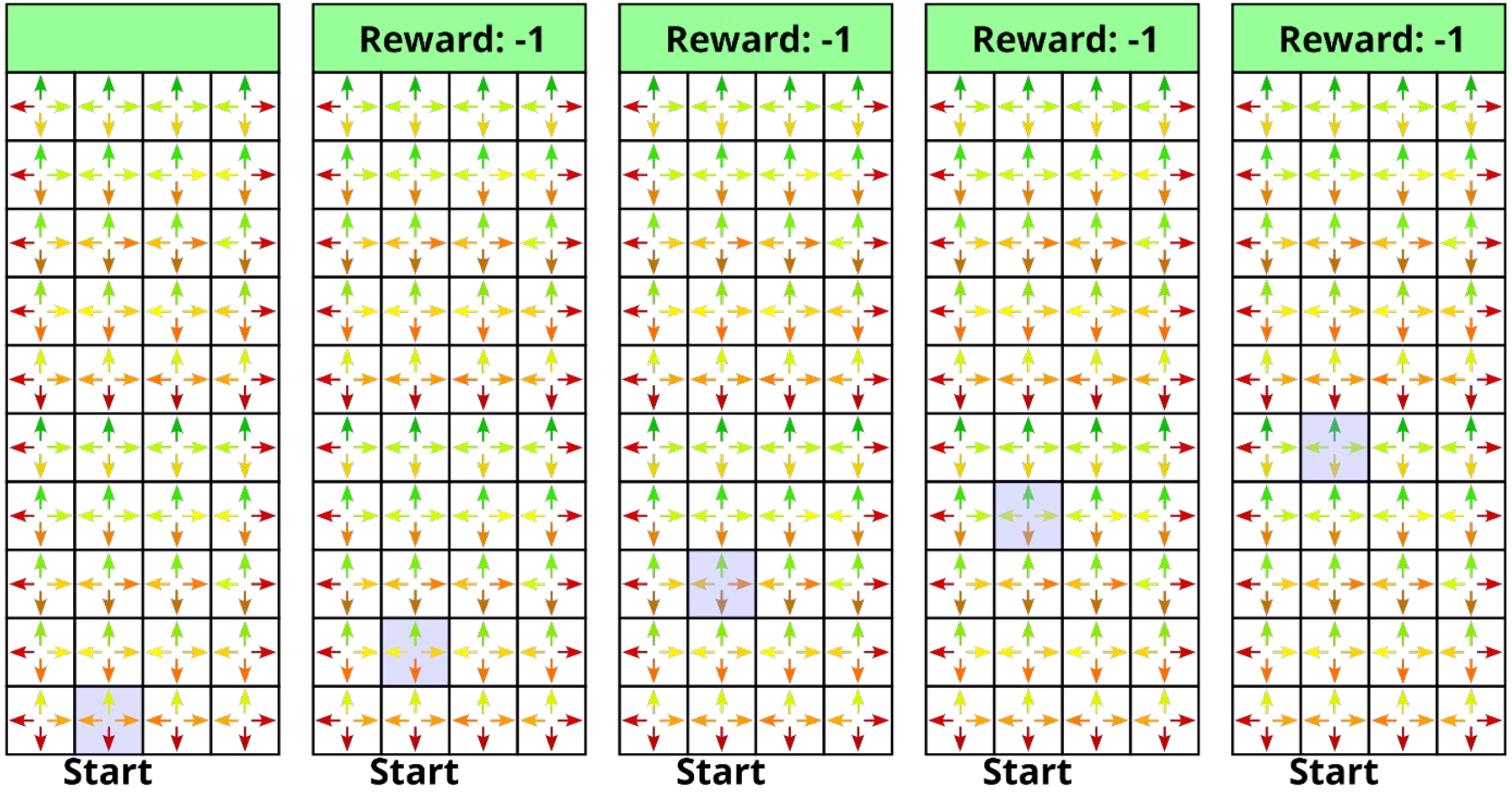




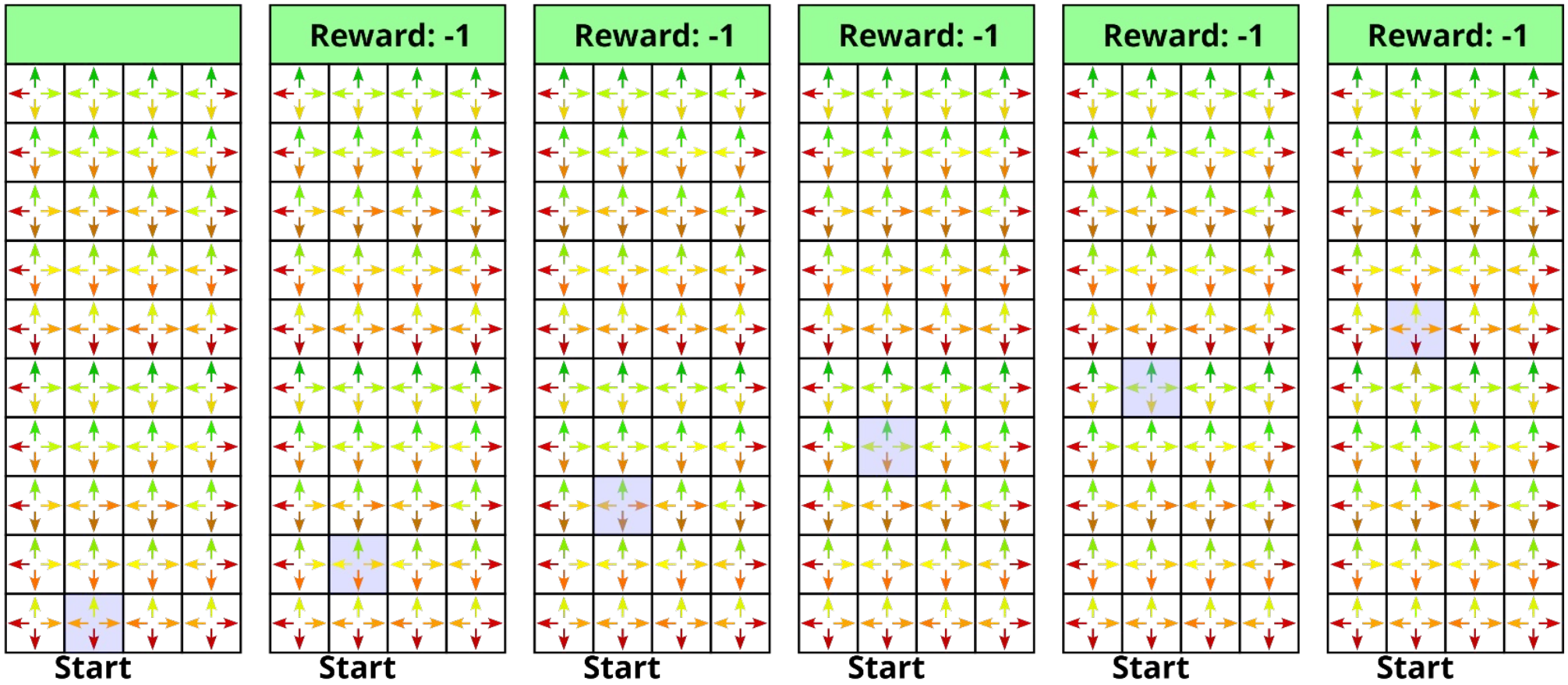
3rd Approach



3rd Approach

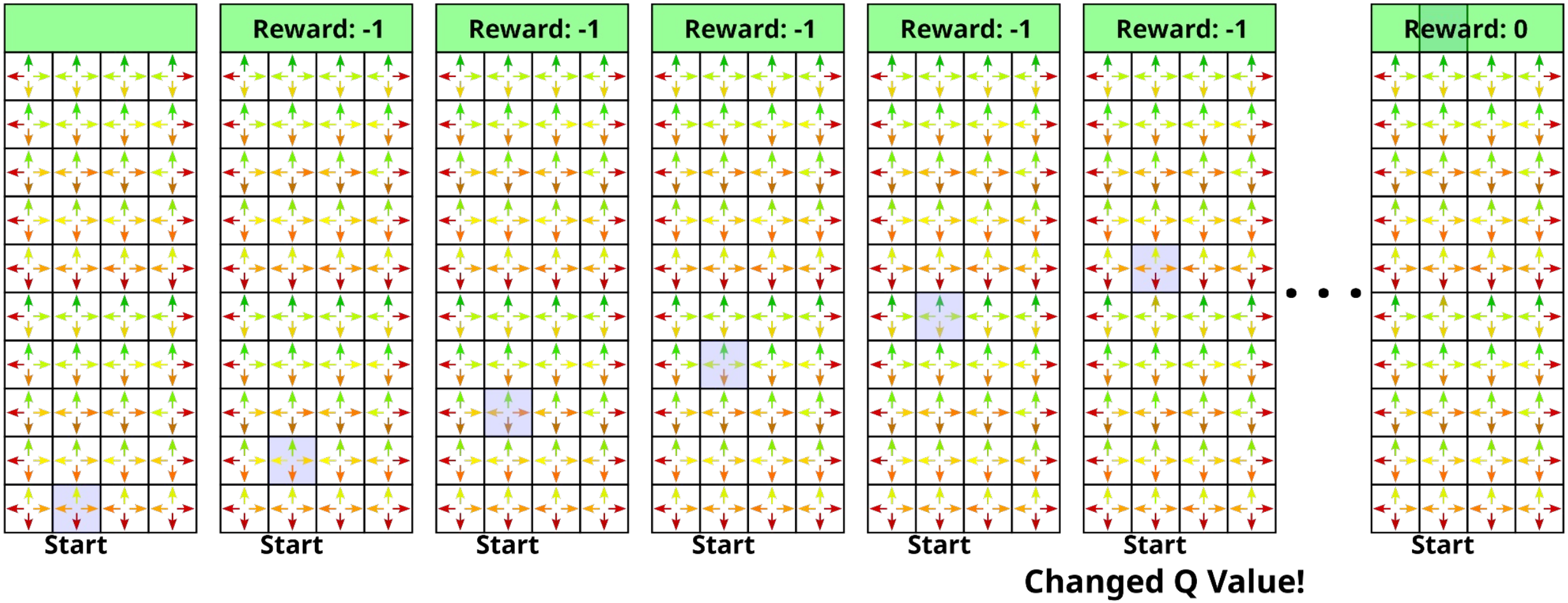


3rd Approach

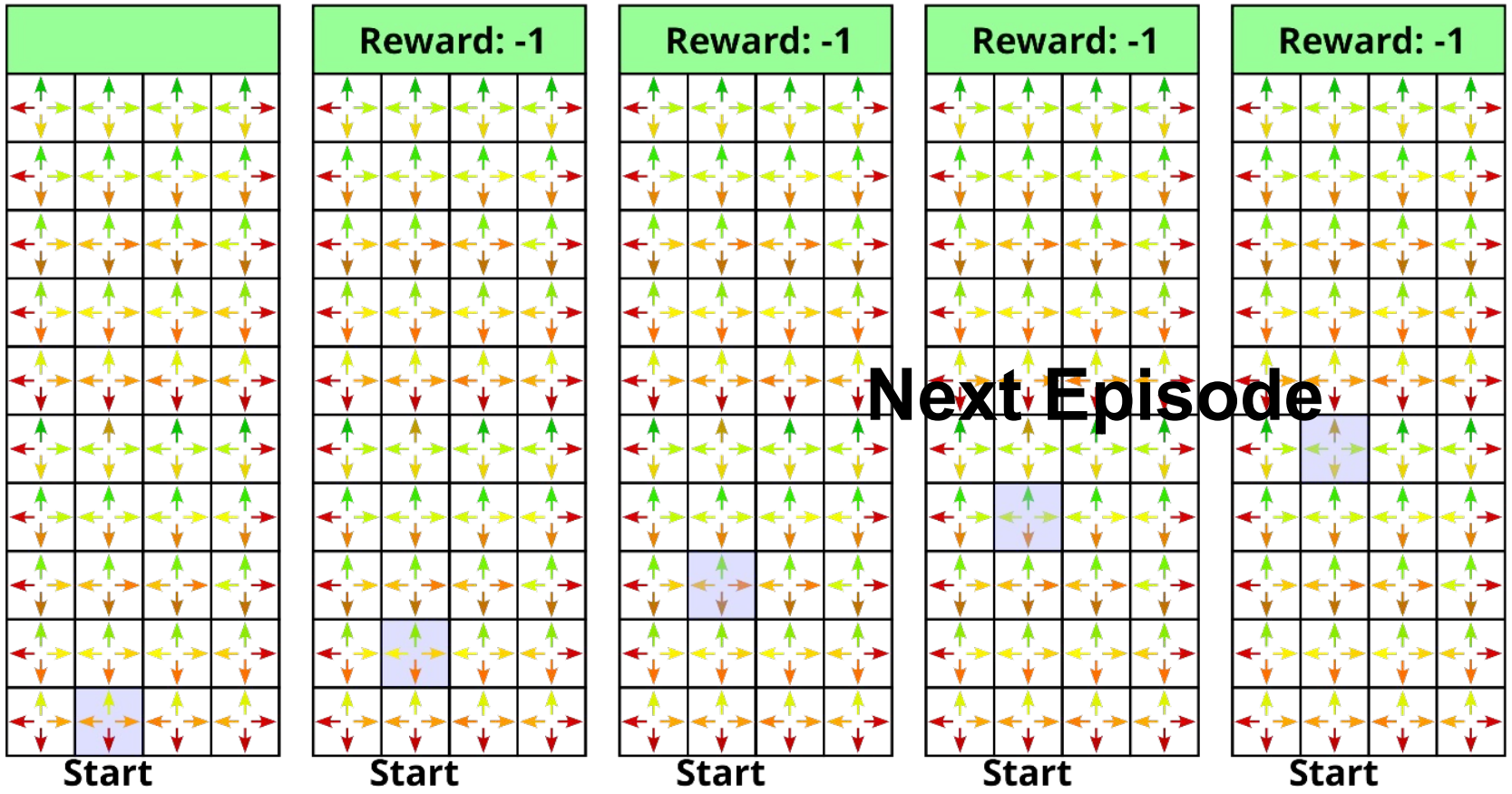


We get penalized where we didn't get penalized before!

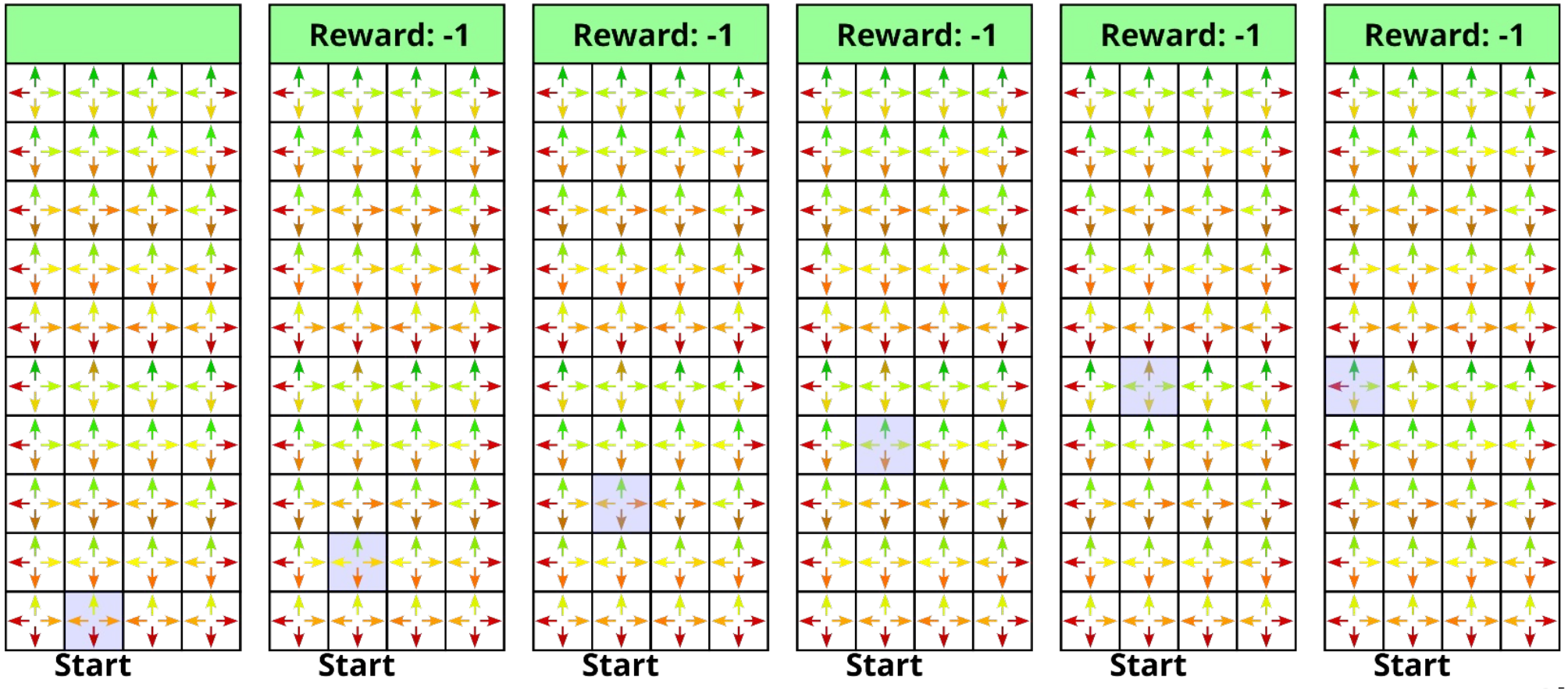
3rd Approach



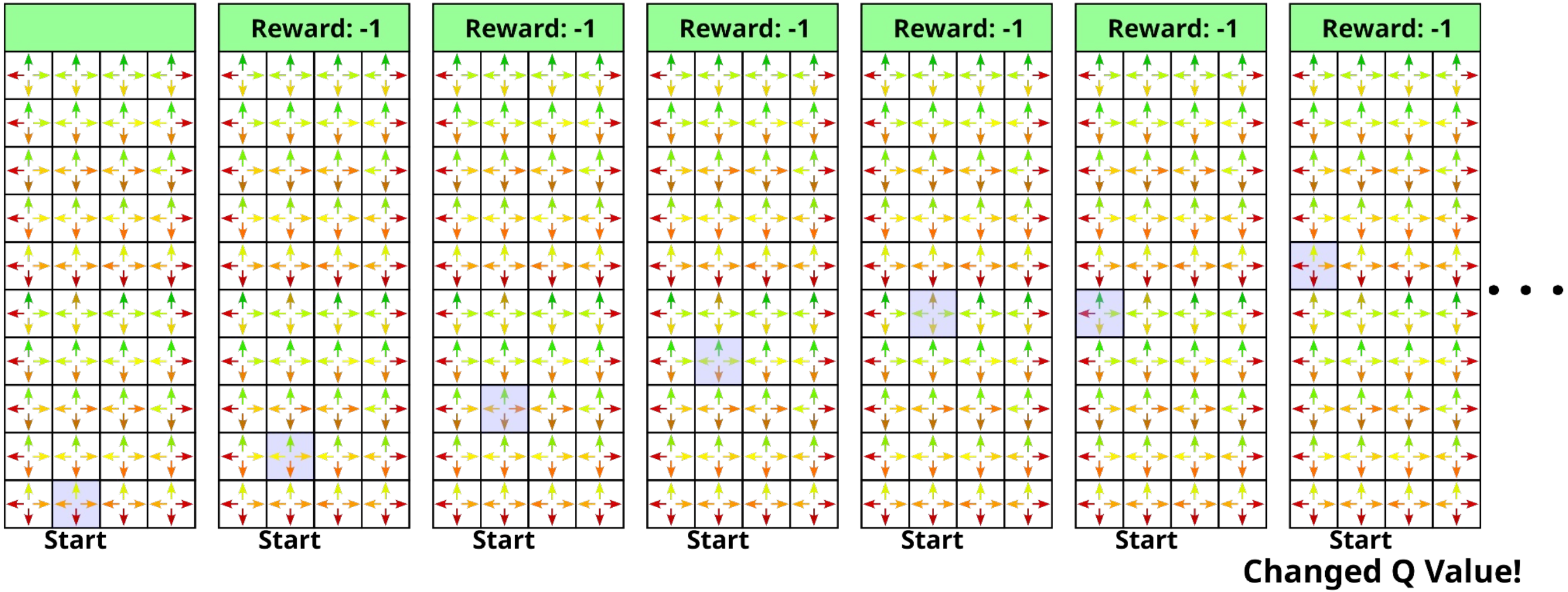
3rd Approach



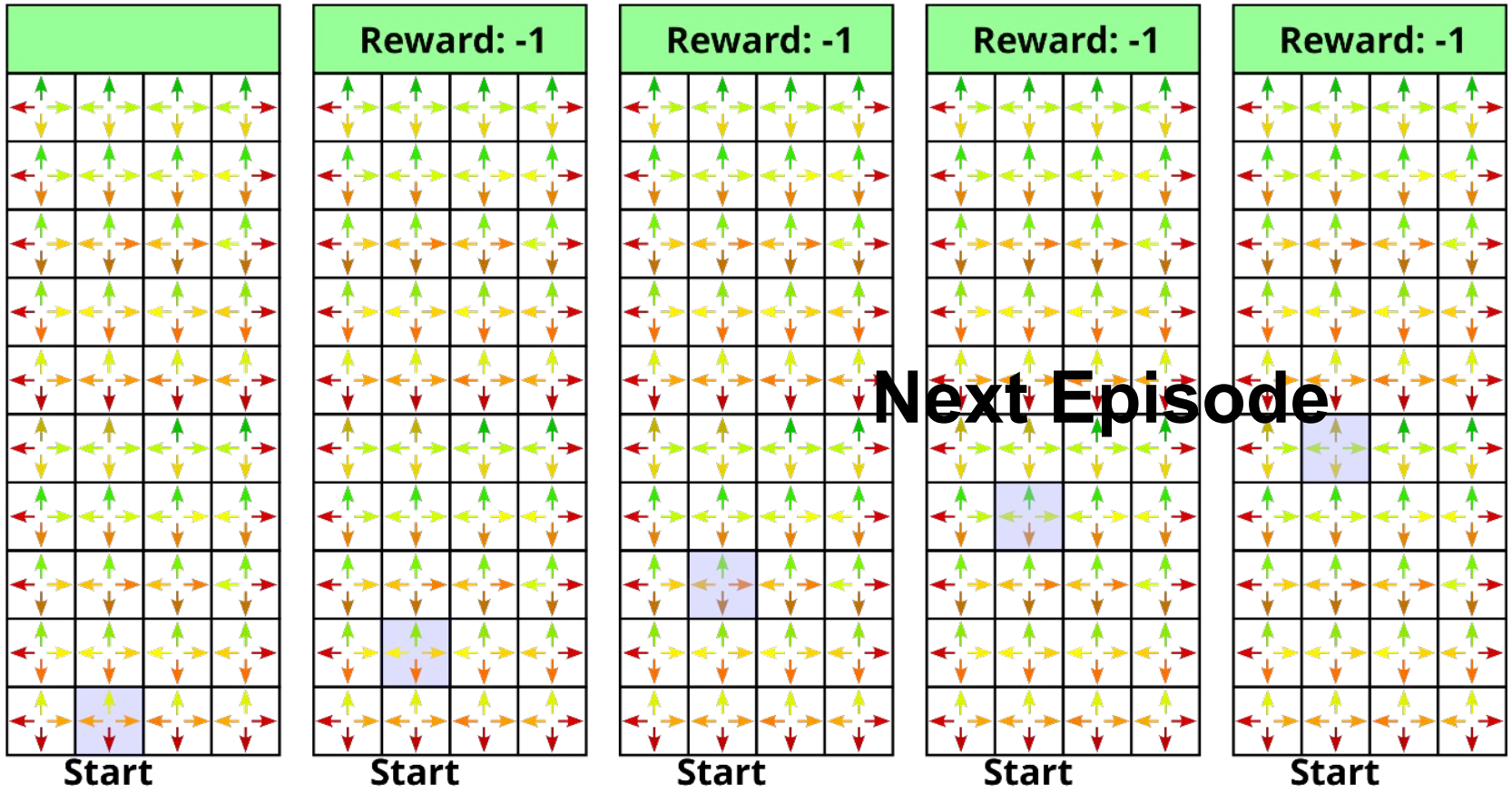
3rd Approach



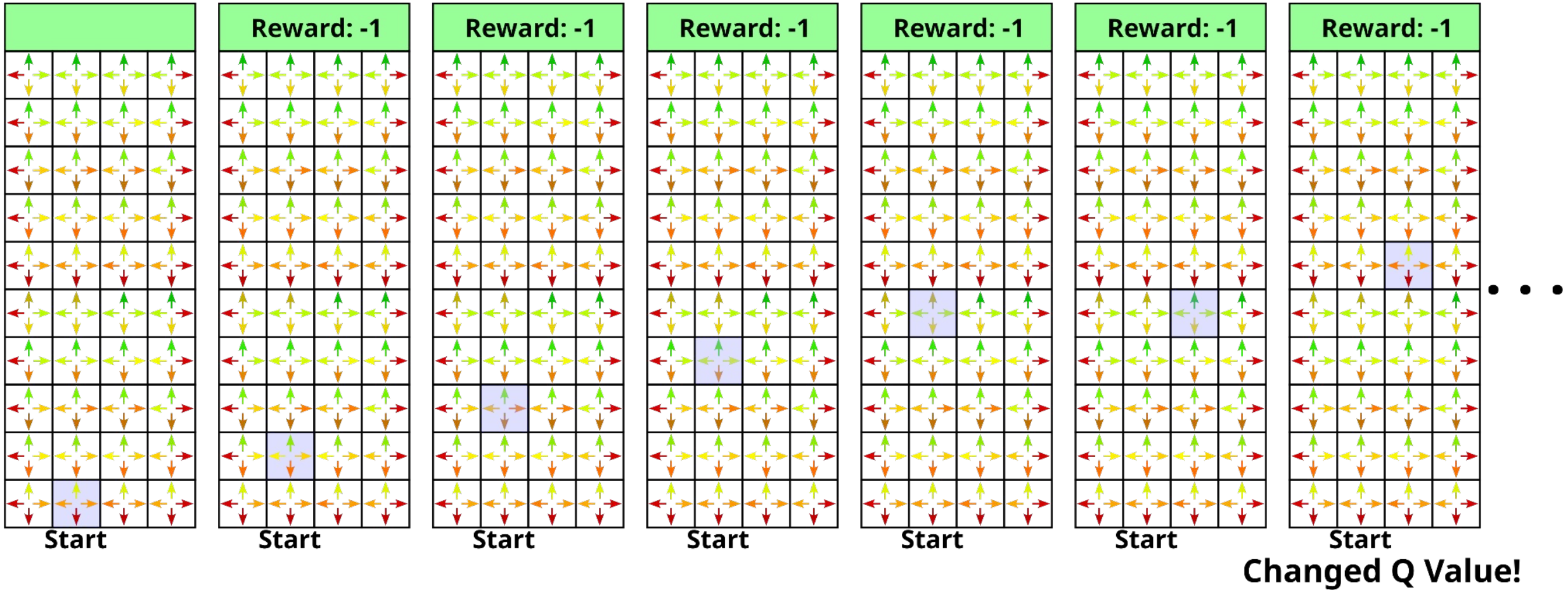
3rd Approach



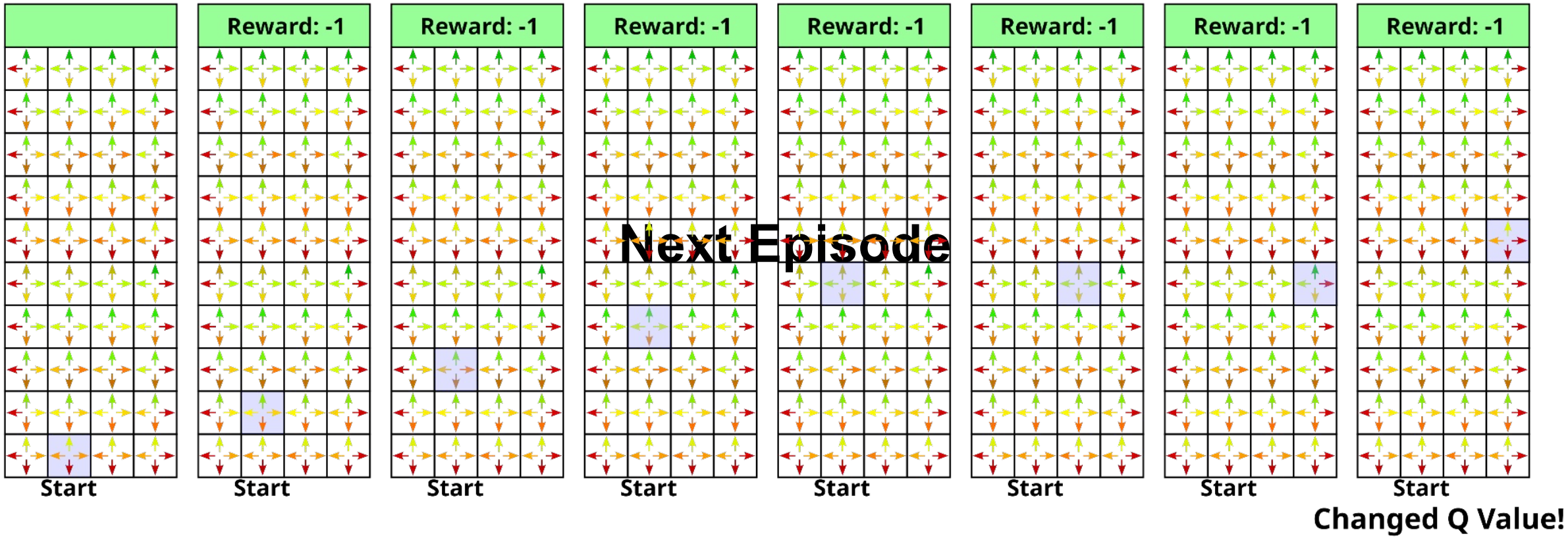
3rd Approach



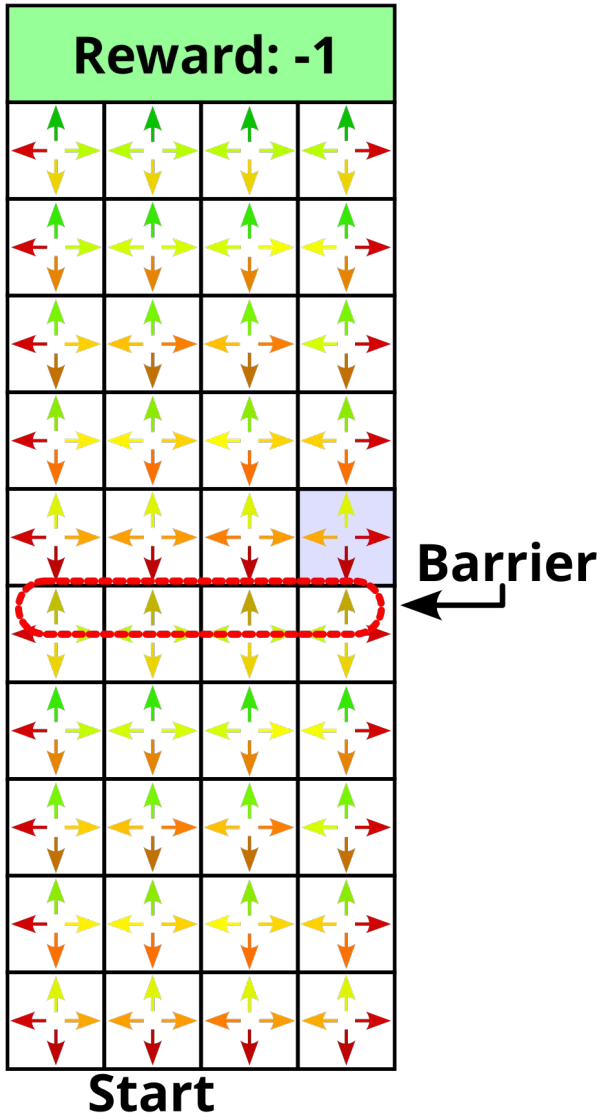
3rd Approach



3rd Approach

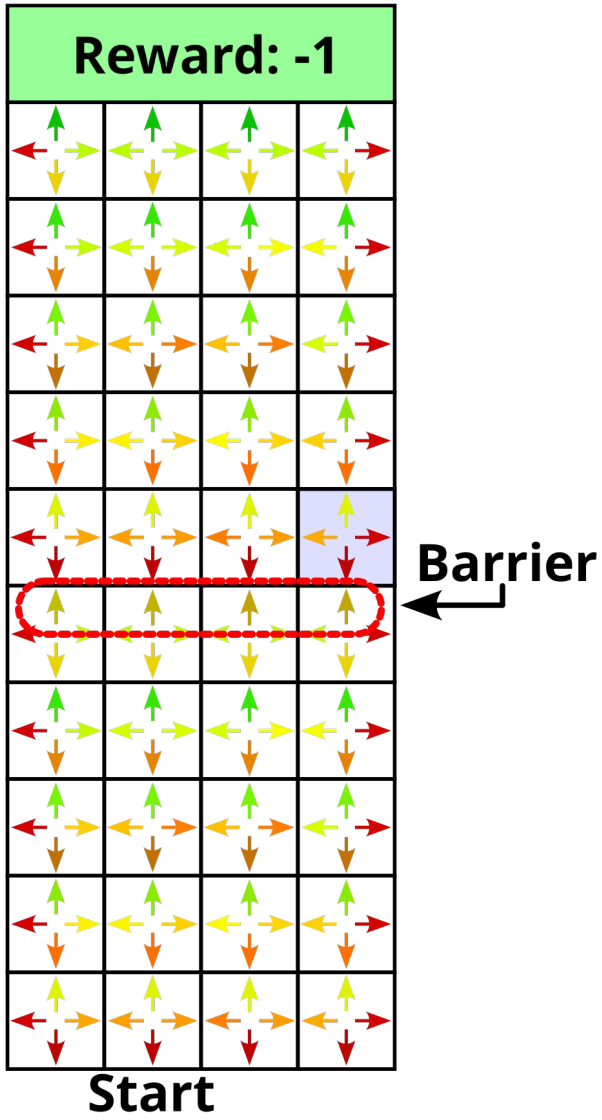


3rd Approach



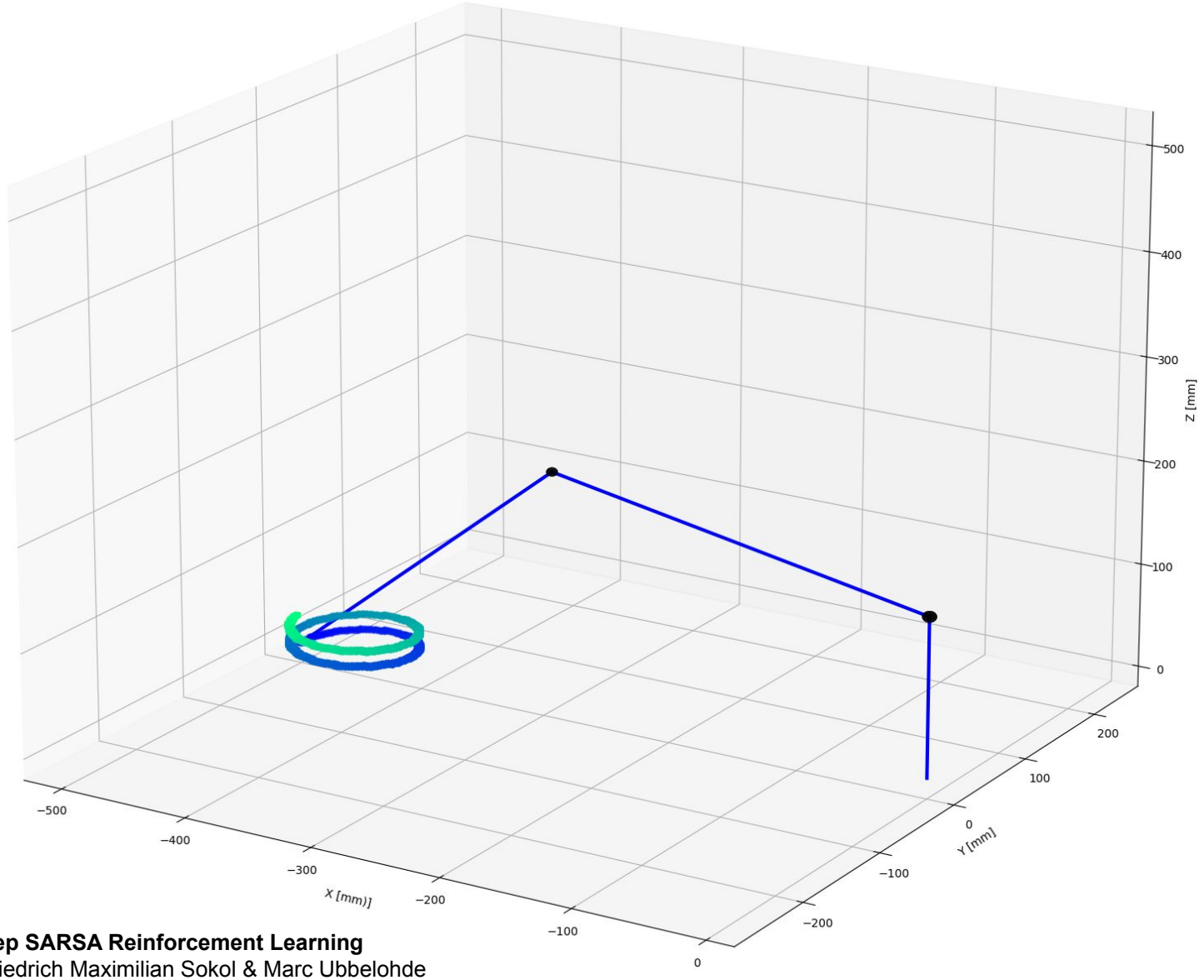
3rd Approach

- Good Q values unlearned
- Experimented with different reward structures
- No well working solution for this problem found
- **⇒ unsatisfactory**



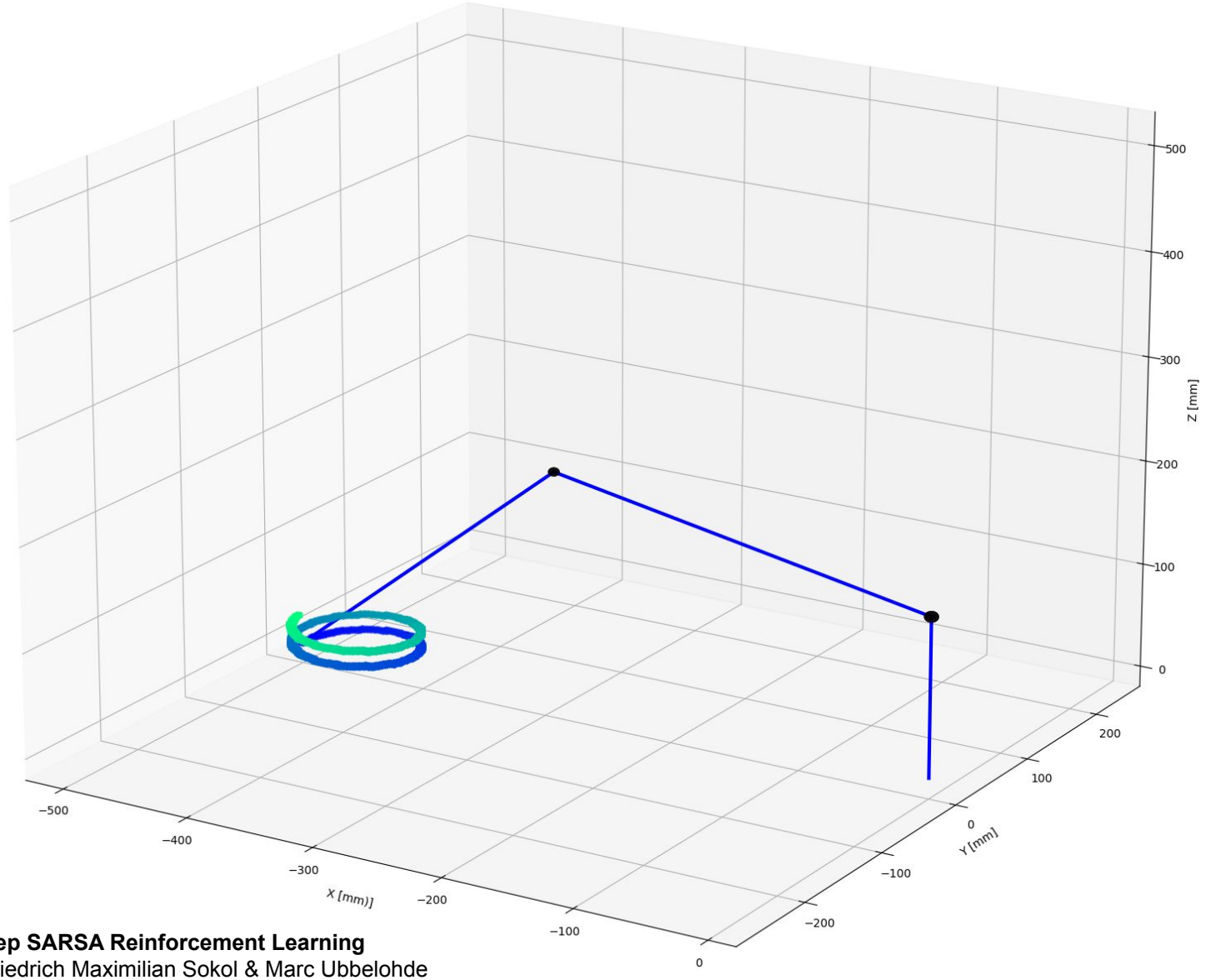
4th Approach

- Simplify robot:
  - Three axis
  - ⇒ $3^3 = 27$ Possible Moves per step



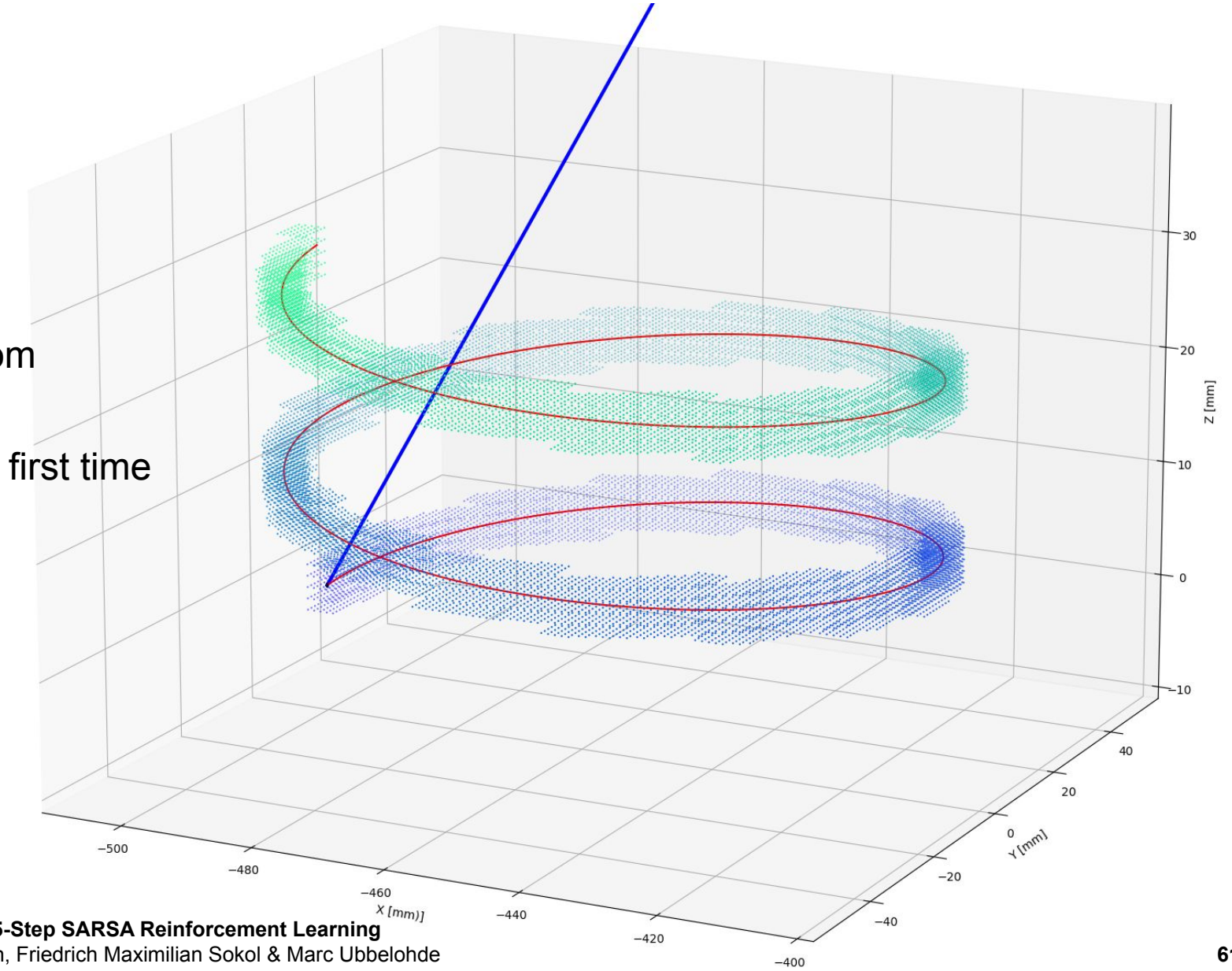
4th Approach

- Simplify robot:
  - Three axis
  - ⇒ $3^3 = 27$ Possible Moves per step
- Implement checkpoints
- Once a checkpoint in an episode is passed, resets only to checkpoint



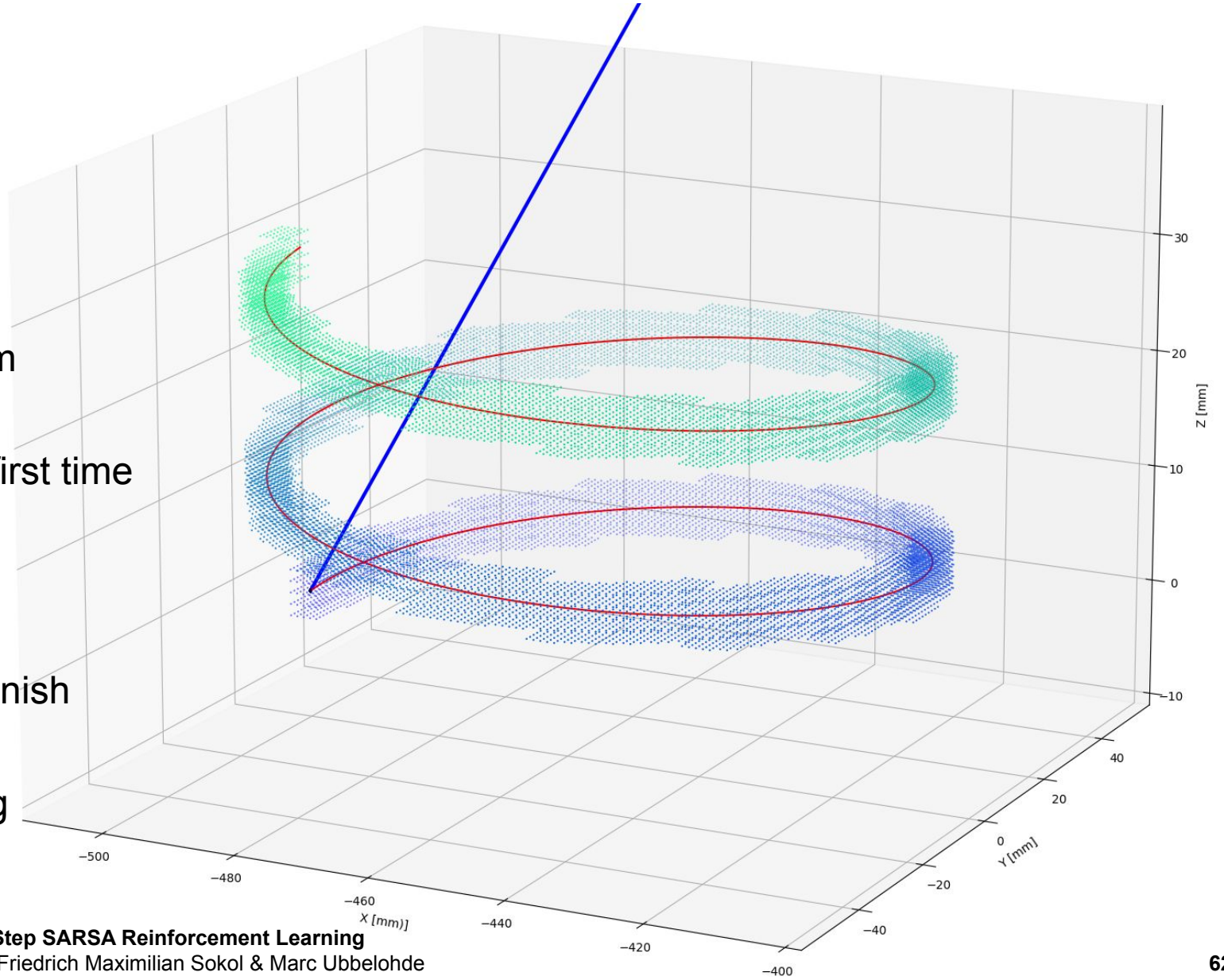
4th Approach

- Revised reward system:
  - -500 for going out of bounds
  - -250 for moving backwards
  - Linear increase of step reward from -100 to 0 moving along helix
  - 0 for passing a checkpoint for the first time
  - 100 when in finishing voxels



4th Approach

- Revised reward system:
  - -500 for going out of bounds
  - -250 for moving backwards
  - Linear increase of step reward from -100 to 0 moving along helix
  - 0 for passing a checkpoint for the first time
  - 100 when in finishing voxels
- **Goals:**
  - Reward more aggressively and punish going into the wrong direction
  - Keep sectioning but still train using whole policy



4th Approach

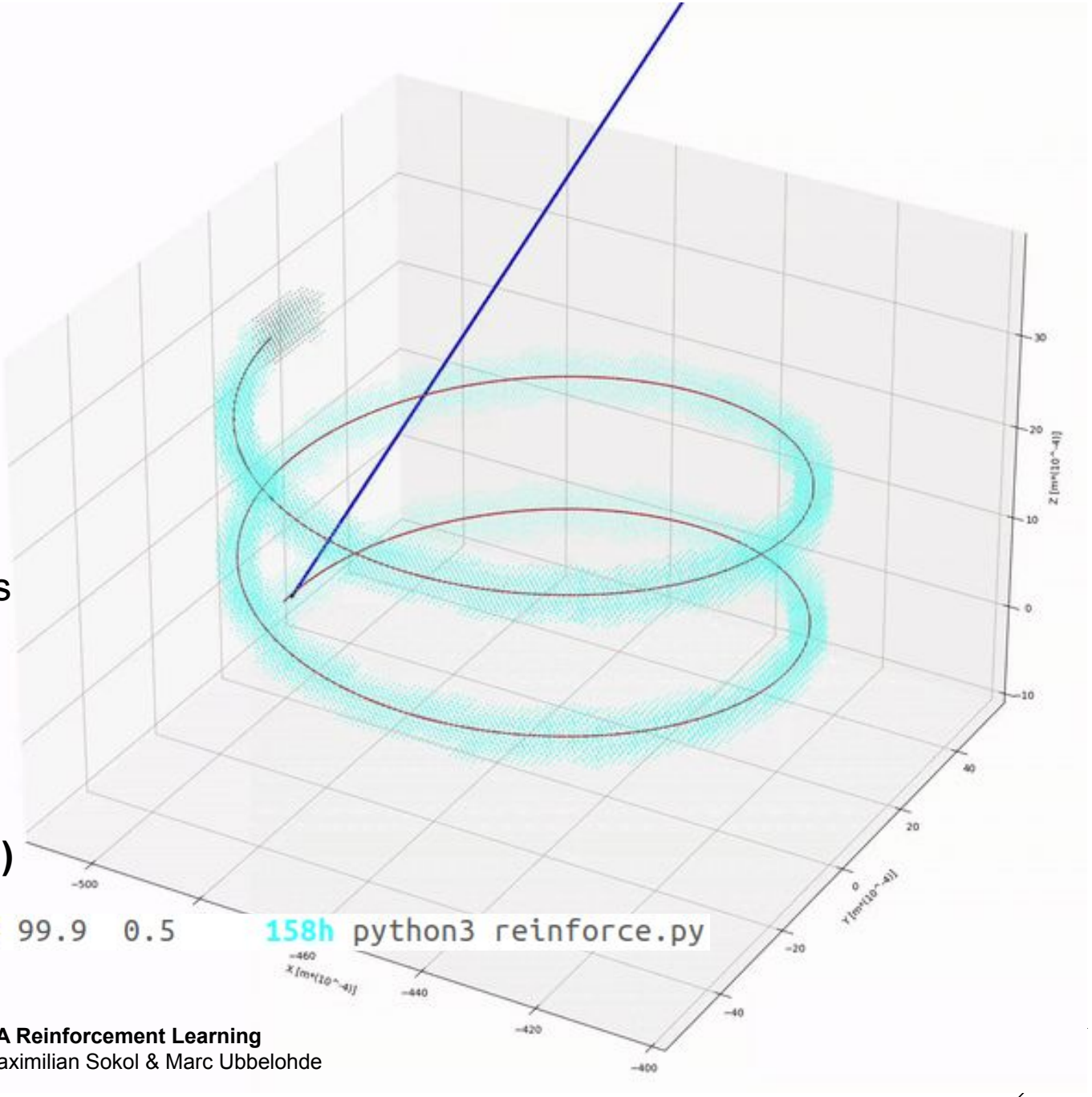
- **… after 5.5 hours of learning:**

This works!

- **Can we use this with the 6-Axis robot?**
- Forward kinematics for 6-Axis robot: 2 times as long
- Decision space: 3^3 times as large
- $5.5h \times 2 \times 3^3 = 297h = 12.375$ Days…

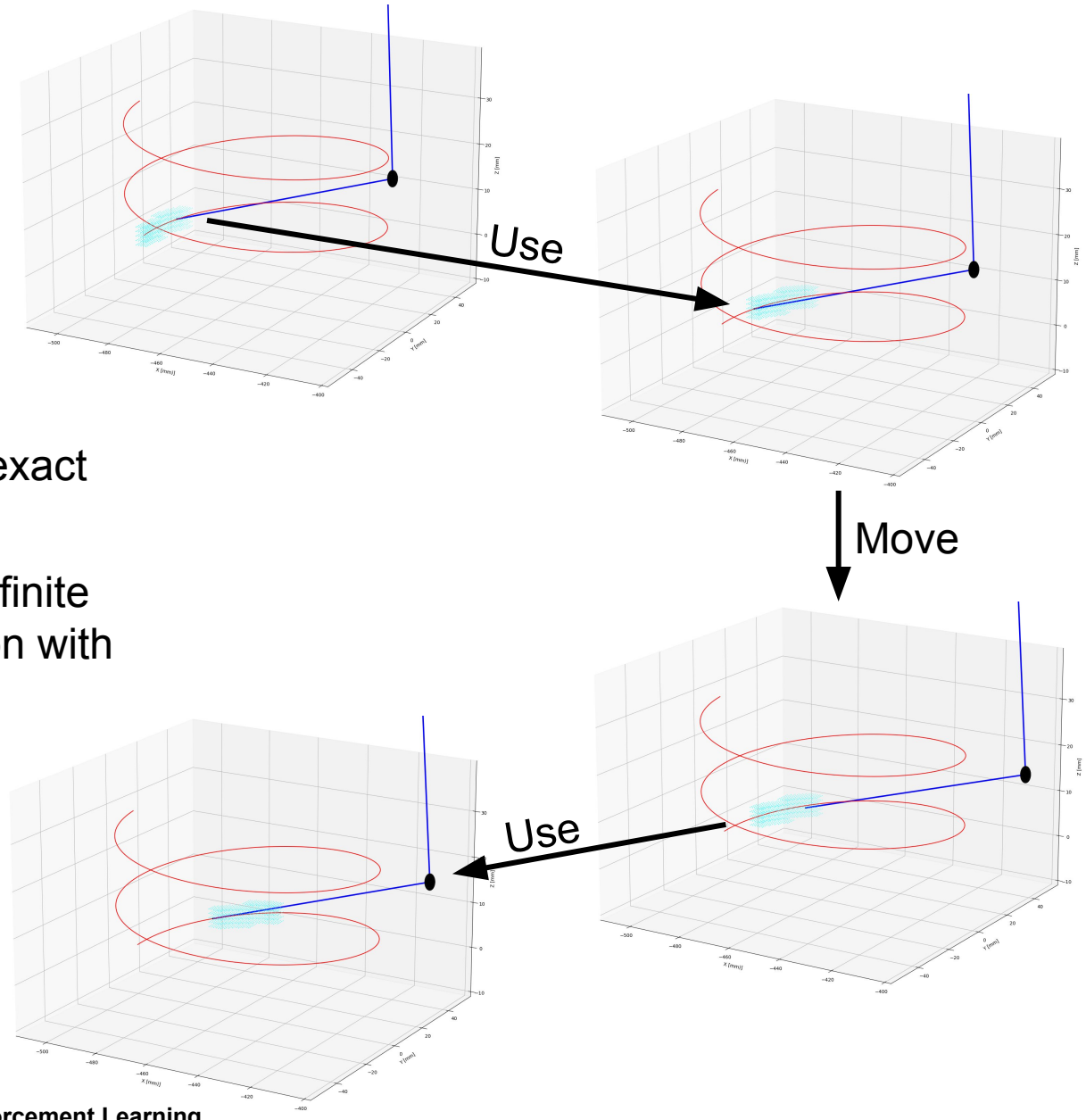
Let's say the computer is still calculating :)

```
235369 mle-admin  20   0 1170M  162M 22464 R 99.9  0.5     158h python3 reinforce.py
```



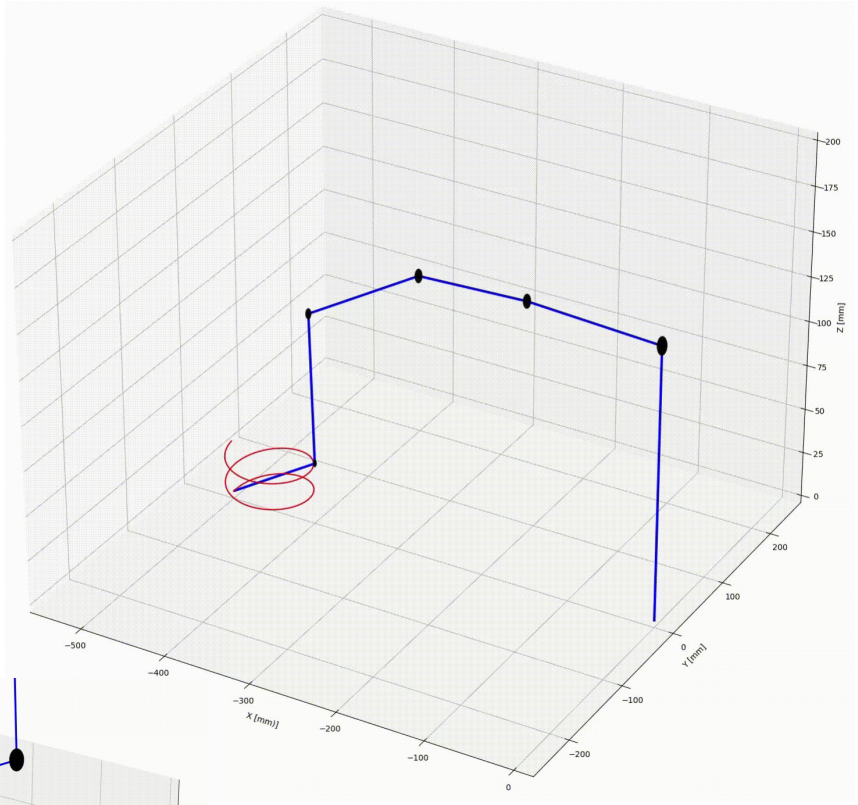
5th Approach

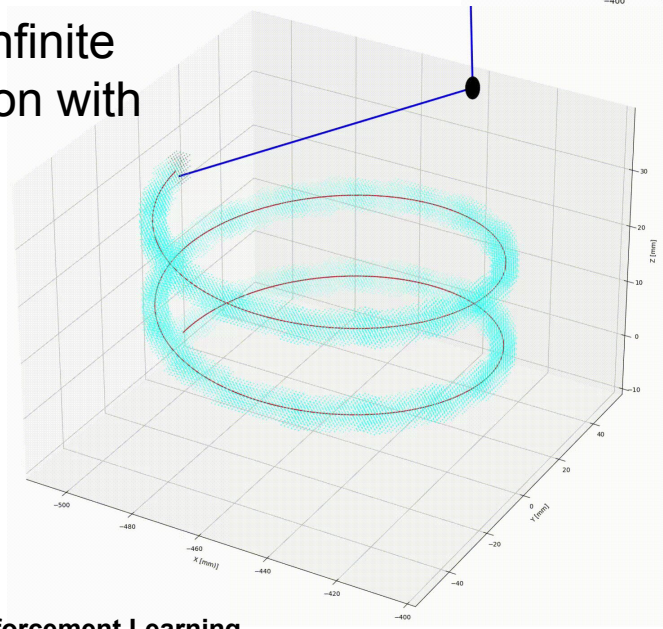
- **Go back to sectioning and parallelization**
- Learn 32 Sections in parallel
- Move the robot through each section
- Set starting angles of a following section to the exact finishing angles of the section before
- If Q values result in going out of bounds or an infinite loop, do 20 more training episodes of that section with the new starting point
- Repeat until all sections are traversed
- Stitch policies together



5th Approach

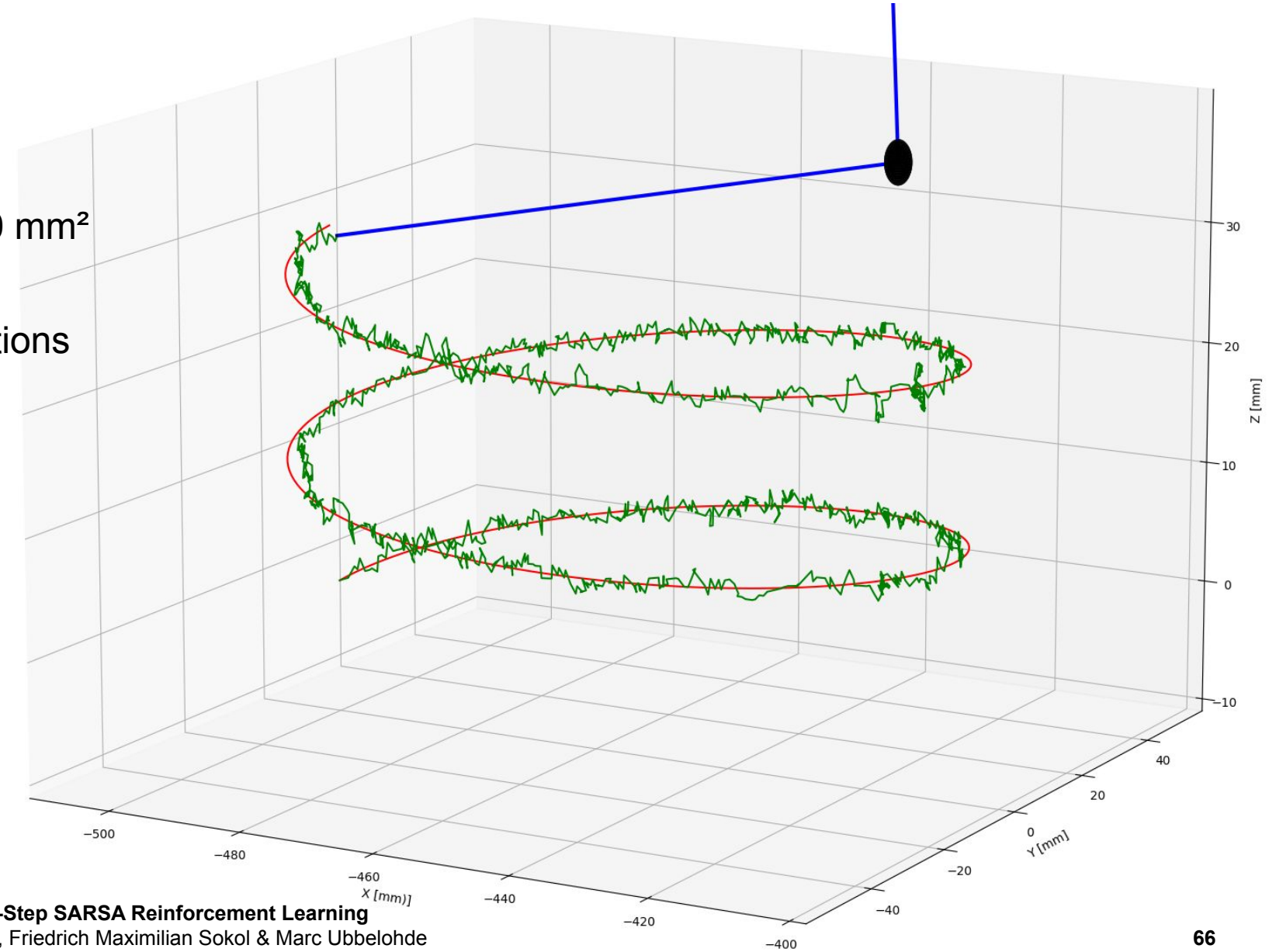
- **Go back to sectioning and parallelization**
- Learn 32 Sections in parallel
- Move the robot through each section
- Set starting angles of a following section to the exact finishing angles of the section before
- If Q values result in going out of bounds or an infinite loop, do 20 more training episodes of that section with the new starting point
- Repeat until all sections are traversed
- Stitch policies together
- let robot traverse section by section





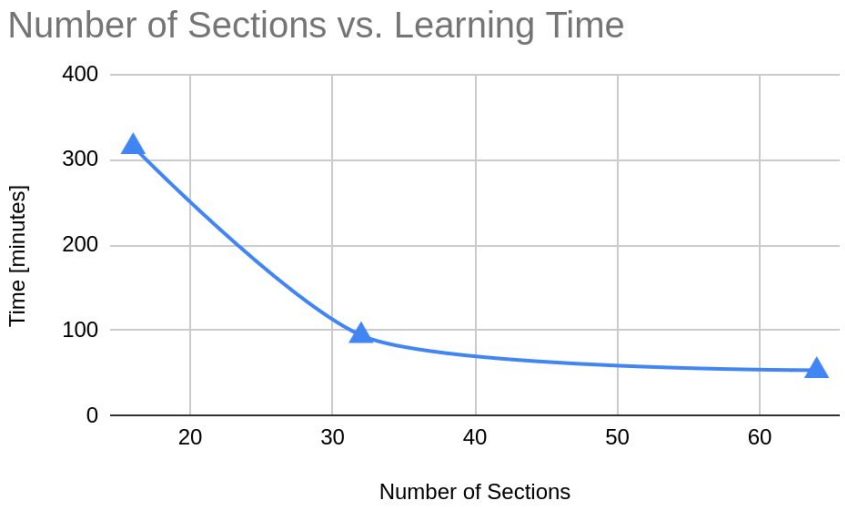
Result

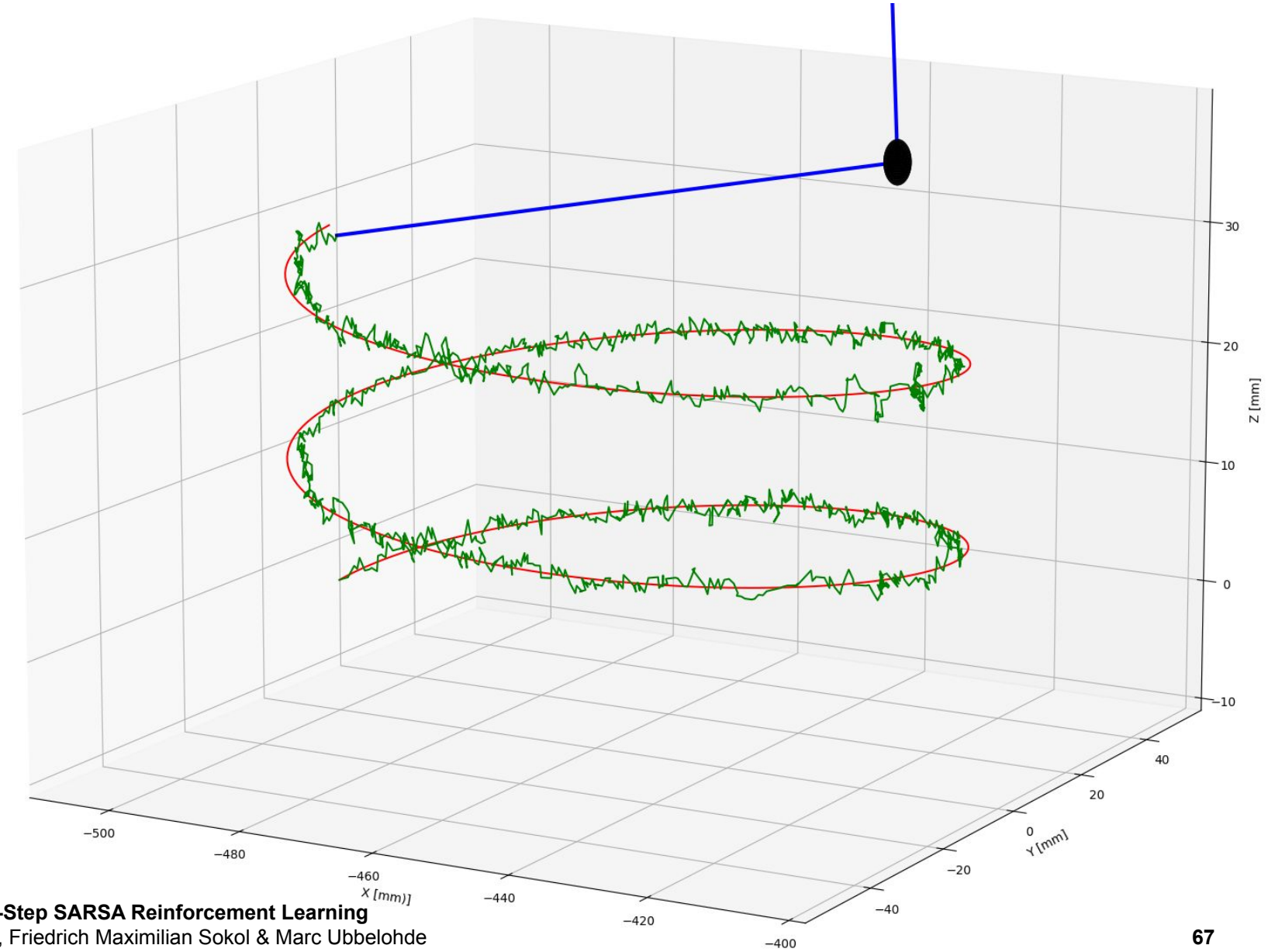
- MSE is between 42.38 mm² and 260 mm² over a total length of 377.52 mm
- time to learn ≈ 53 minutes at 64 sections
  - i9 12900k and 32 GBytes of RAM



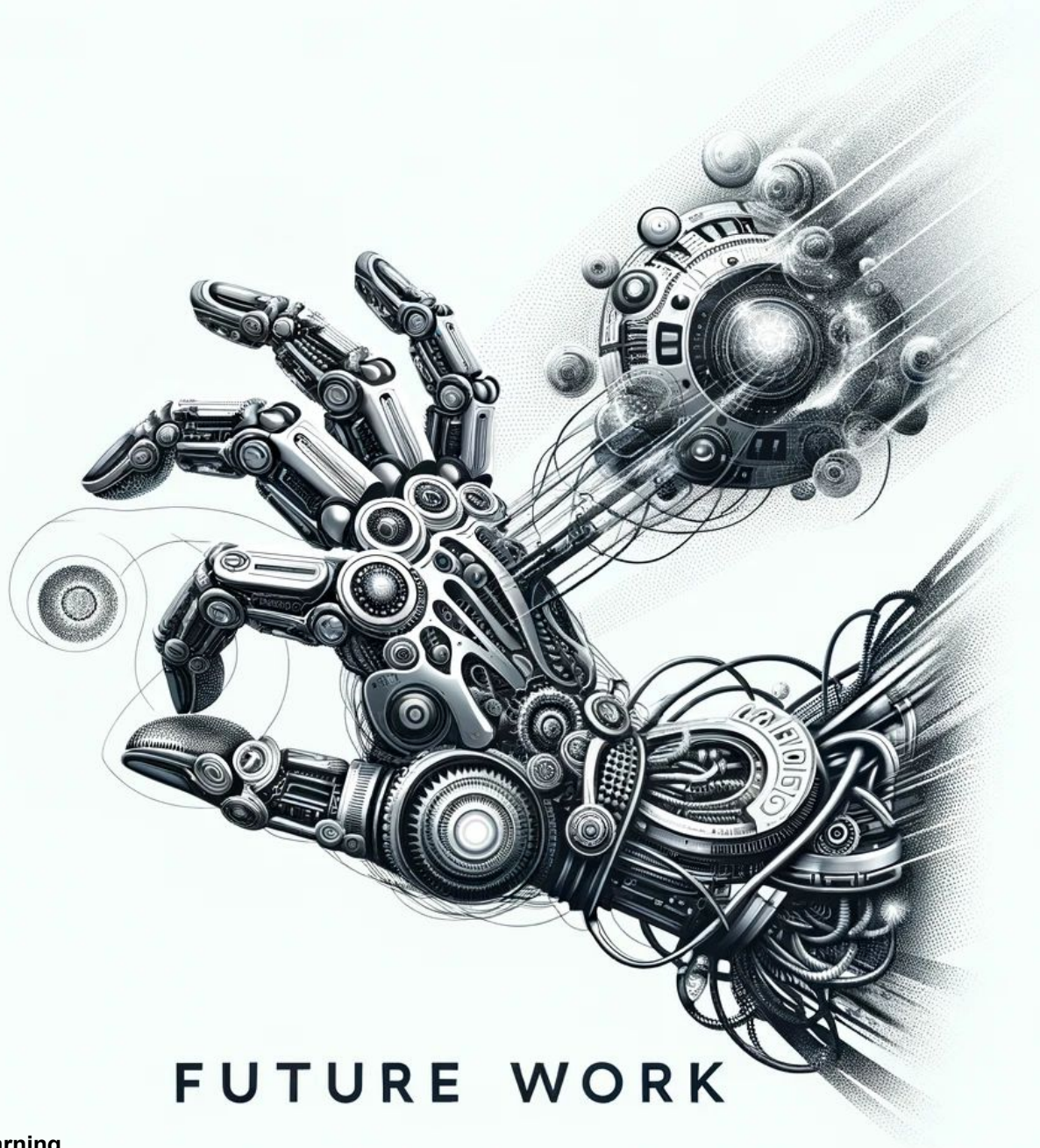
Result

Num Sections	Time (Minutes)	# Steps to traverse helix
16	315	1614
32	94	1511
64	53	2043





- Increase efficiency of the environment
  - Probably a lot of improvements can be done here
- Experiment with more reward structures
  - Reward structure is crucial in Reinforcement Learning (RL) Efficiency
- Remove Voxelization of space
  - Voxelization tends to increase MSE
  - Use a different RL method like Deep Reinforcement Learning to get rid of Voxelization



Source: Dall-E

Questions?

Source: Dall-E