Autonomous Drone Racing Optimization vs. RL

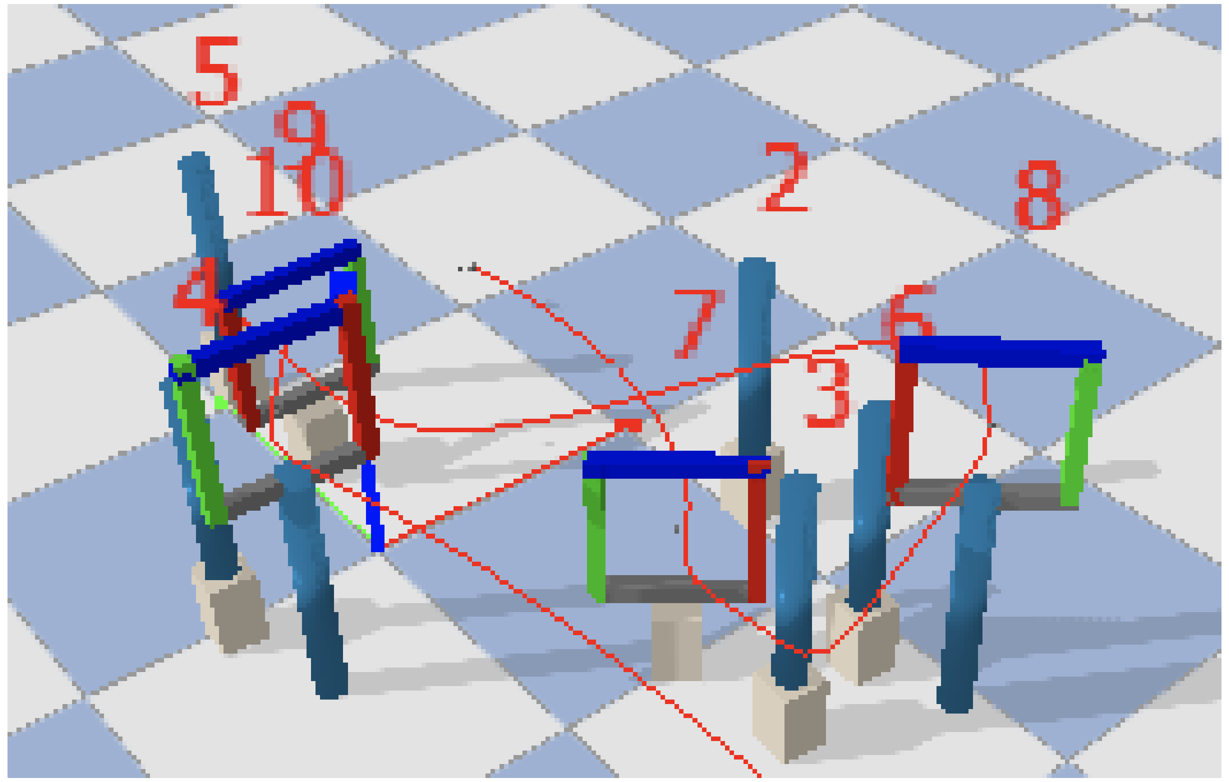
Autonomous Drone Racing - Final Presentation
17.07.2024

Tim Lindenau

The Challenge – Autonomous Drone Racing In Uncertain Environments

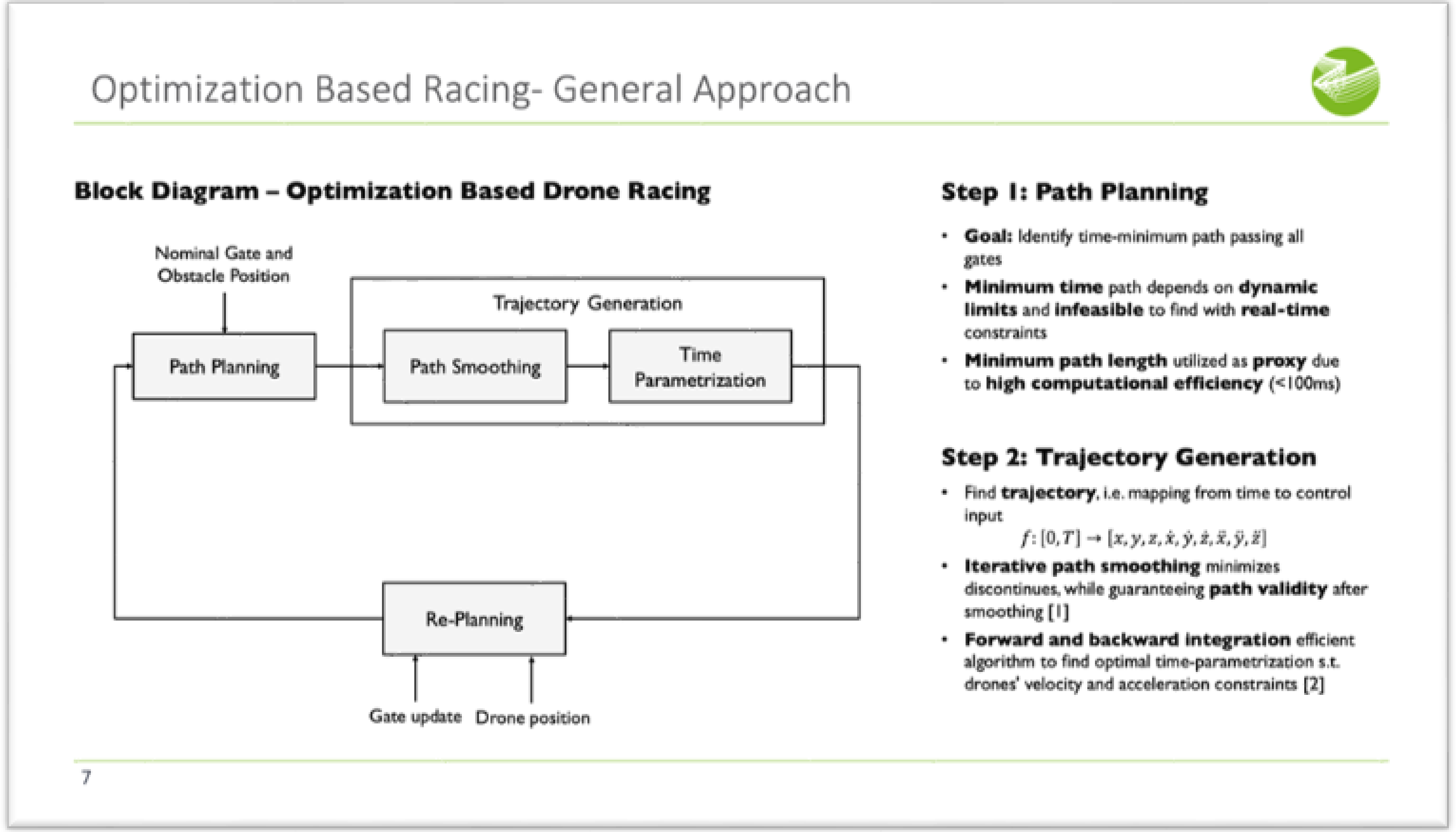
Goal: Developing high level controller to safely slalom through a set of gates in the shortest time possible

Challenge: Uncertainties in the position of gates and obstacles require within-fight updates as true gate position only becomes available shortly before passing

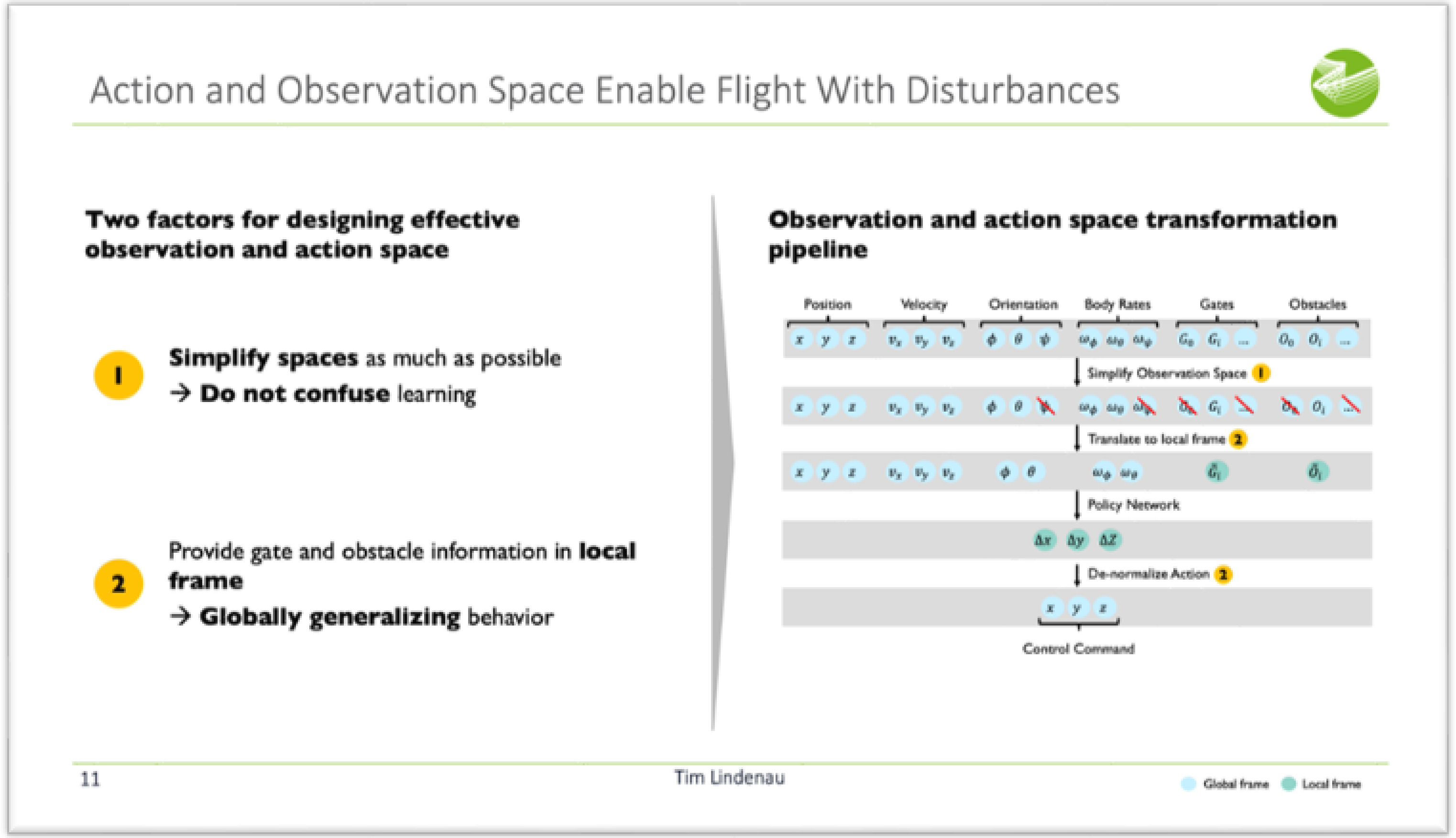


Two Possible Approaches: Optimization and Reinforcement Learning

Approach 1: Optimization

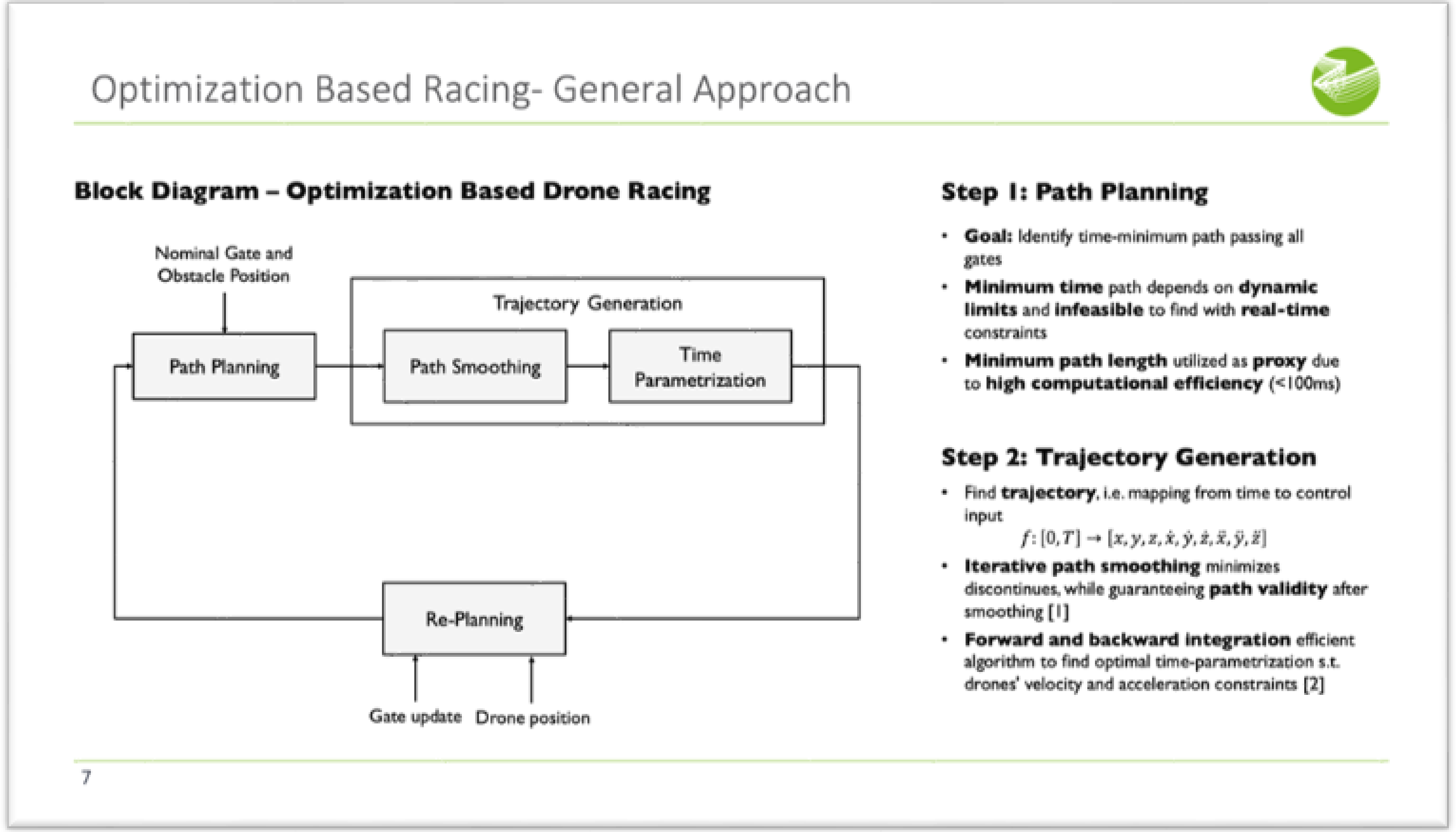


Approach 2: Reinforcement Learning

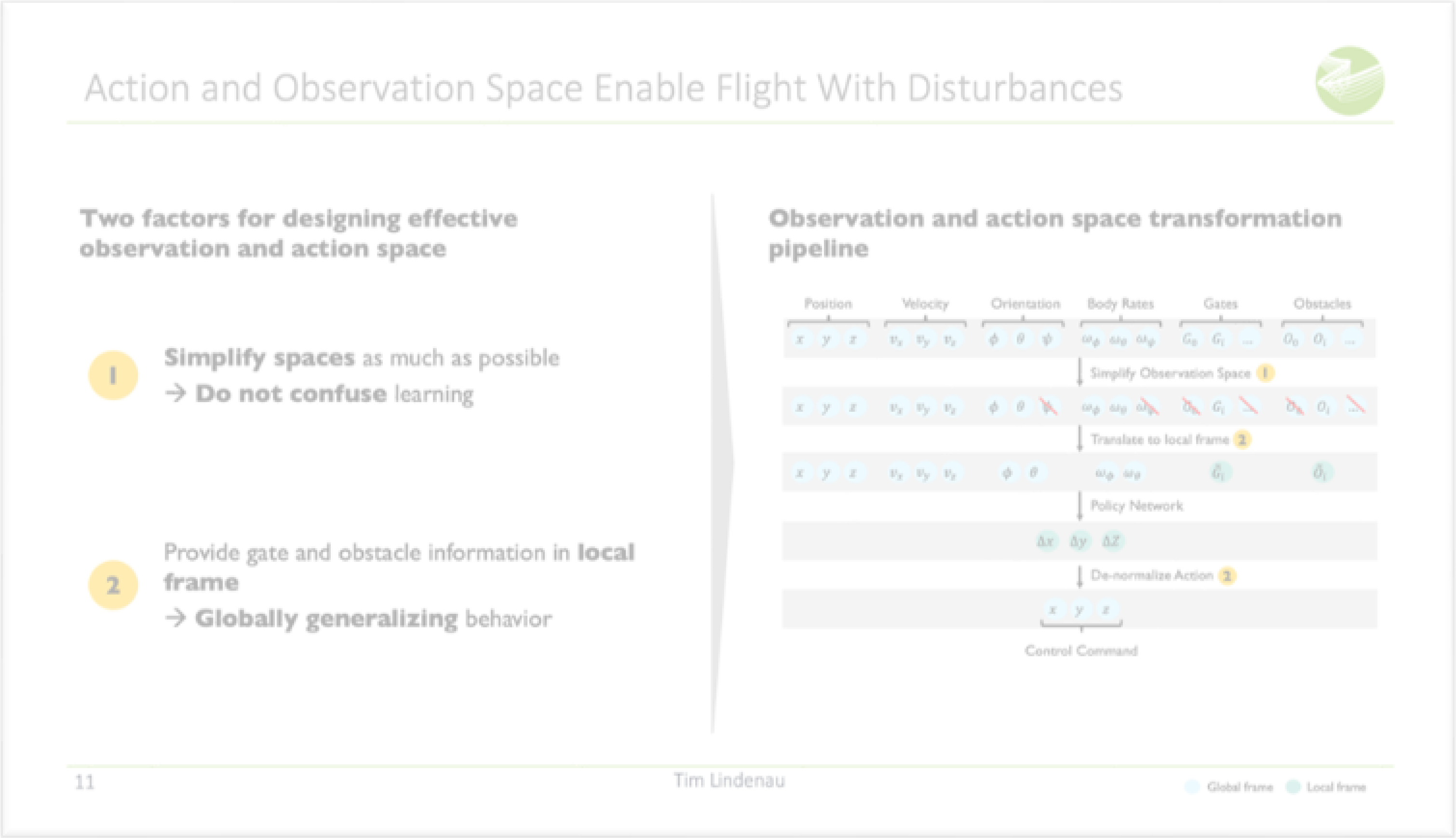


Two Possible Approaches: Optimization and Reinforcement Learning

Approach 1: Optimization

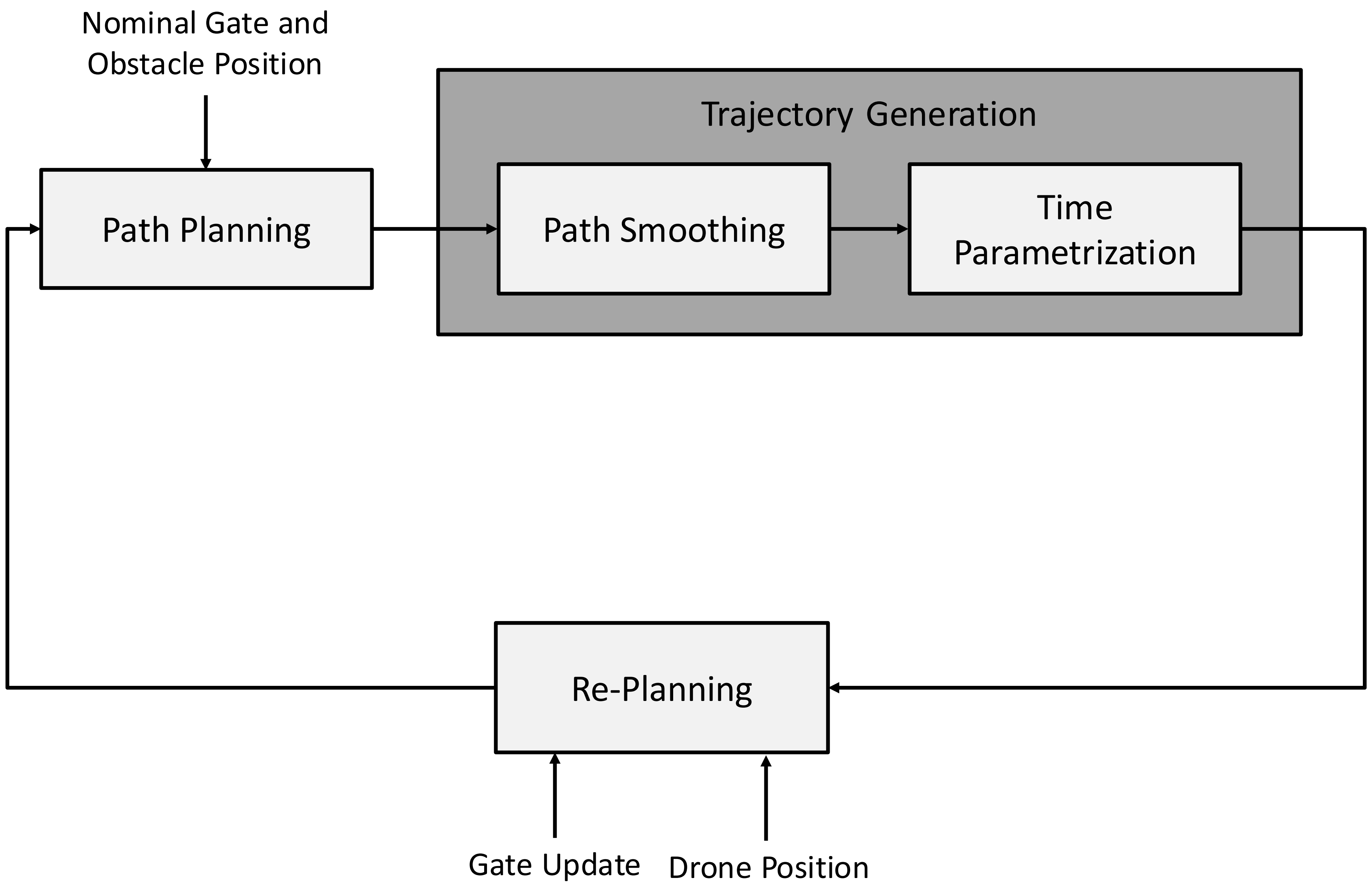


Approach 2: Reinforcement Learning



Optimization Based Racing- General Approach

Block diagram – Optimization based drone racing



Step 1: Path planning

- **Goal:** Identify time-minimum path
- **Minimum time** path depends on **dynamics** and **computationally expensive** to find
- **Minimum path length** utilized as **proxy** due to **high computational efficiency** (<100ms)

Step 2: Trajectory generation

- Find **trajectory**
  $$f: [0, T] \rightarrow [x, y, z, \dot{x}, \dot{y}, \dot{z}, \ddot{x}, \ddot{y}, \ddot{z}]$$
- **Iterative path smoothing** minimizes discontinues, while guaranteeing **path validity** [1]
- **Forward and backward integration**[1] to find **optimal time-parametrization** s.t. drones' velocity and acceleration constraints [2]

Step 3: Re-planning

- Re-plan **within flight** to adapt to updates
- Currently **only updating gates** not obstacles

[1] See https://www.youtube.com/watch?v=WR2oPxJi-H0&t=115s for introduction

Challenges of Within Flight Re-Planning

Re-planning: The challenge

1. **Computational efficient** trajectory generation pipeline, enables computation times of around **100ms**

2. **Too slow** to enable within-flight re-computations. **Drone moving** multiple centimeters **without control input**

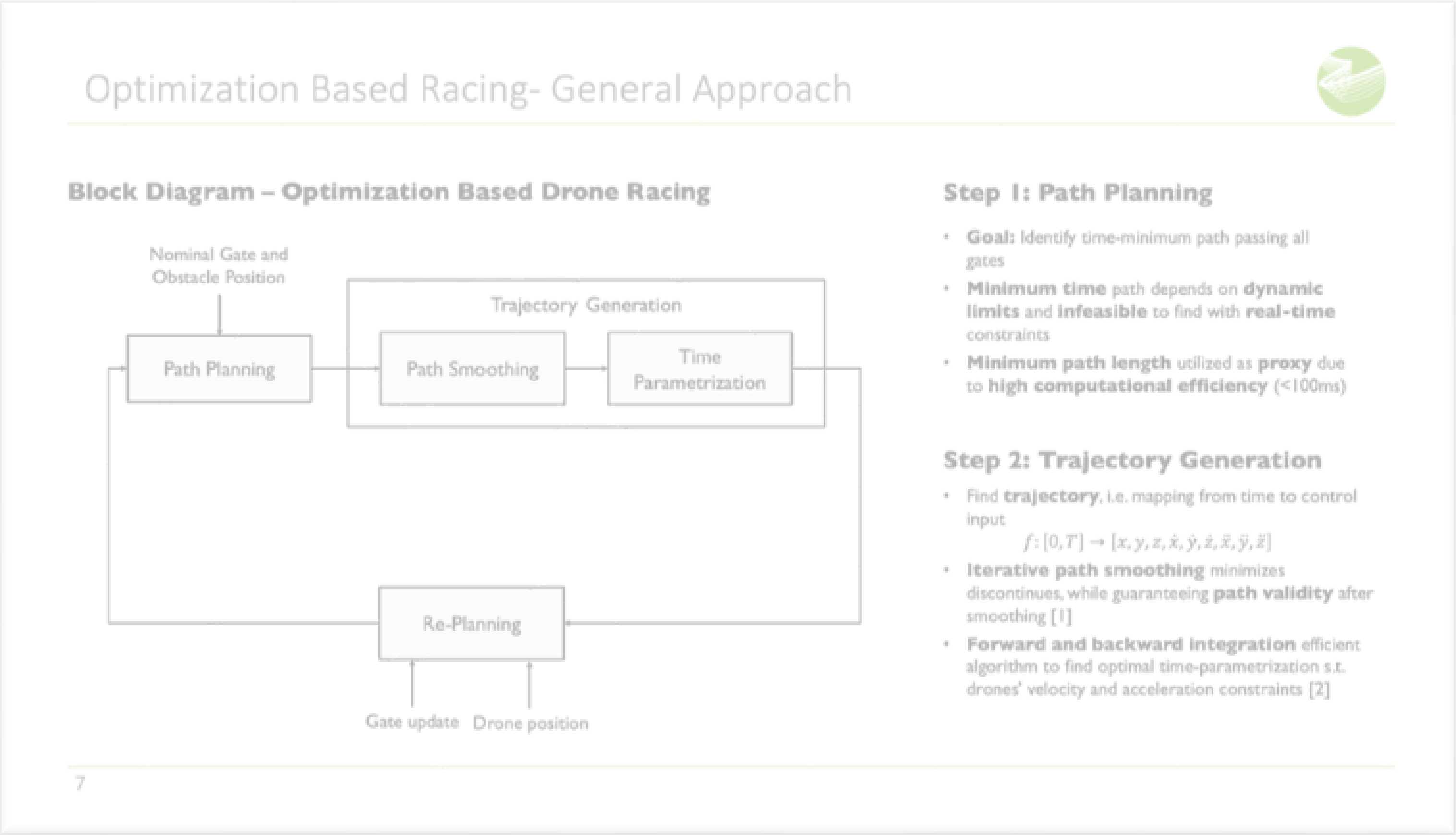
Non-blocking implementation and start-point offset enable re-planning without loss of control

1. **Non-blocking trajectory generation**
   - Trajectory computation in **extra thread** prevents **blocking** of **main control loop**
   - **Controller** continues based on **previous trajectory** until re-planning completed

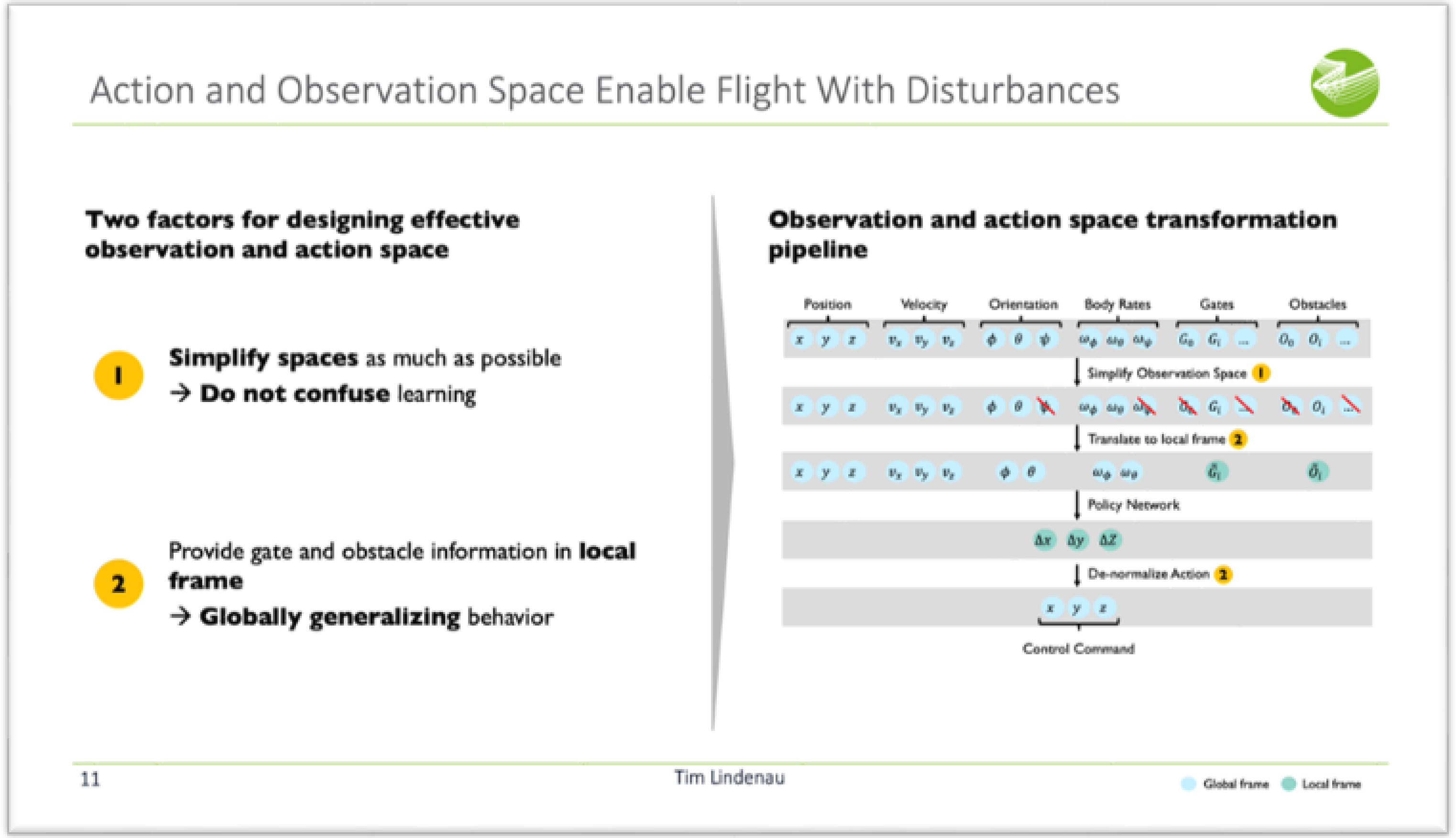
2. **Start-point offset**
   - **Start point offsetted** along trajectory, based on expected computation time
   - Trajectories **smoothly merged** after re-planning

Two Possible Approaches: Optimization and Reinforcement Learning

Approach 1: Optimization



Approach 2: Reinforcement Learning



Reinforcement Learning Motivation & Challenge

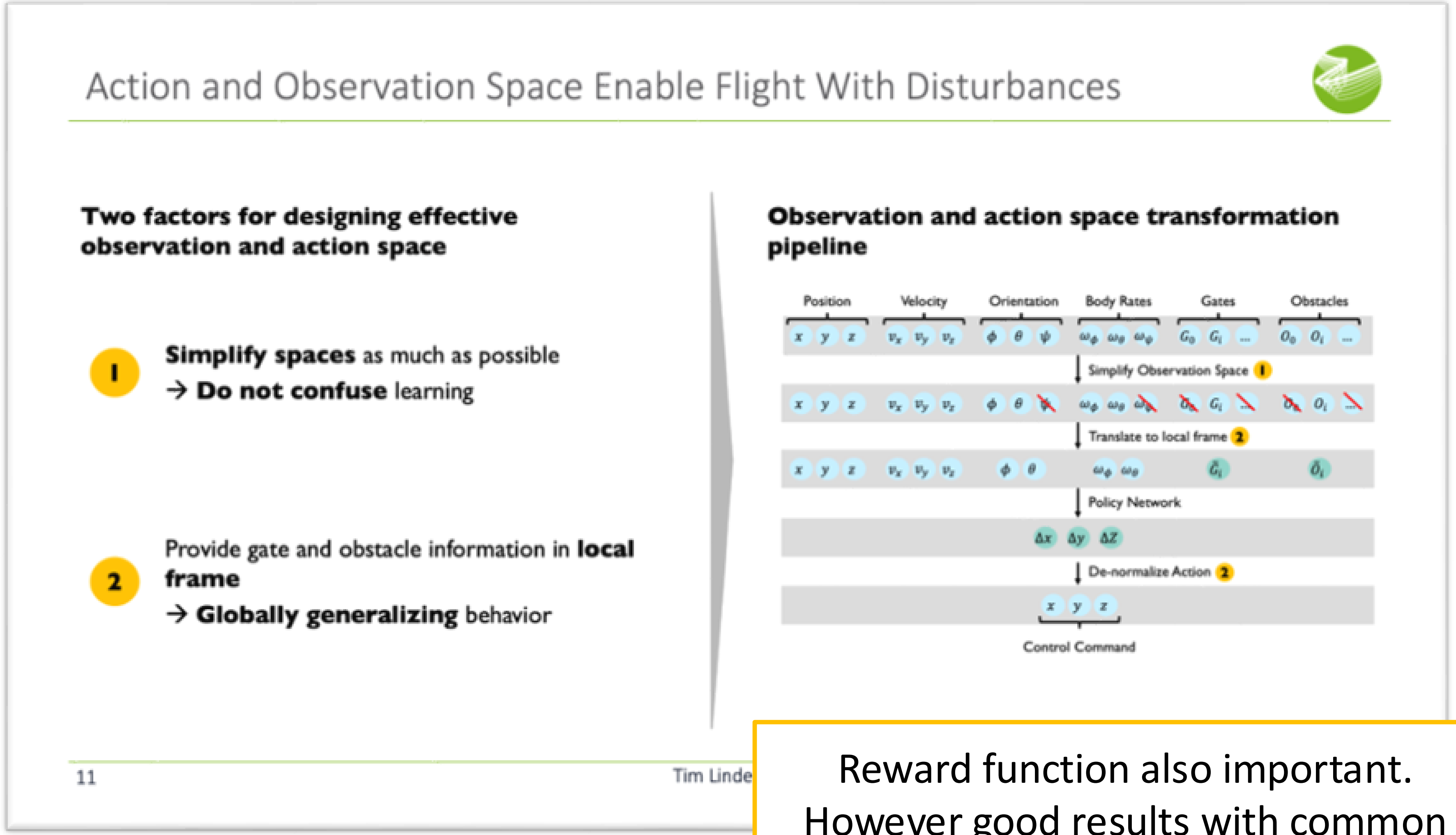
Reinforcement learning: Motivation & challenge

1. **Optimization complex** and with **tradeoffs** in flight time to enable re-planning capabilities

2. **Reinforcement learning outperformed** optimization in previous work [3, 4]

3. **Weak generalization** capabilities, potentially **hinder adapting** to **changing gate positions**

Generalization to disturbances enabled by well designed action, -and observation space



Reward function also important. However good results with common per time-step progress

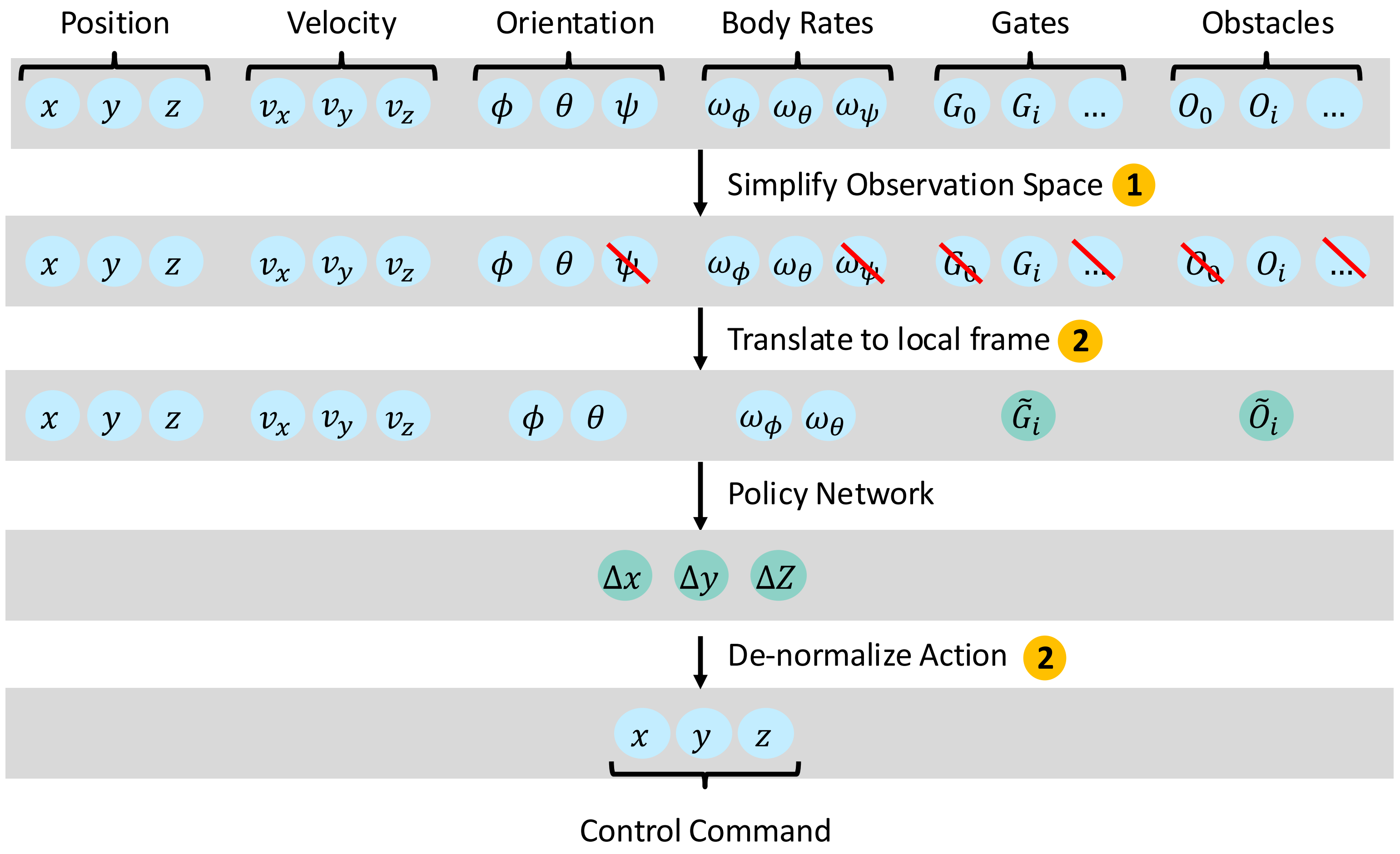
Well-tuned Action and Observation Space Enable Safe Flight With Disturbances

Two factors for designing effective observation and action space

1. **Simplify spaces** as much as possible
→ **Do not confuse** learning

2. Provide gate and obstacle information in **local frame**
→ **Globally generalizing** behavior

Observation and action space transformation pipeline



Global frame Local frame

Experimental Results (Simulation)

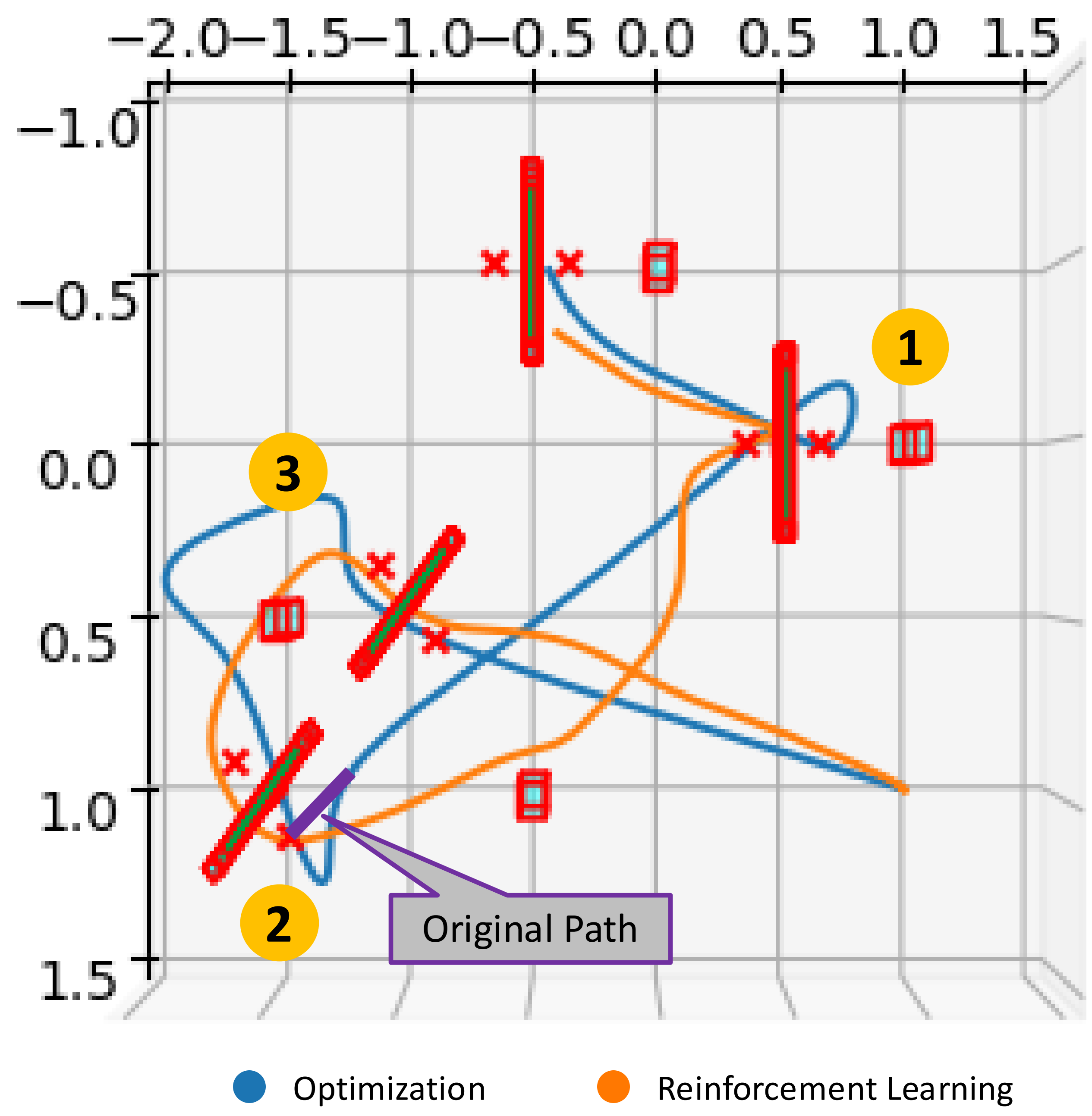
Result collection – Average lap times for different approaches
[values collected over 100 runs]

Algorithm	Difficulty	Mean Time (s)	Success Rate (%)
Baseline	Static	12.47	100
Baseline	Disturbed	12.47	18
Optimization	Static	6.50	86
RL	Static	**5.75**	**100**
Optimization	Disturbed	9.78	**81**
RL	Disturbed	**6.49**	47

1. **Optimization** and **RL, twice as fast** as baseline in static environment without disturbances
2. **Baseline** without re-planning **fails** in **disturbed environment**, only reaching success rate of 18%
3. **Optimization most robust** in disturbed environment, however **tradeoff in lap time** required to reach robustness
4. 50% success rate in disturbed environments indicates **RL agent learning** to **react** to **shifted gates**

Three Factors Enable RL to Outperform Optimization in Lap-Time

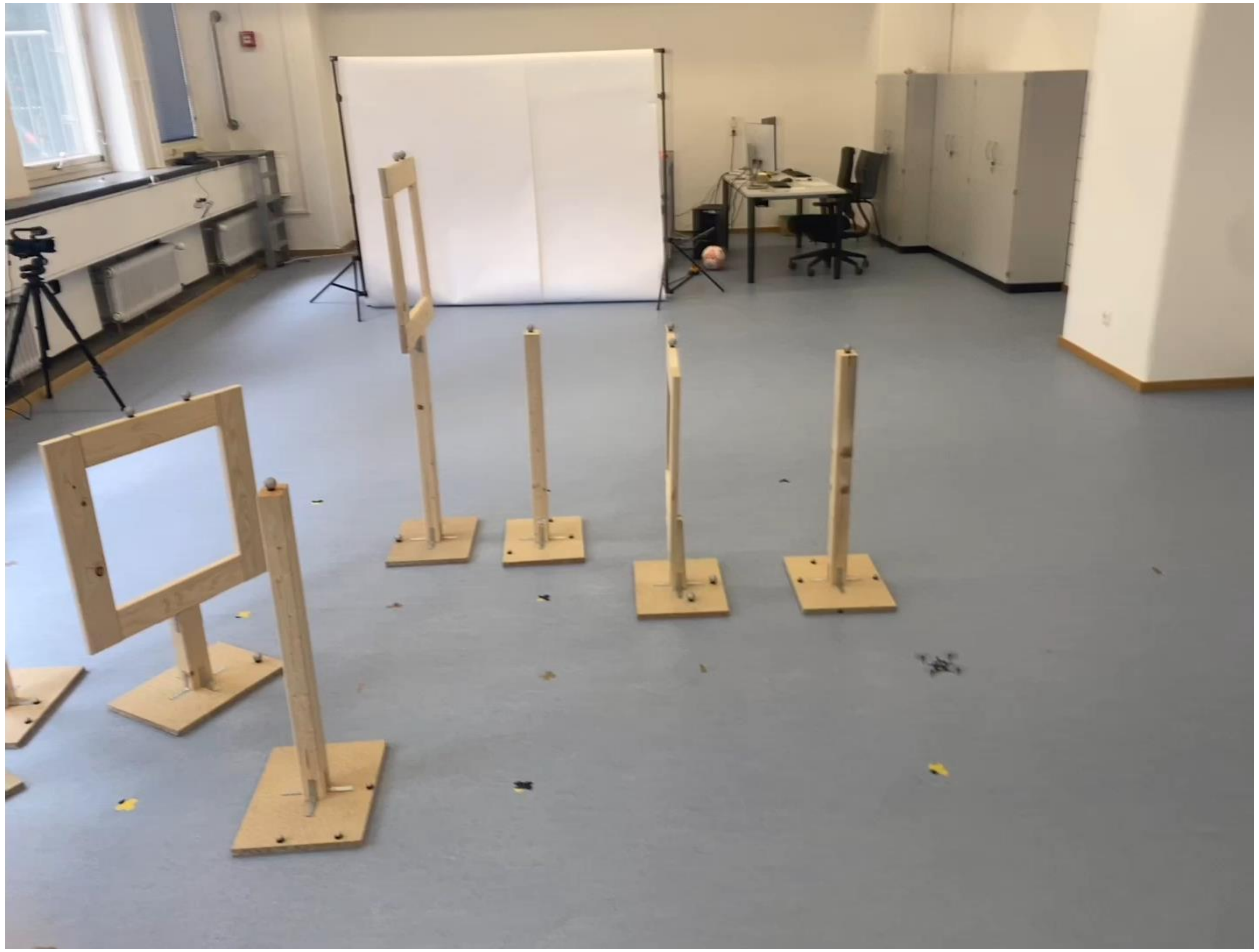
Flight trajectories: Optimization (blue) vs. Reinforcement Learning (orange)



Three factors explain how the RL agent can outperform optimization in lap-time

1. **Missing controllability** due to onboard controller indirection cause **overshoots**
2. **Shortest path length** criterion introduces sharp turns **reducing speed** and **amplifying overshoots**
3. **Padding around obstacles** required to guarantee robustness **increases path length**

Sim2Real Transfer – Optimization Based Drone Control



1 **Optimization** approach **successfully** transferred to **real world** (video)

2 **No time** to **transfer RL** approach. Experiences of other groups indicate **challenges**

Milestones and Key Results

Milestones

Initial Goals

1. Implement **optimization-based** drone racing with **re-plan** capabilities
2. Apply algorithm to **real-drone outperforming** original **baseline**
3. Develop **fast RL agent** for **static environment**
4. Develop **fast RL agent** for **disturbed environment**
5. **Sim2Real** transfer **RL** agent

Key Results

- **Achieved** optimization-based drone racing with re-planning by focusing on **computational efficient multithreaded C++** implementation
- Reached **fast lap times in static** environment and **high robustness** to disturbances
- Future work: Include **obstacle updates**

- **Solved static environment** with 100% success rates and fast lap times
- **Tuning observation and action space** enabled flying in **dynamic environment**
- Future work: Further **improve robustness** and **Sim2Real** transfer

Achieved Partially Achieved Not Achieved

Sources

1. "Ompl::Geometric::RRTstar Class Reference." Accessed July 6, 2024. https://ompl.kavrakilab.org/classompl_1_1geometric_1_1RRTstar.html.
2. Kunz, Tobias, and Mike Stilman. "Time-Optimal Trajectory Generation for Path Following with Bounded Acceleration and Velocity." In *Robotics*, edited by Nicholas Roy, Paul Newman, and Siddhartha Srinivasa, 209–16. The MIT Press, 2013. https://doi.org/10.7551/mitpress/9816.003.0032.
3. "Champion-Level Drone Racing Using Deep Reinforcement Learning | Nature." Accessed April 23, 2024. https://www.nature.com/articles/s41586-023-06419-4.
4. Song, Yunlong, Mats Steinweg, Elia Kaufmann, and Davide Scaramuzza. "Autonomous Drone Racing with Deep Reinforcement Learning." In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1205–12. Prague, Czech Republic: IEEE, 2021. https://doi.org/10.1109/IROS51168.2021.9636053.

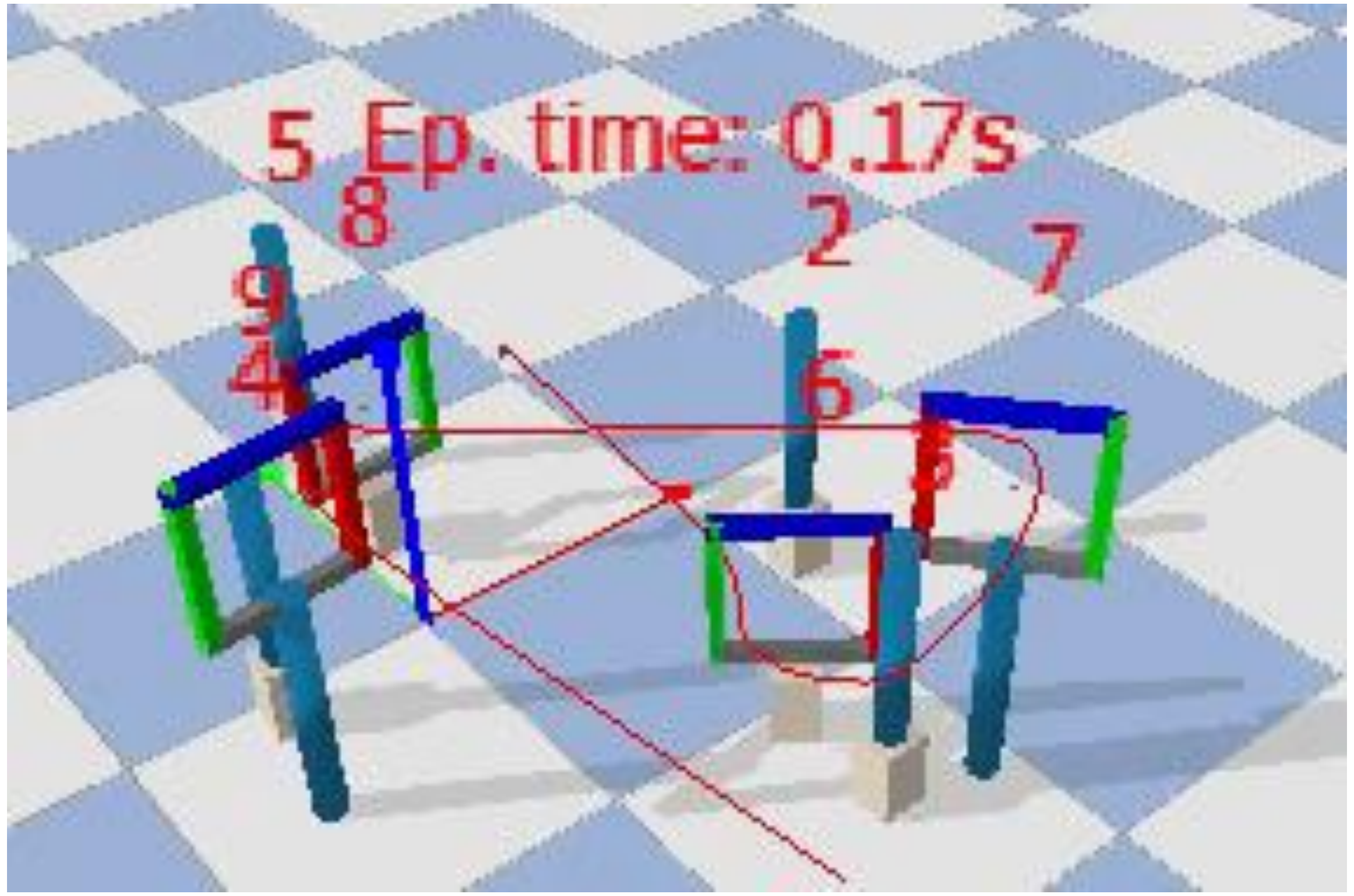
Appendix

Autonomous Drone Racing - Final Presentation
17.07.2024

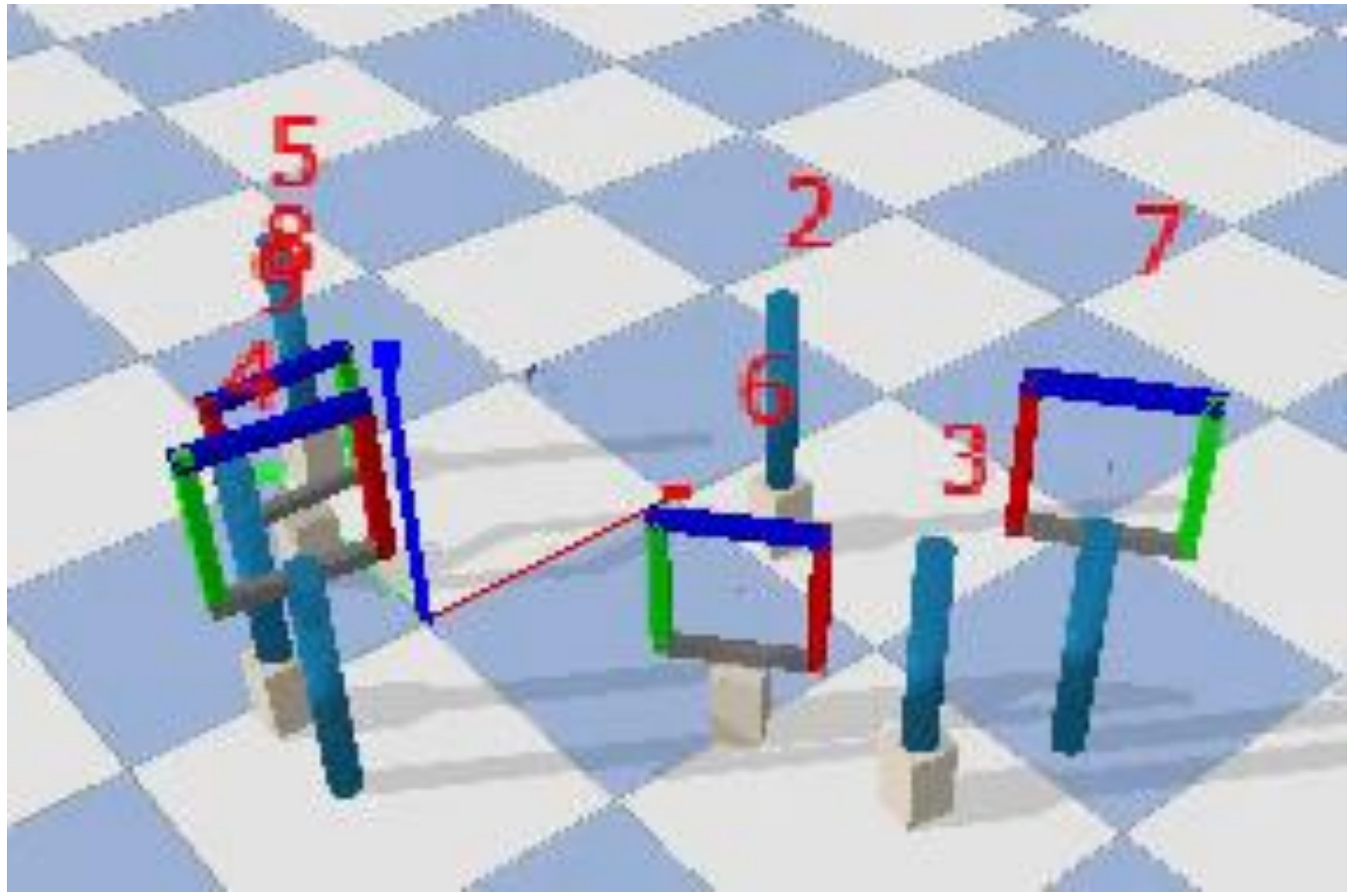
Tim Lindenau

Visualization in Simulation (With Disturbances)

Approach 1: Optimization



Approach 2: Reinforcement Learning



Reinforcement Learning: Reward Function

Key Requirements For Effective Reward Function

1. **Alignment** with goal of **drone racing**, i.e., finishing track in minimum time

2. **Dense supervision** at each time-step to simplify information gain of learning algorithm

Path progress provides dense proxy reward for minimizing lap times

- **Lap time reward** immediately rewards fast lap time, however reward sparse only at end of successful episodes
- **Per-time-step progress** towards next goal, good **proxy** reward, **providing dense supervision** at each time step

$$r_{prog}(t) = \|\boldsymbol{x}(t-1) - \boldsymbol{g}\|_2 - \|\boldsymbol{x}(t) - \boldsymbol{g}\|_2$$

- **Total reward function** completed with binary gate-pass, and collision reward

$$r(t) = \lambda_{prog} r_{prog}(t) + \lambda_{gate}\, r_{gate}(t) + \lambda_{col}\, r_{col}(t)$$