A Multi-Pronged Approach to Computational Challenges in HJ Reachability

Mo Chen | mochen@cs.sfu.ca

Multi-Agent Robotic Systems Lab | sfumars.com

School of Computing Science, Simon Fraser University

Introduction

- Assistant Professor, School of Computing Science, Simon Fraser University
  - SFU Multi-Agent Robotics Lab
  - SFU Visual Computing Group
- I love multidisciplinary research and collaborations
  - Postdoc in Stanford AA with Marco Pavone
  - PhD in Berkeley EECS with Claire Tomlin
  - Bachelors in UBC Engineering Physics

- Dimensionality reduction
  - System decomposition

- Optimized software
  - Memory locality and parallelism

Outline

- The challenge: curse of dimensionality
  - Finite difference: a numerically convergent approach
- Dimensionality reduction
  - “Chained” systems and projections
- Optimizing software
  - Maximizing brute-force computation capabilities
- Novel applications of HJ PDE solutions in machine learning

Outline

- **The challenge: curse of dimensionality**
  - **Finite difference: a numerically convergent approach**
- Dimensionality reduction
  - “Chained” systems and projections
- Optimizing software
  - Maximizing brute-force computation capabilities
- Novel applications of HJ PDE solutions in machine learning

Challenges in Safety-Critical Systems

- Account for **all** possible system behaviors
  - Formal verification is needed
- Complex environment
  - Weather conditions
  - Unpredictable
  - Adversarial agents
- Complex dynamics
- **High-dimensional system dynamics**

The Hamilton-Jacobi PDE

- One version of the equation

$$\frac{\partial V}{\partial t} + \max_{u} \min_{d} \left[\left(\frac{\partial V}{\partial x} \right)^{\mathsf{T}} f(x, u, d) \right] = 0, \qquad t \leq 0, \qquad V(0, x) = l(x)$$

- Solved on a grid

- Numerically convergent
  - As state space and time discretization go to zero, we approach the exact solution

Advantages

- Globally optimal solution

- Flexibility: variants of value/cost functions

$$V(t,x(t)) = \min_{\Gamma[u](\cdot)} \max_{u(\cdot)} l(x(0)) \qquad \Rightarrow \frac{\partial V}{\partial t} + \max_{u} \min_{d} \left[\left(\frac{\partial V}{\partial x} \right)^{\top} f(x,u,d) \right] = 0$$

$$V(t,x(t)) = \min_{\Gamma[u](\cdot)} \max_{u(\cdot)} \min_{s \in [t,0]} l(s,x(s)) \qquad \Rightarrow \min \left\{ \frac{\partial V}{\partial t} + \max_{u} \min_{d} \left[\left(\frac{\partial V}{\partial x} \right)^{\top} f(x,u,d) \right], l(t,x) - V(t,x) \right\} = 0$$

  - Several other variants for different reachability problems

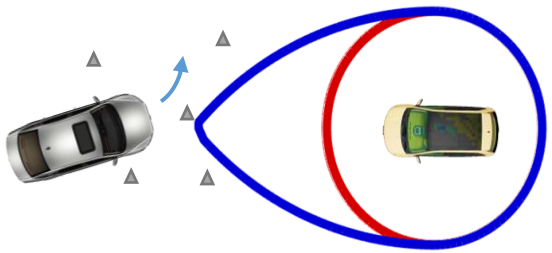
Advantages

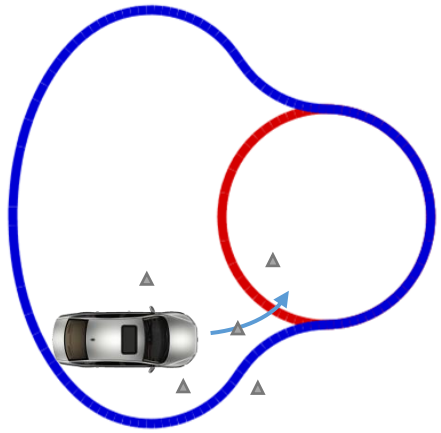
- Disturbances

$$V(t,x(t)) = \min_{\Gamma[u](\cdot)} \max_{u(\cdot)} l(x(T)) \qquad \Rightarrow \frac{\partial V}{\partial t} + \max_{u} \min_{d} \left[\left(\frac{\partial V}{\partial x} \right)^{\mathsf{T}} f(x,u,d) \right] = 0$$

- Nonlinear dynamics

- Can represent sets of arbitrary shapes





Challenges in Safety-Critical Systems

- Account for **all** possible system behaviors
  - Formal verification is needed

- Complex environment
  - Weather conditions
  - Unpredictable
  - Adversarial agents

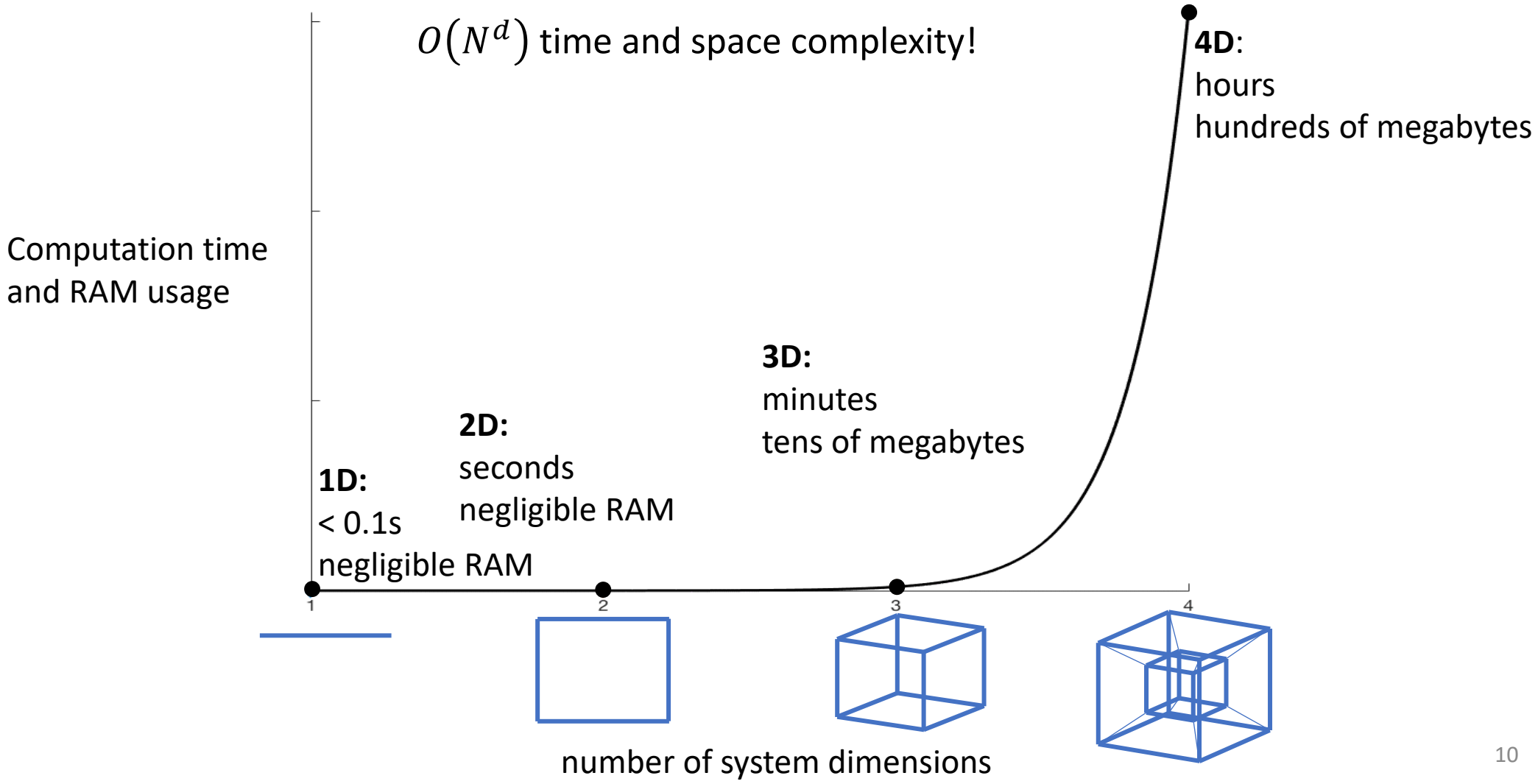
- Continuous-time system dynamics

- **High-dimensional system dynamics**

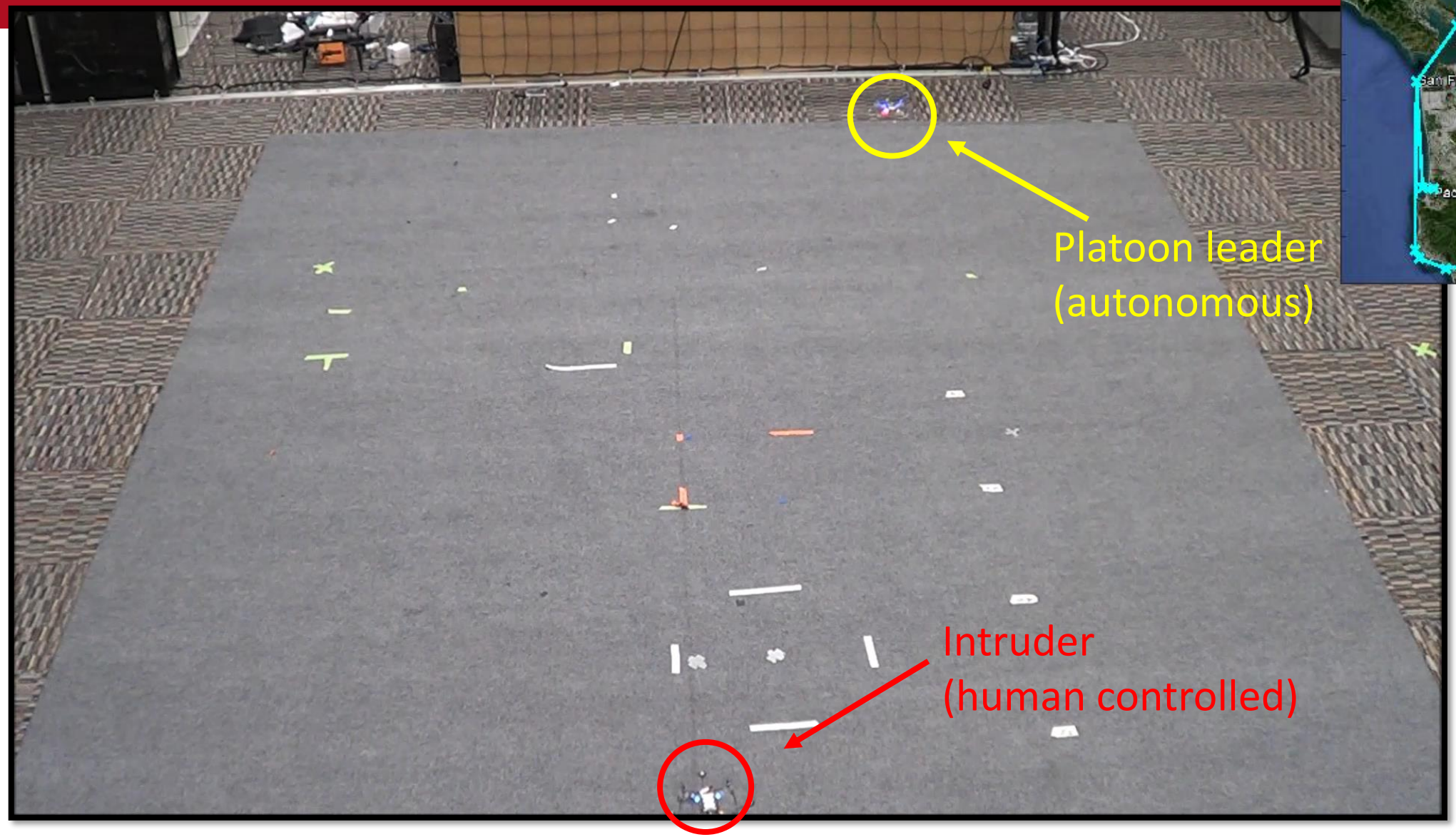
Main Challenge: Exponential Computational Complexity

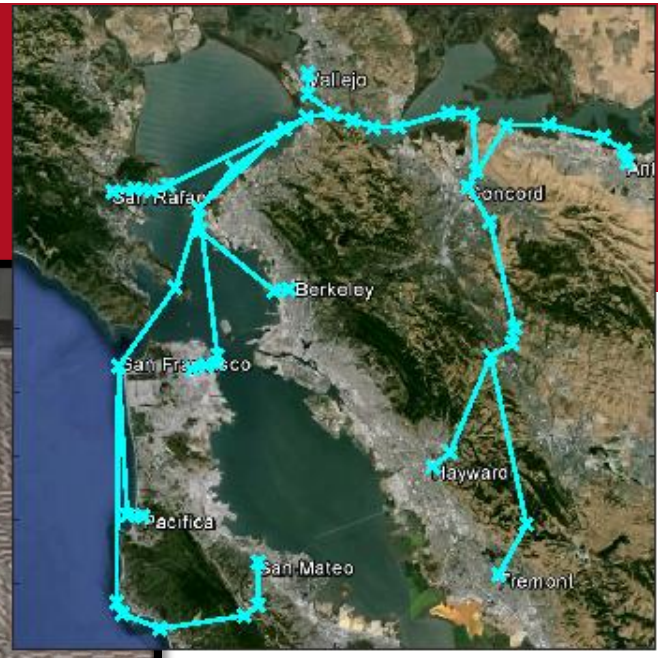
6D:
intractable!

5D:
days
gigabytes



Drone Intruder Avoidance





Chen et al., JGCD 2015

Car Experiment

- More complex model (7D)
  - Intractable → simplify model
  - Modelling errors resulted in positional errors of up to ~15cm

- Rate of change in turn rate and acceleration is important!

Leung et al., IJRR 2018

Outline

- The challenge: curse of dimensionality
  - Finite difference: a numerically convergent approach

- Dimensionality reduction
  - “Chained” systems and projections

- Optimizing software
  - Maximizing brute-force computation capabilities

- Novel applications of HJ PDE solutions in machine learning

Outline

- The challenge: curse of dimensionality
  - Finite difference: a numerically convergent approach

- **Dimensionality reduction**
  - **“Chained” systems and projections**

- Optimizing software
  - Maximizing brute-force computation capabilities

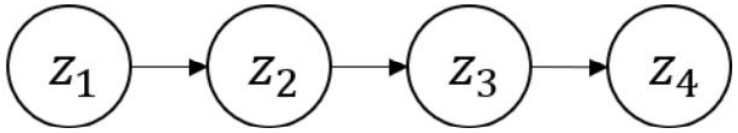
- Novel applications of HJ PDE solutions in machine learning

State Dependency-Based Decomposition

Example: 4D Quadruple Integrator

$$\dot{z}_1 = z_2 + d$$
$$\dot{z}_2 = z_3$$
$$\dot{z}_3 = z_4$$
$$\dot{z}_4 = u$$

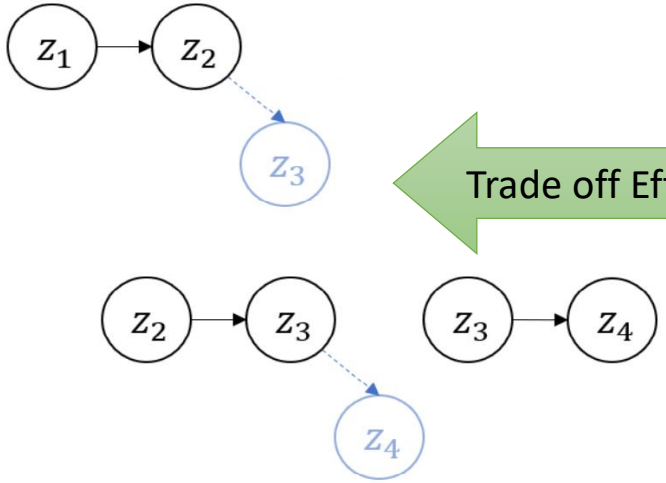
State Dependency Graph:



Li, Chen. ACC '20

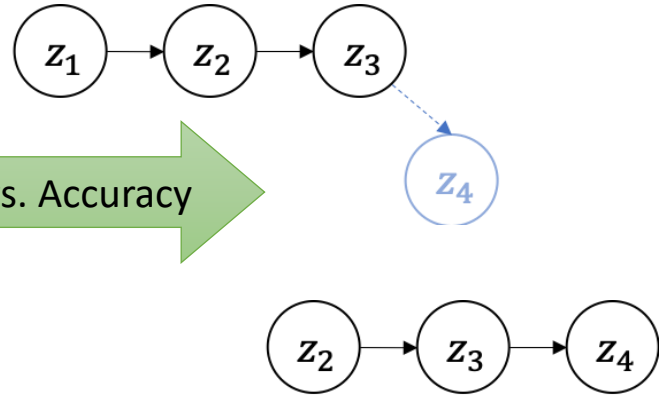
Option 1: Three 2D subsystems

$$S_1 : \dot{x}_1 = \begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \end{bmatrix} = \begin{bmatrix} z_2 + d \\ z_3 \end{bmatrix}$$
$$S_2 : \dot{x}_2 = \begin{bmatrix} \dot{z}_2 \\ \dot{z}_3 \end{bmatrix} = \begin{bmatrix} z_3 \\ z_4 \end{bmatrix}$$
$$S_3 : \dot{x}_3 = \begin{bmatrix} \dot{z}_3 \\ \dot{z}_4 \end{bmatrix} = \begin{bmatrix} z_4 \\ u \end{bmatrix}$$



Option 2: Two 3D subsystems

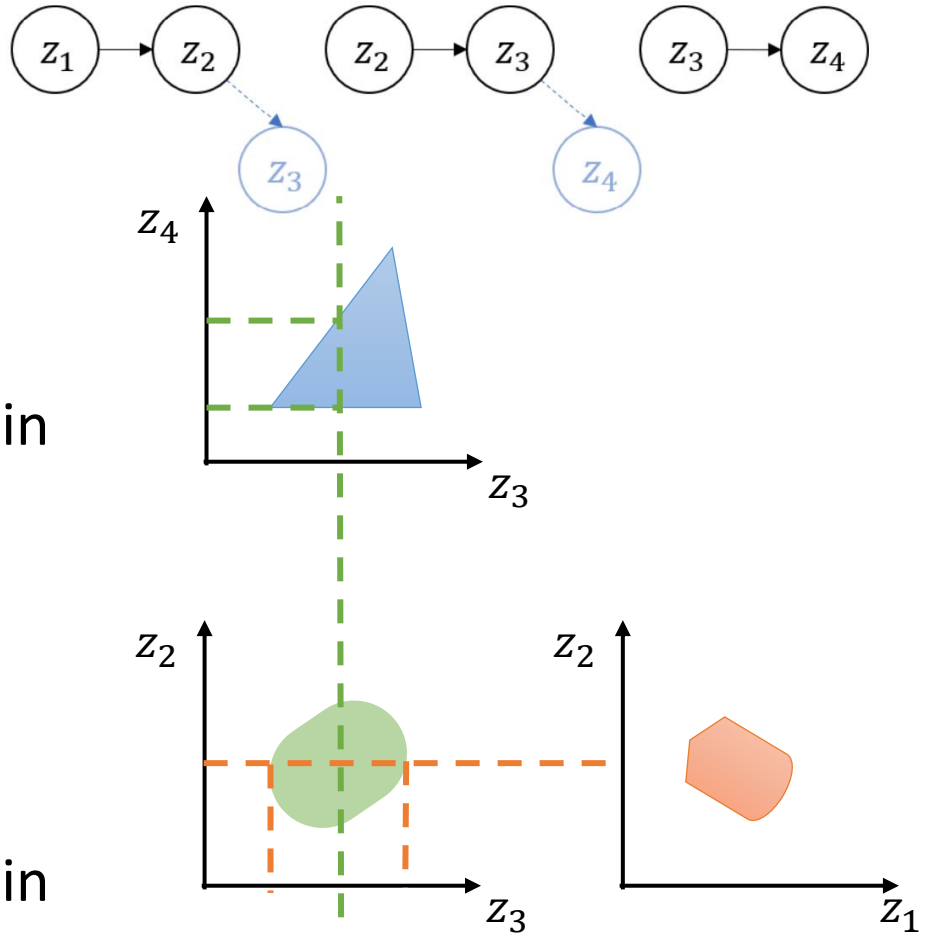
$$S_1 : \dot{x}_1 = \begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \\ \dot{z}_3 \end{bmatrix} = \begin{bmatrix} z_2 + d \\ z_3 \\ z_4 \end{bmatrix}$$
$$S_2 : \dot{x}_2 = \begin{bmatrix} \dot{z}_2 \\ \dot{z}_3 \\ \dot{z}_4 \end{bmatrix} = \begin{bmatrix} z_3 \\ z_4 \\ u \end{bmatrix}$$



Trade off Efficiency vs. Accuracy

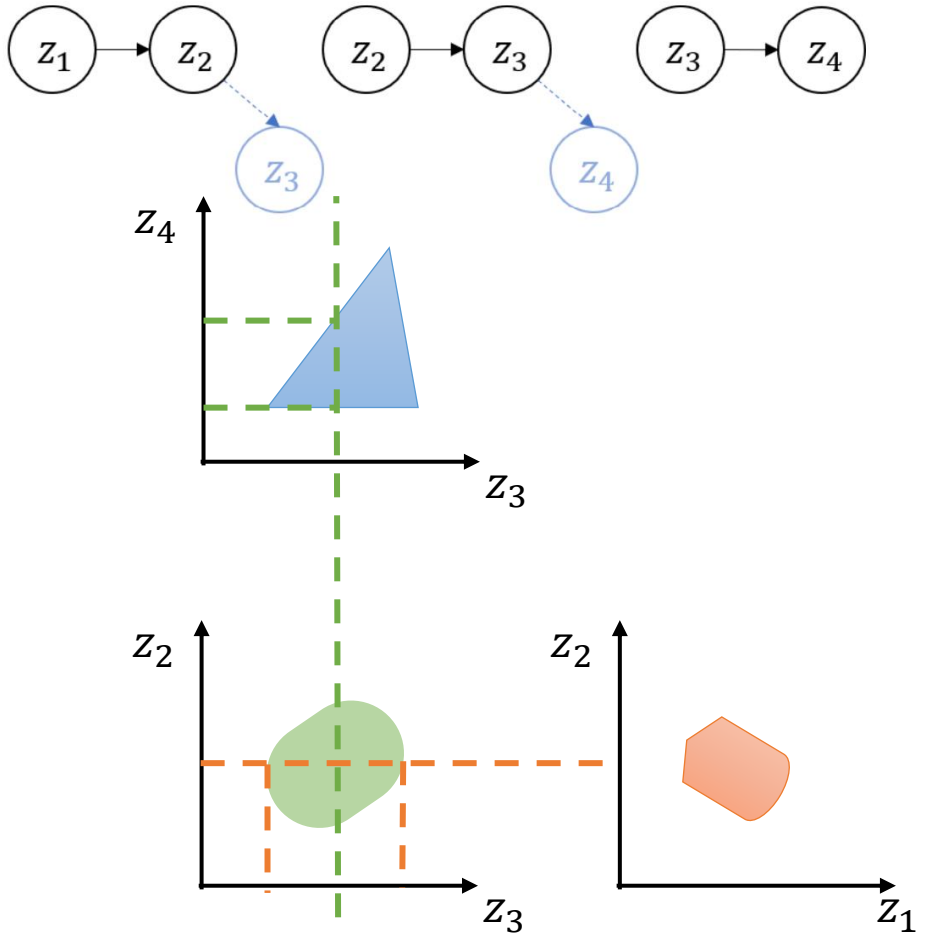
Missing States

- (z_3, z_4) is self-contained
  - Induces a bound for z_4, given z_3
- Evolution of (z_2, z_3) depends on z_4, which is unknown
  - Use worst-case z_4 given all available information, in this case value of z_3
  - This induces a bound for z_3, given z_2
- Evolution of (z_1, z_2) depends on z_3, which is unknown
  - Use worst-case z_3 given all available information, in this case value of z_2



Result: Over-approximation

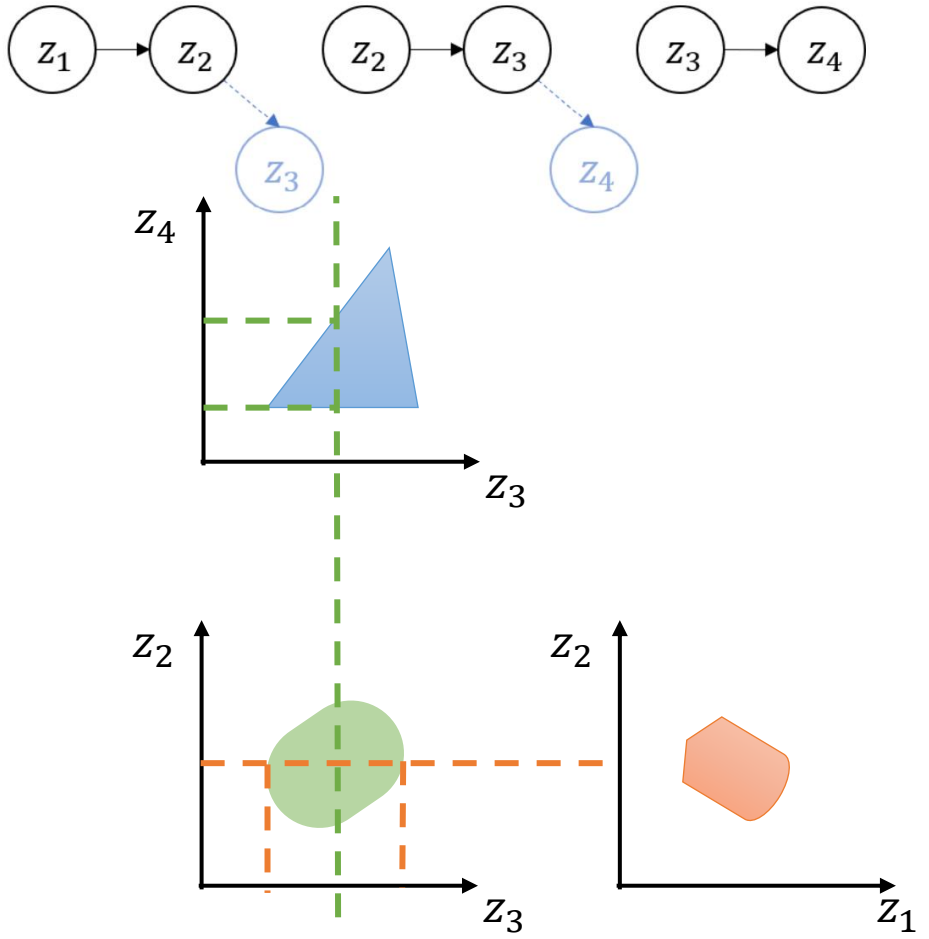
- Coloured shapes represent projections of reachable sets onto each subspace
- Over-approximation of true reachable set is obtained by back-projection and intersection
- Safety is guaranteed



Computational Complexity

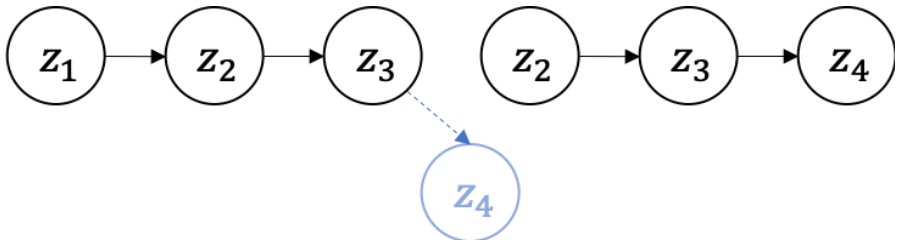
- Memory:
  - Need to store value functions for each subsystem
  - So maximum dimension of subsystems
  - (Black nodes, 2 in this case)

- Time:
  - Need to compute value function for each subsystem
  - Need to search over missing variables
  - So maximum dimension of subsystems, including missing states
  - (Black + blue nodes, 3 in this case)



Missing States

- (z_2, z_3, z_4) is self-contained
  - Induces a bound for z_4, given (z_2, z_3)



- Evolution of (z_1, z_2, z_3) system depends on z_4, which is unknown
  - Use worst-case z_4 given all available information, in this case value of (z_2, z_3)

Approximation vs Groundtruth

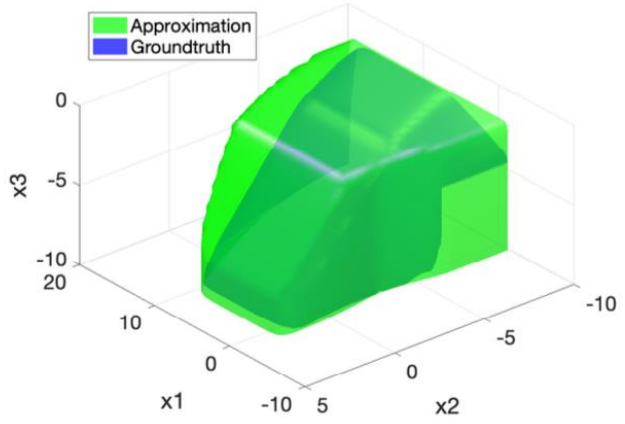
4D Quadruple Integrator

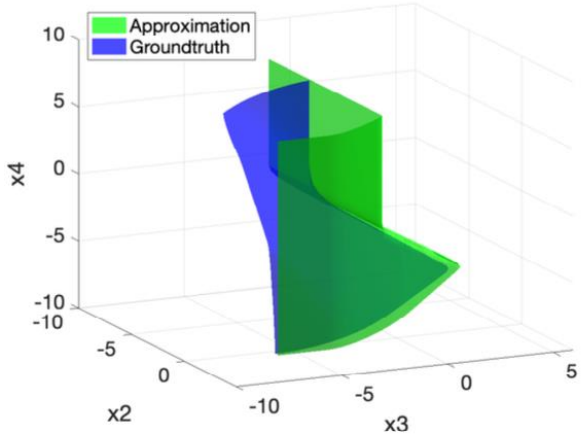
$$\begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \\ \dot{z}_3 \\ \dot{z}_4 \end{bmatrix} = \begin{bmatrix} z_2 + d \\ z_3 \\ z_4 \\ u \end{bmatrix}$$

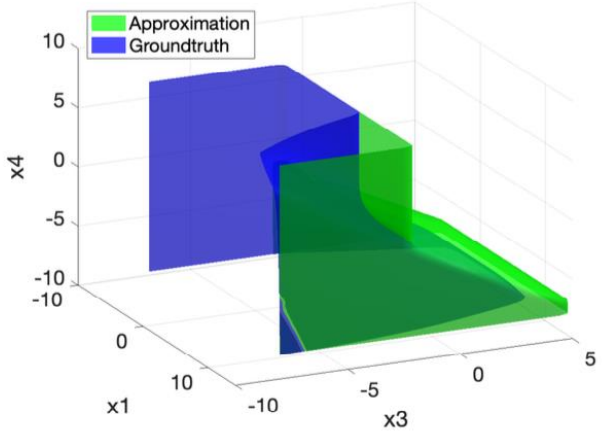
$$\dot{x}_1 = \begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \end{bmatrix}$$

$$\dot{x}_2 = \begin{bmatrix} \dot{z}_2 \\ \dot{z}_3 \end{bmatrix}$$

$$\dot{x}_3 = \begin{bmatrix} \dot{z}_3 \\ \dot{z}_4 \end{bmatrix}$$







Time comparison:
Groundtruth: 420s
Ours: 2.5s

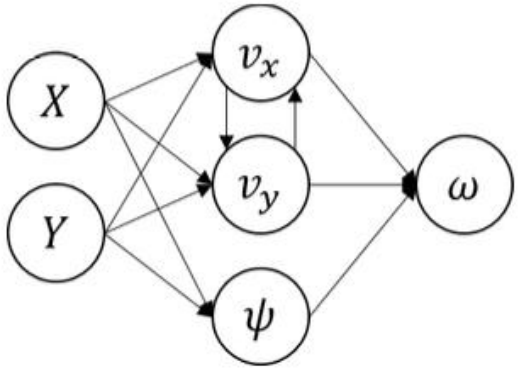
Groundtruth (blue)
Our over-approximation (green)

Over-approximation ⇒ guaranteed to avoid unsafe states

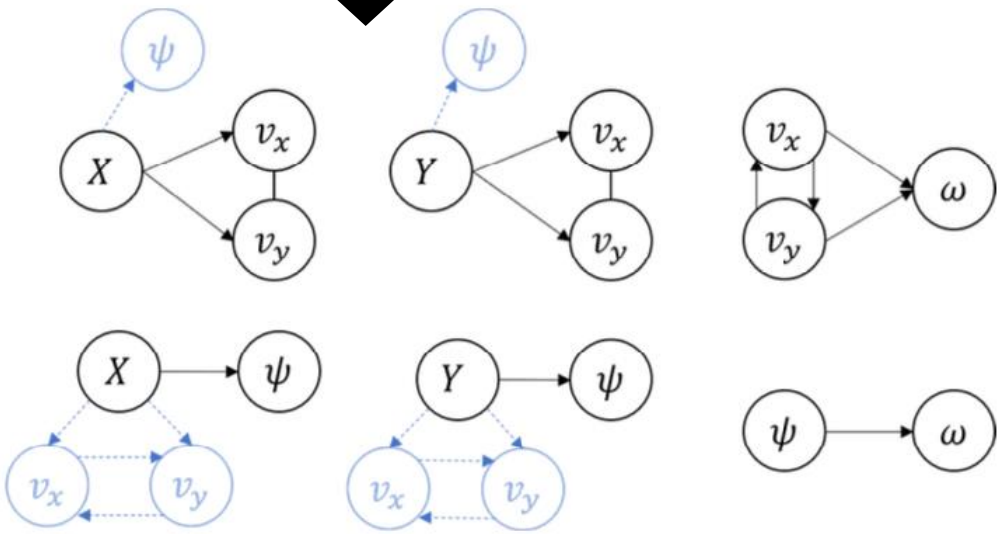
6D Bicycle Model

- Commonly used to model autonomous vehicles

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\psi} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_x \cos\psi - v_y \sin\psi \\ v_x \sin\psi + v_y \cos\psi \\ \omega \\ \omega v_y + a_x \\ -\omega v_x + \frac{2}{m}(F_{c,f}\cos\delta_f + F_{c,r}) \\ \frac{2}{I_z}(l_f F_{c,f} - l_r F_{c,r}) \end{bmatrix}$$

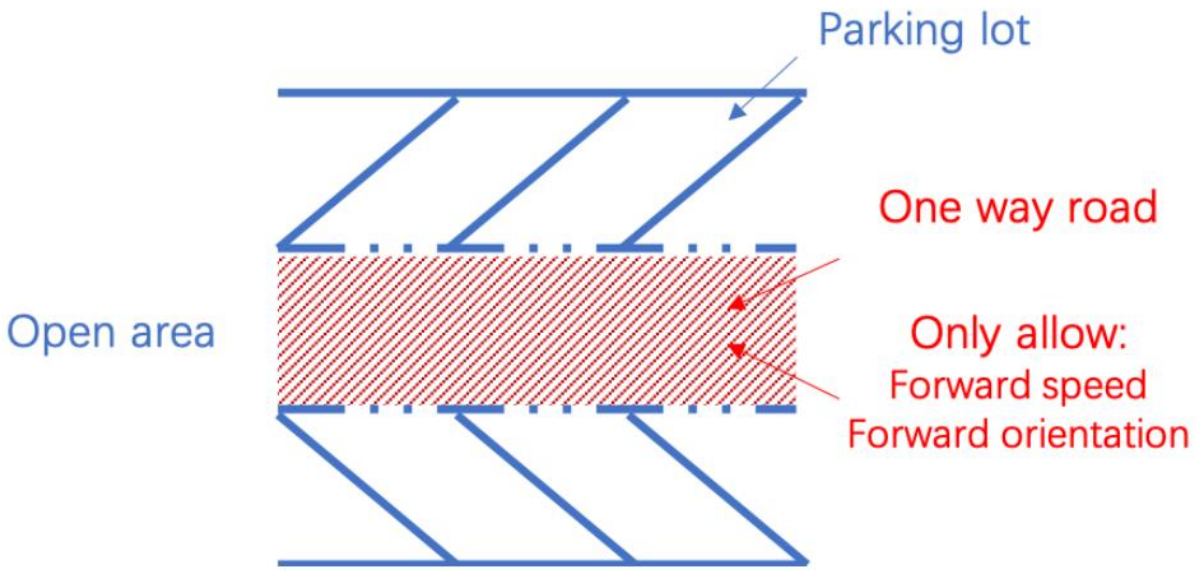


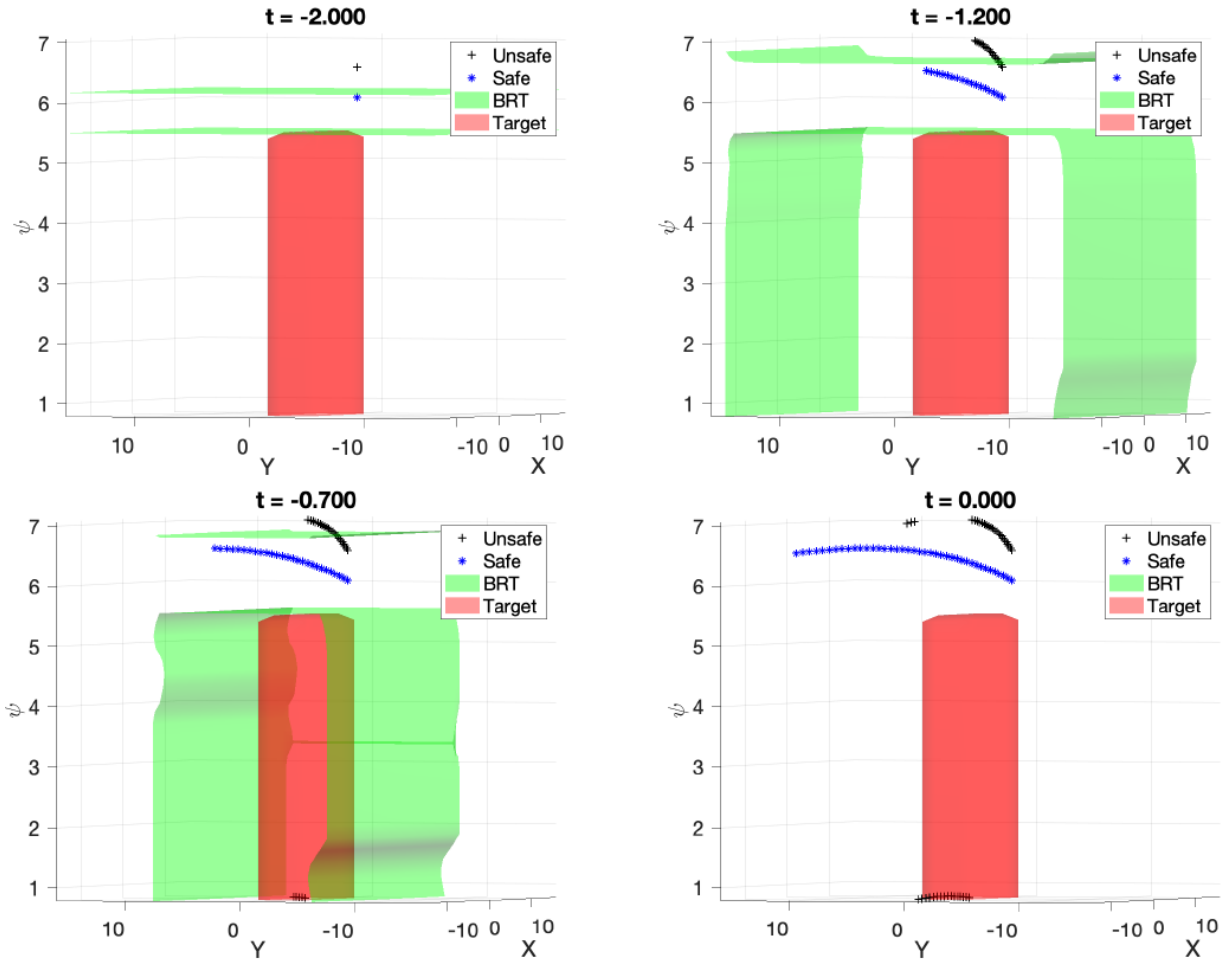
- Decomposed form:
  - 3D value functions
  - 4D computations



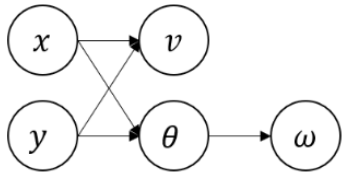
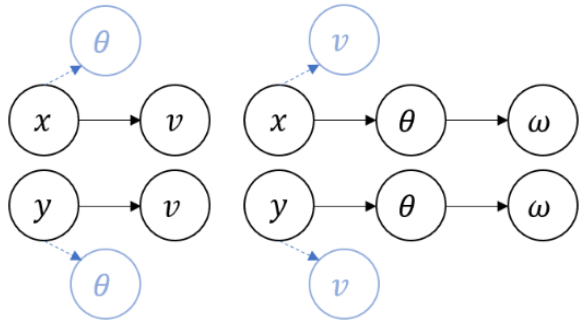
6D Bicycle Model: Decomposition

- Parking lot scenario

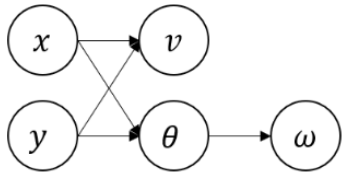
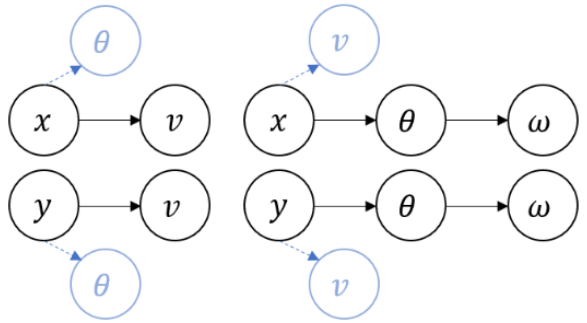
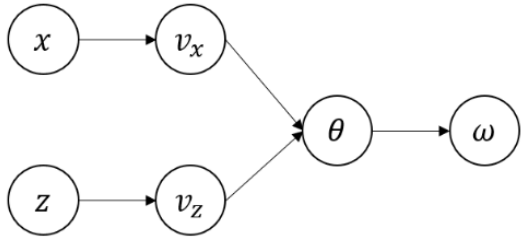
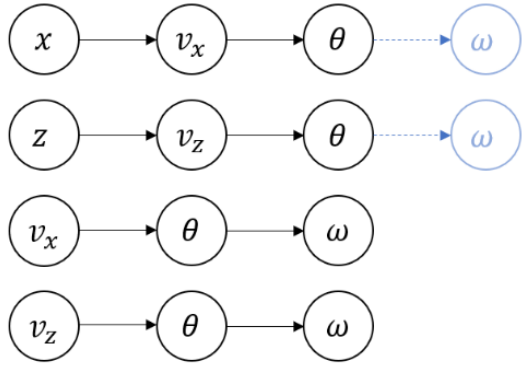




Other Examples

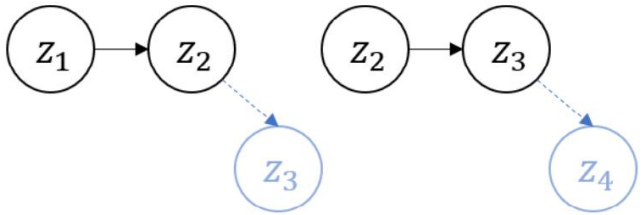
System configuration	System dynamics	State Dependency Graph	Decomposed State Dependency Graph	Time and space
5D Car (x, y)-position θ - heading v - speed ω - turn rate u_a - accel. control u_α - ang. accel. control	$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \\ u_a \\ u_\alpha \end{bmatrix}$			Ground truth: both $O(k^5)$ Decomposition: $O(k^4)$ and $O(k^3)$

Other Examples

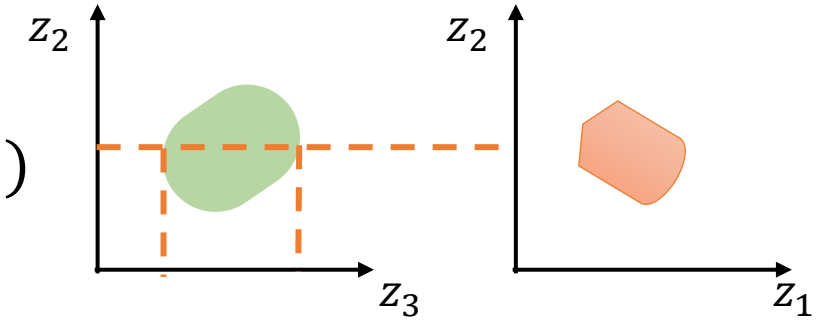
System configuration	System dynamics	State Dependency Graph	Decomposed State Dependency Graph	Time and space
5D Car (x, y)-position θ - heading v - speed ω - turn rate u_a - accel. control u_α - ang. accel. control	$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \\ u_a \\ u_\alpha \end{bmatrix}$			Ground truth: both $O(k^5)$ Decomposition: $O(k^4)$ and $O(k^3)$
6D Planar Quadrotor (x, y)-position (v_x, v_z) - velocity θ - pitch ω - pitch rate u_T - thrust control u_τ - ang.accel.control	$\begin{bmatrix} \dot{x} \\ \dot{z} \\ \dot{v_x} \\ \dot{v_z} \\ \dot{\theta} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_x \\ v_z \\ -u_T\sin\theta \\ u_T\cos\theta - g \\ \omega \\ u_\tau \end{bmatrix}$			Ground truth: both $O(k^6)$ Decomposition: $O(k^4)$ and $O(k^3)$

Future Work and Limitations

- Leaking corners
  - Reconstructing the full-dimensional reachable set involves taking intersections or union of subsystem reachable sets
  - For collision avoidance, one must take the intersection, not union
  - Well-known problem in optimal control and differential games

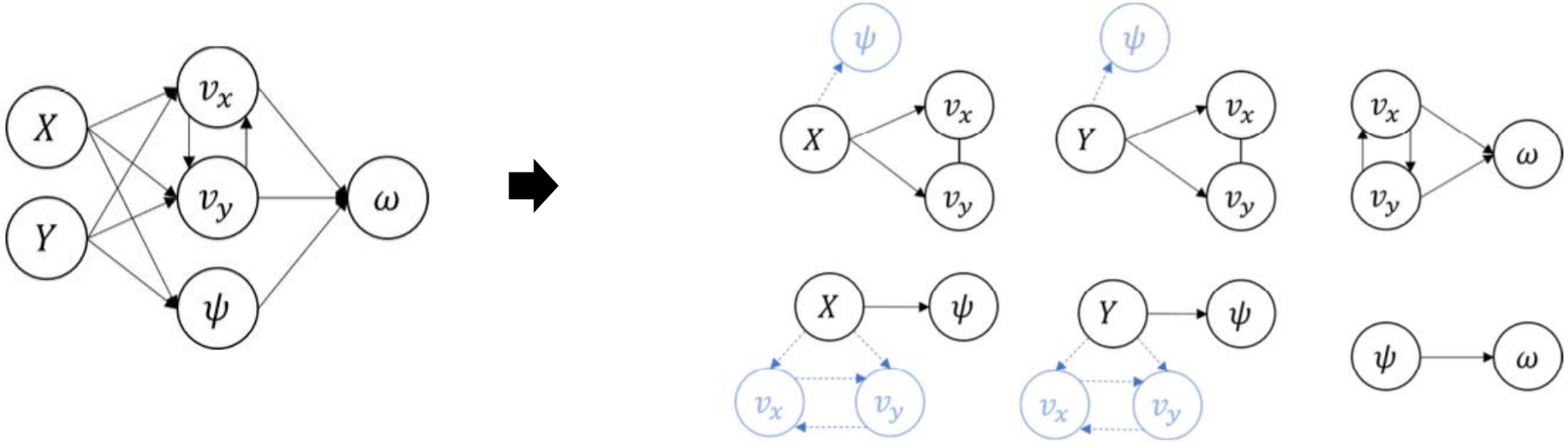


- Sets must exist in other subspaces
  - For example, reachable set cannot be empty in (z_2, z_3)



Future Work and Limitations

- Automatic decomposition based on directed graph



z_1

Outline

- The challenge: curse of dimensionality
  - Finite difference: a numerically convergent approach
- Dimensionality reduction
  - “Chained” systems and projections
- Optimizing software
  - Maximizing brute-force computation capabilities
- Novel applications of HJ PDE solutions in machine learning

Outline

- The challenge: curse of dimensionality
  - Finite difference: a numerically convergent approach
- Dimensionality reduction
  - “Chained” systems and projections
- **Optimizing software**
  - **Maximizing brute-force computation capabilities**
- Novel applications of HJ PDE solutions in machine learning

Optimizing Numerical Methods for PDEs

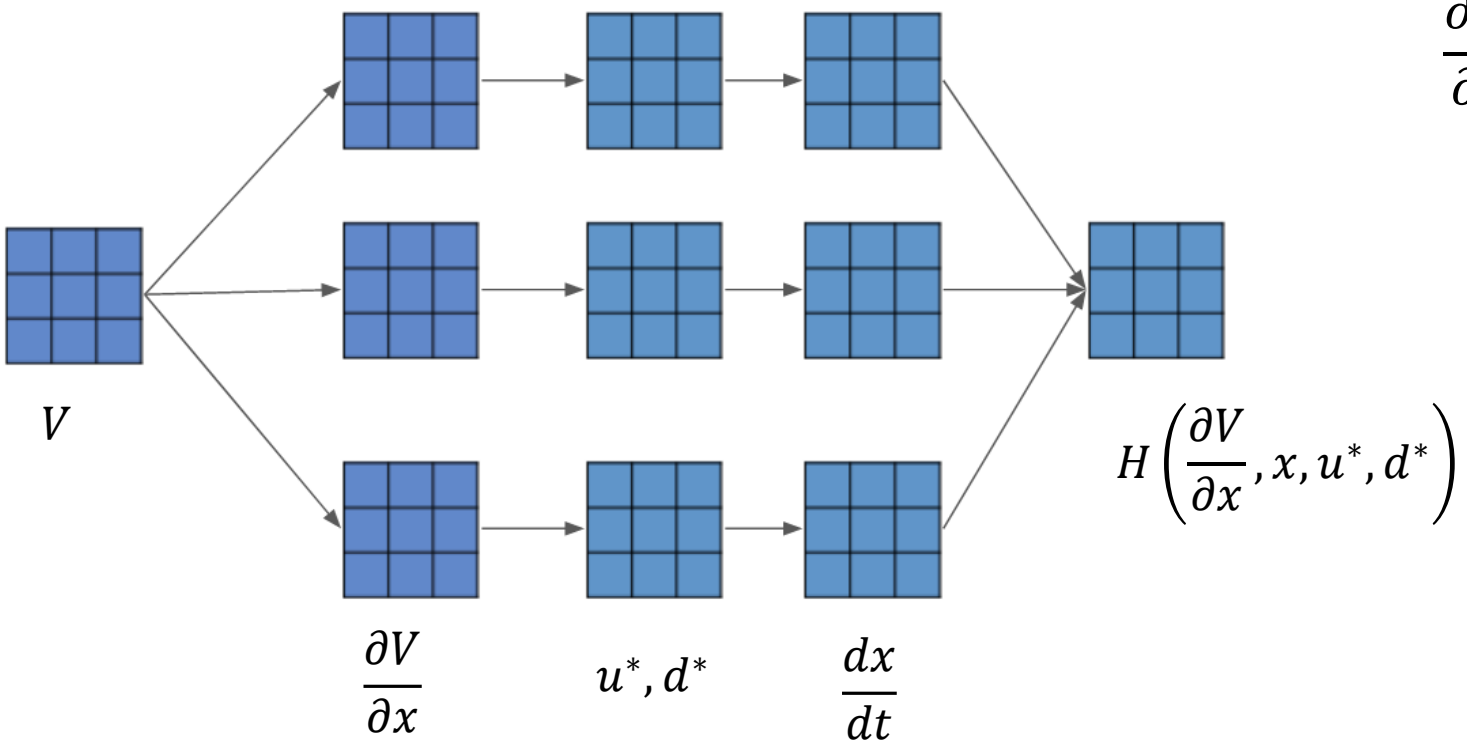
- Much of the progress in deep learning depends on being able to utilize a large amount of compute resources

- PDE solvers are most often run locally

- Can we alleviate computational challenges via software optimization?
  - Different hardware such as CPU, GPU, FPGA and cloud compute
  - Enables analysis of higher-dimensional subsystems
  - Enables real-time applications

Time-Dependent HJ PDE

- Consider the time-dependent HJ PDE
  - Similar observations can be made for other variants

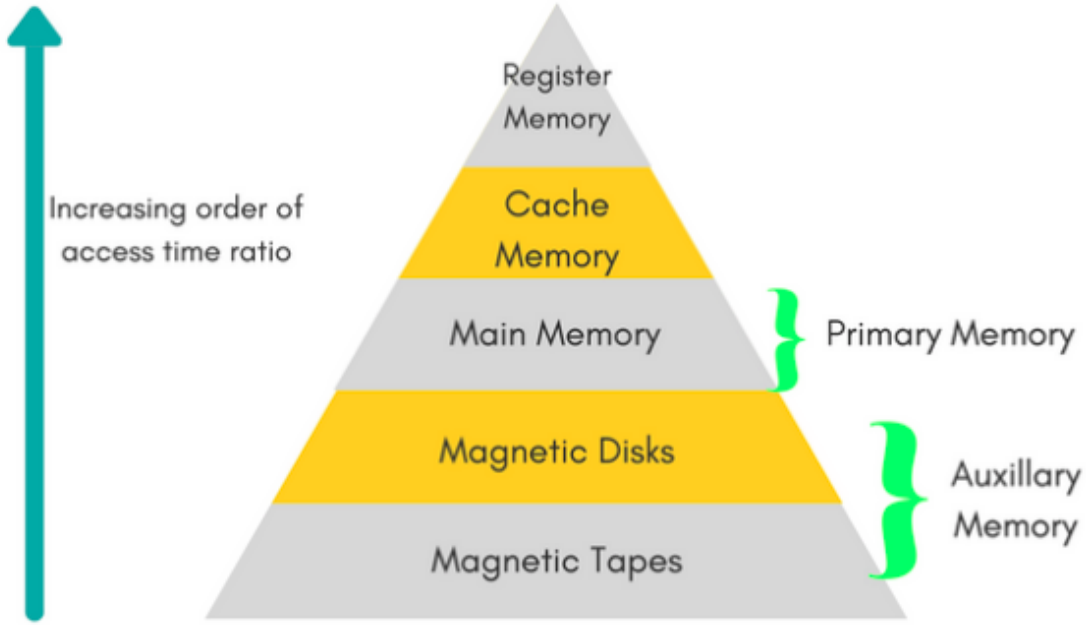


$$\frac{\partial V}{\partial t} + \underbrace{\max_u \min_d \left[\left(\frac{\partial V}{\partial x}\right)^{\mathsf{T}} f(x,u,d)\right]}_{H\left(\frac{\partial V}{\partial x}, x, u^*, d^*\right)} = 0$$

Optimization

- Memory locality and loop order
- Parallelism
- HeteroCL
- Tiling
- Dynamics-dependent loop order

Computer Memory Pyramid



Relative access latency comparison:

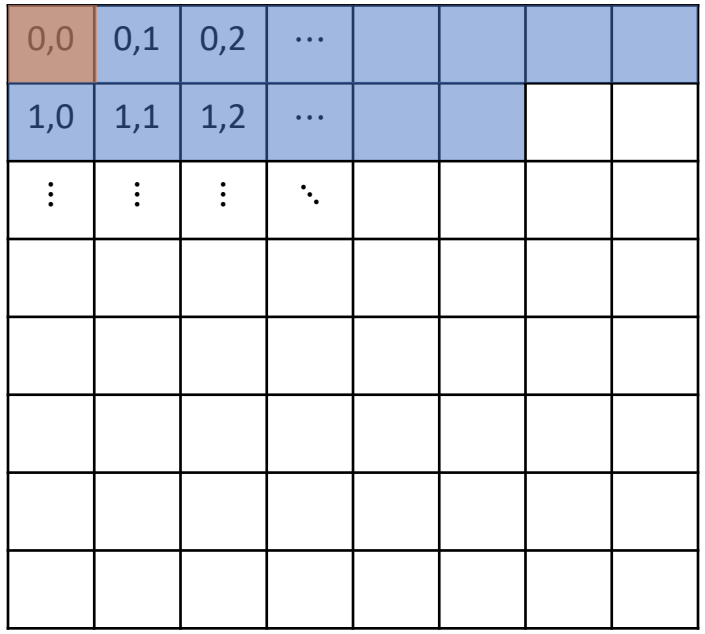
- Register memory : (1 x)
- Cache : (~ 16 x)
- Main memory: (~ 60 x)
- Disk: (~ 100 x - 1000 x)

Cache

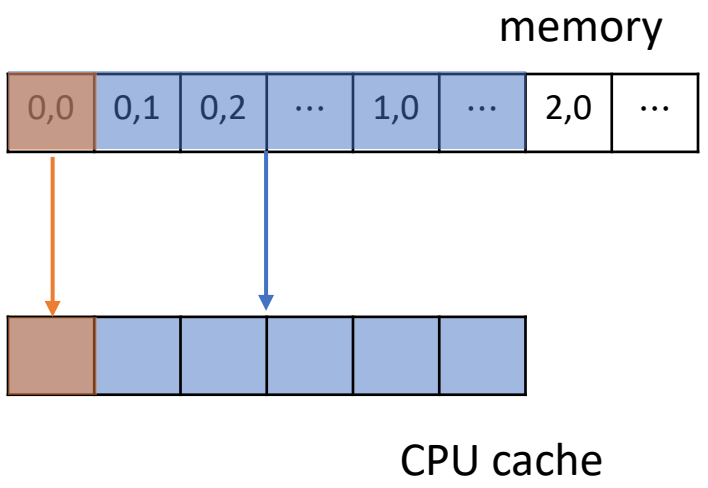
- keeps a copy of a neighbourhood of accessed memory
- Flushed and replaced if data is unused after a while

Memory Locality and Loop order

- Memory is linear
- When a memory location is accessed, “neighbours” are loaded into the CPU cache to improve access times
- Therefore, in the example it is faster to iterate over rows



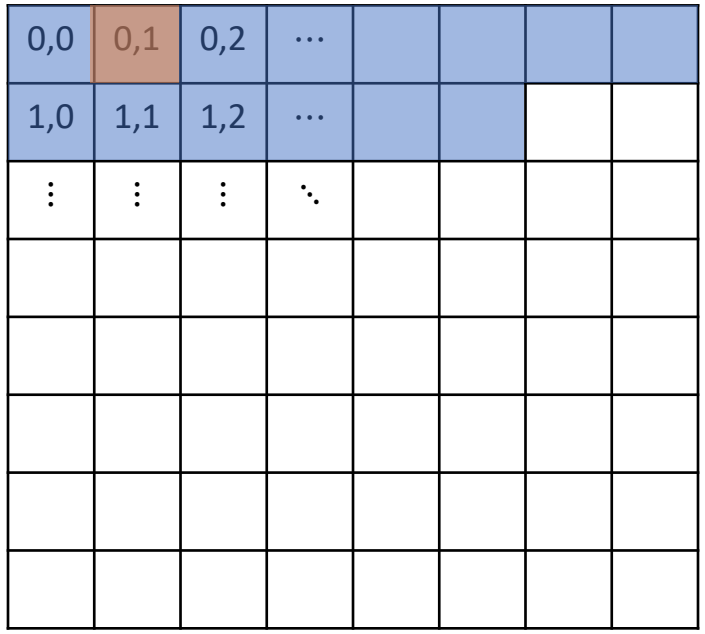
Grid



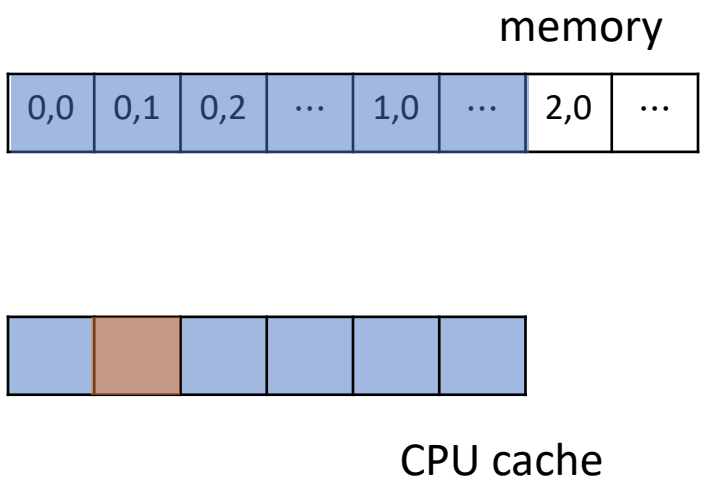

```
//allocate V_0[100][100][100]
//allocate Hamiltonian[100][100][100]
for (i = 0 ... 100)
  for (j = 0 ... 100)
    for (k = 0 .. 100)
      // Calculate dV_dx
      dV_dx = ...
        .
        .
        .
      // Calculate time dynamics
      dx_dt = ...
      Hamiltonian[i][j][k] = ...
```

Memory Locality and Loop order

- Memory is linear
- When a memory location is accessed, “neighbours” are loaded into the CPU cache to improve access times
- Therefore, in the example it is faster to iterate over rows



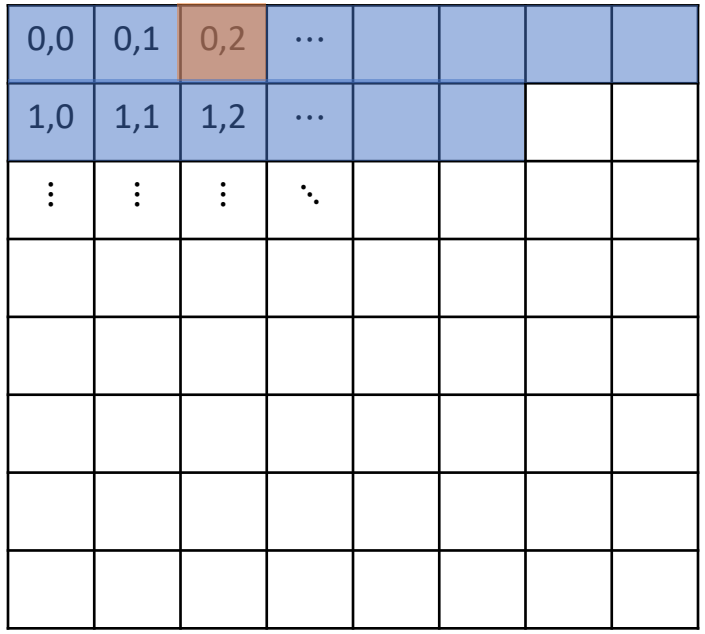
Grid



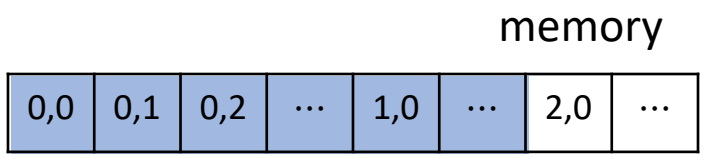

```
//allocate V_0[100][100][100]
//allocate Hamiltonian[100][100][100]
for (i = 0 ... 100)
  for (j = 0 ... 100)
    for (k = 0 .. 100)
      // Calculate dV_dx
      dV_dx = ...
        .
        .
        .
      // Calculate time dynamics
      dx_dt = ...
      Hamiltonian[i][j][k] = ...
```

Memory Locality and Loop order

- Memory is linear
- When a memory location is accessed, “neighbours” are loaded into the CPU cache to improve access times
- Therefore, in the example it is faster to iterate over rows



Grid

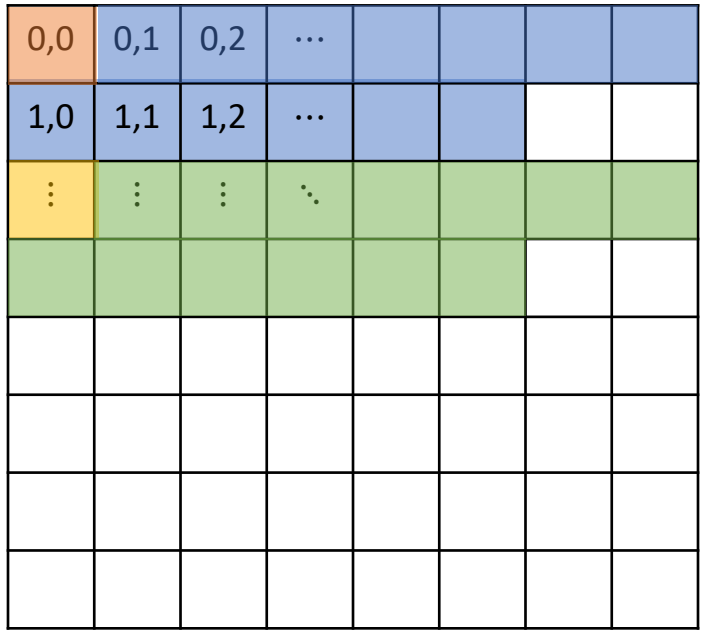


CPU cache

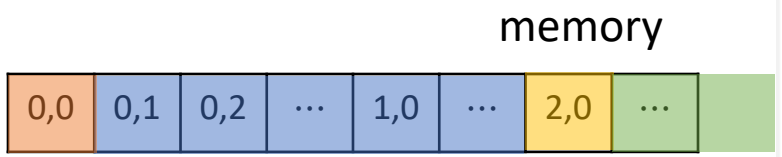
```
//allocate V_0[100][100][100]
//allocate Hamiltonian[100][100][100]
for (i = 0 ... 100)
  for (j = 0 ... 100)
    for (k = 0 .. 100)
      // Calculate dV_dx
      dV_dx = ...
        .
        .
        .
      // Calculate time dynamics
      dx_dt = ...
      Hamiltonian[i][j][k] = ...
```

Parallelism

- Memory is linear
- When a memory location is accessed, “neighbours” are loaded into the CPU cache to improve access times
- Therefore, in the example it is faster to iterate over rows



Grid

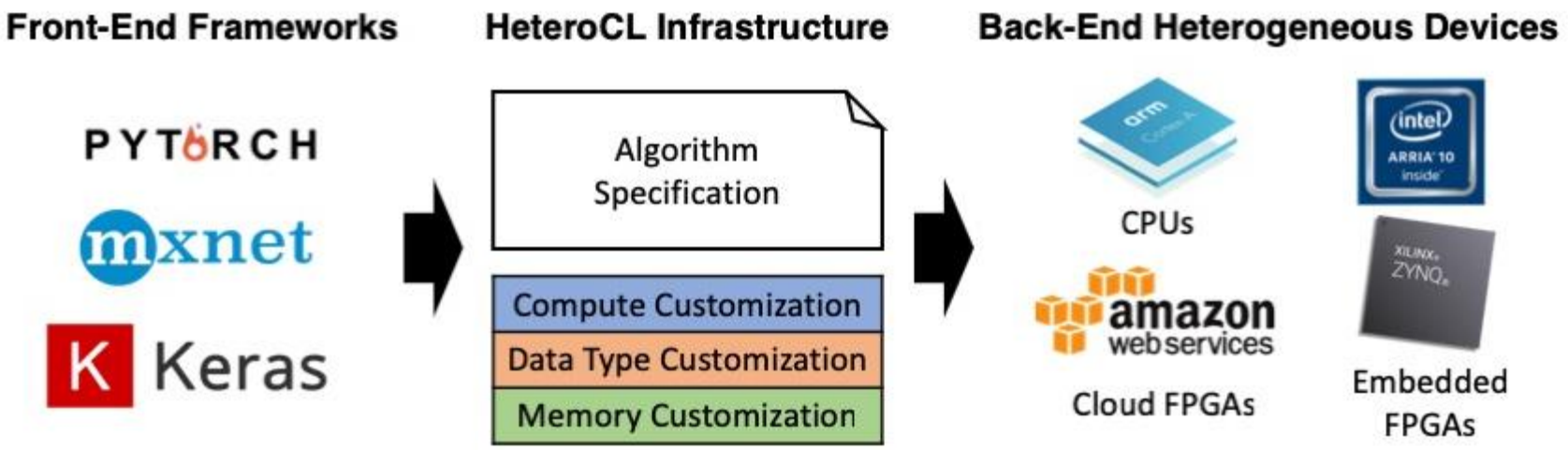


CPU cache

```
//allocate V_0[100][100][100]
//allocate Hamiltonian[100][100][100]
for (i = 0 ... 100)
  for (j = 0 ... 100)
    for (k = 0 .. 100)
      // Calculate dV_dx
      dV_dx = ...
        .
        .
        .
      // Calculate time dynamics
      dx_dt = ...
      Hamiltonian[i][j][k] = ...
```

HeteroCL

- HeteroCL is a programming infrastructure composed of a Python-based domain-specific language (DSL) and a compilation flow
  - http://heterocl.csl.cornell.edu/web/



HeteroCL

- Start with a naive version of the code (e.g. simple nested for loops)
  - Transformations of code for optimization purposes automatically done with HeteroCL
  - No need to manually re-implement anything
  - Guarantees algorithmic correctness
- Main algorithm written in Python (user friendly)
  - HeteroCL-specific syntax translates Python code to to high performance C code
- Easy to re-use and optimize code for different hardware platforms in mind (e.g. CPUs, GPUs, FPGAs)

Results: Dubins Car

$$\dot{x} = v\cos\theta$$
$$\dot{y} = v\sin\theta$$
$$\dot{\theta} = \omega$$

- Memory locality and loop order + parallelism
  - Implemented with HeteroCL
  - Times reported in seconds, per integration time step

Intel i9-9900K CPU at 3.8 GHz (Desktop)

Grid size	80^3	90^3	100^3
MATLAB level set toolbox	0.047	0.0726	0.10

Results: Dubins Car

$$\dot{x} = v\cos\theta$$
$$\dot{y} = v\sin\theta$$
$$\dot{\theta} = \omega$$

- Memory locality and loop order + parallelism
  - Implemented with HeteroCL
  - Times reported in seconds, per integration time step

Intel i9-9900K CPU at 3.8 GHz (Desktop)

Grid size	80^3	90^3	100^3
MATLAB level set toolbox	0.047	0.0726	0.10
heteroCL	0.0018	0.0022	0.0025
Speed-up	26 times	30 times	38 times

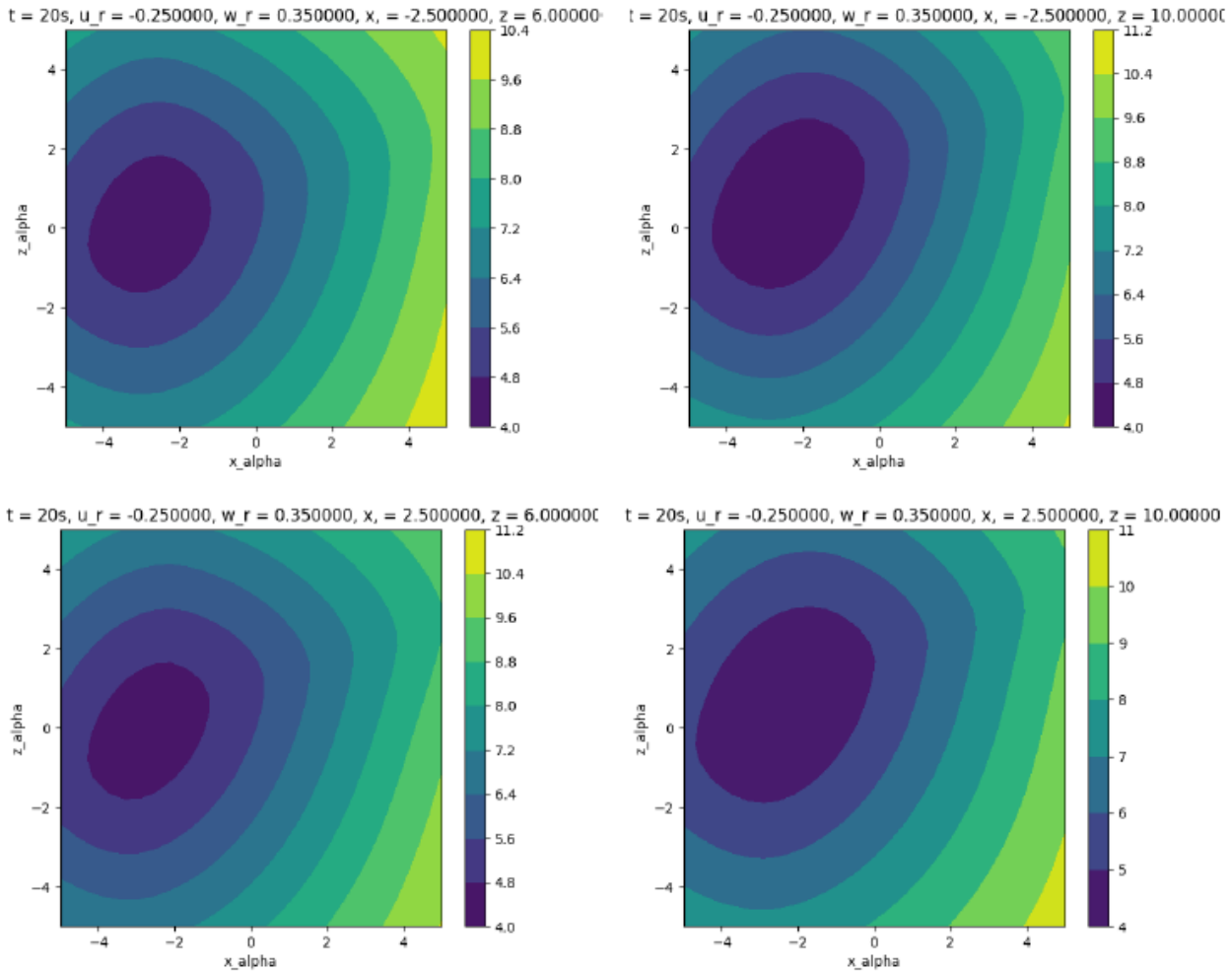
AMD A10-8700P CPU at 3.2 GHz (Laptop)

Grid size	80^3	90^3	100^3
MATLAB level set toolbox	0.58	0.83	1.05
heteroCL	0.011	0.015	0.018
Speed-up	54 times	57 times	58 times

Results: 6D Underwater Vehicle

$$\dot{x}_\alpha = u_r + G_1(z)\cos\big(\sigma(x,t)\big) + d_x - b_x$$
$$\dot{z}_\alpha = w_r + G_1(z)\big(-\sin\big(\sigma(x,t)\big)\big) + d_z - b_z$$
$$\dot{u}_r = \frac{1}{m - X_{\dot{u}}}(\Phi_{11}(b - X_{\dot{u}})u_r + \cdots$$
$$\dot{w}_r = \frac{1}{m - Z_{\dot{w}}}(\Phi_{22}(b - Z_{\dot{w}})w_r + \cdots$$
$$\dot{x} = u_r + G_1(z)\cos\big(\sigma(x,t)\big) + d_x$$
$$\dot{z} = w_r + G_1(z)\big(-\sin\big(\sigma(x,t)\big)\big) + d_z$$

- Tracking error bounds under plane-progressive waves
  - Grid size: 25^6
  - 5.1 seconds per step
  - Previously intractable

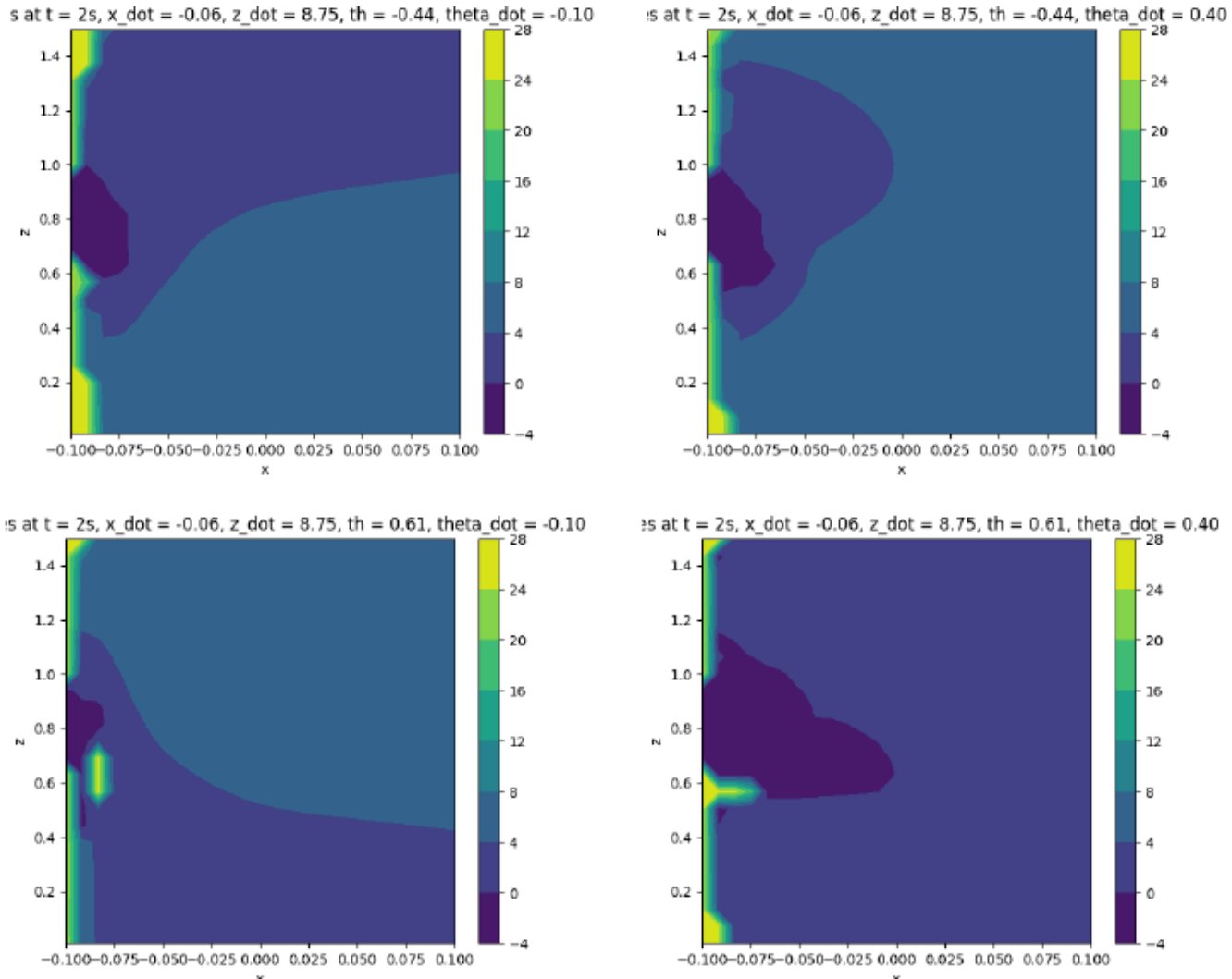


Siriya et al., CDC ‘20 (submitted)

Results: 6D Simplified Humanoid Model

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = \frac{g + u_2}{x_3}(x_1 + u_1) + u_3$$
$$\dot{x}_3 = x_4$$
$$\dot{x}_4 = u_2$$
$$\dot{x}_5 = x_6$$
$$\dot{x}_6 = \frac{x_3}{J} u_3$$

- Backward reachable set
  - Grid size: 25^6
  - 2 seconds per step
  - Previously intractable

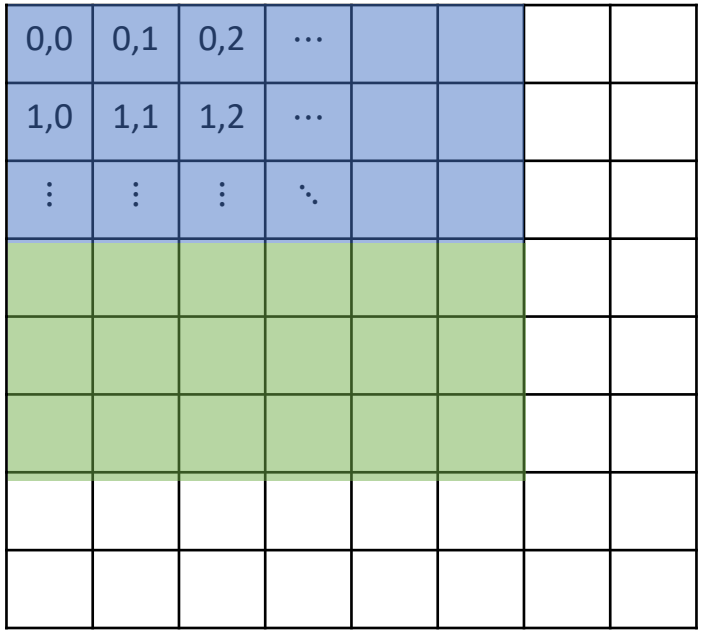


Future work

- Optimizations
  - Tiling
  - Dynamics-dependent loop order

Tiling

- Load a part of memory into CPU cache
  - Perform all computations that require this data
  - Nested loops that iterate only over this data
- Load another part of memory into CPU cache
  - This replaces old data, which is no longer needed
  - Perform all computations that require this data
- Minimize data movement



Dynamics-Dependent Loop Order

- Dubins Car dynamics:

$$\dot{x} = v\cos\theta$$
$$\dot{y} = v\sin\theta$$
$$\dot{\theta} = \omega$$

  - θ does not depend on x and y
  - So it would make sense to iterate over different θ values in the outer loop
    - Values of $\dot{x}$ and $\dot{y}$ would change less often in the inner loops
  - Combine this with tiling

Future work

- Optimizations
  - Tiling
  - Dynamics-dependent loop order
- Applications
  - Real-time reachability computations
  - Model fidelity study
- Easy-to-use toolbox

Outline

- The challenge: curse of dimensionality
  - Finite difference: a numerically convergent approach
- Dimensionality reduction
  - “Chained” systems and projections
- Optimizing software
  - Maximizing brute-force computation capabilities
- Novel applications of HJ PDE solutions in machine learning

Outline

- The challenge: curse of dimensionality
  - Finite difference: a numerically convergent approach
- Dimensionality reduction
  - “Chained” systems and projections
- Optimizing software
  - Maximizing brute-force computation capabilities
- **Novel applications of HJ PDE solutions in machine learning**

Novel Applications in Machine Learning

- Reinforcement Learning
  - Using HJ PDE solutions for reward shaping
- Visual Navigation
  - Using HJ PDE solutions as training data

Model-Free Reinforcement Learning

- Given: simulator of a system (or real system) and reward function
  - Determine (locally) optimal policy without knowing a model of the system
  - Try control policies that involve exploration and exploitation
  - Use control policies that acquire more reward more often

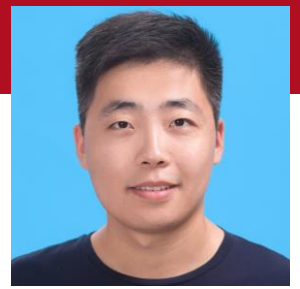
- Advantages
  - Does not require system dynamics
  - Flexible in terms of input data (eg. sensor data)
  - Scalable to large system state spaces when using function approximators

- Disadvantages
  - Poor sample efficiency and generalizability
  - Difficult to incorporate knowledge of system behaviour

Reward Shaping

- Tasks are sometimes specified by a sparse reward
  - Goal: high reward (e.g. +1000)
  - Obstacles: very negative reward (e.g. -5000)
  - Other states: 0 reward
- Very little reward signal!
- A couple of heuristics for reward shaping to achieve better signal during training
  - Distance to goal (closer is better)
  - Inverse reinforcement learning (requires expert demonstration)

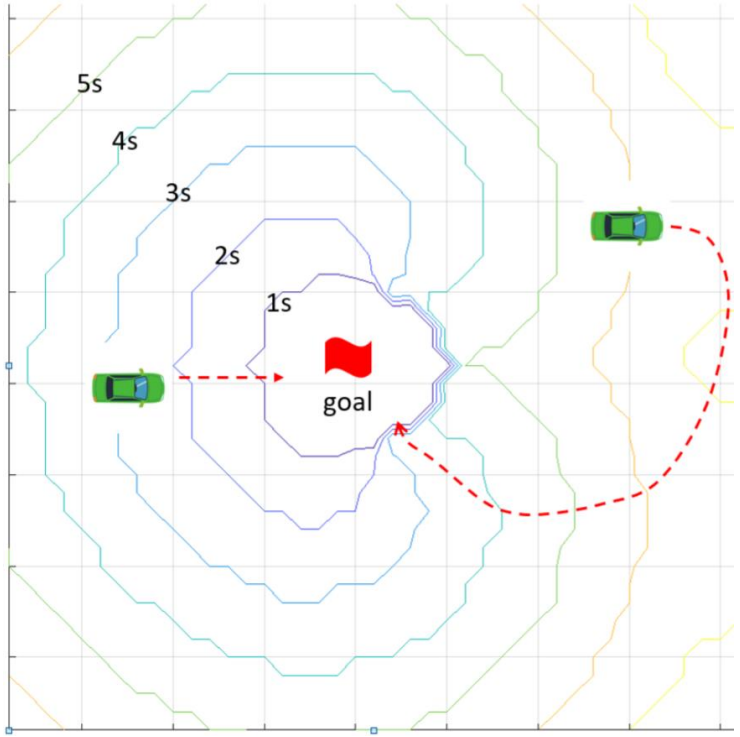
TTR-Based Reward Shaping



- TTR: "time-to-reach"
  - Minimum time required to reach the goal from any state s, $T(s)$
  - Requires system dynamics $\dot{s} = f(s, u)$
  - Intractable to compute for complex systems

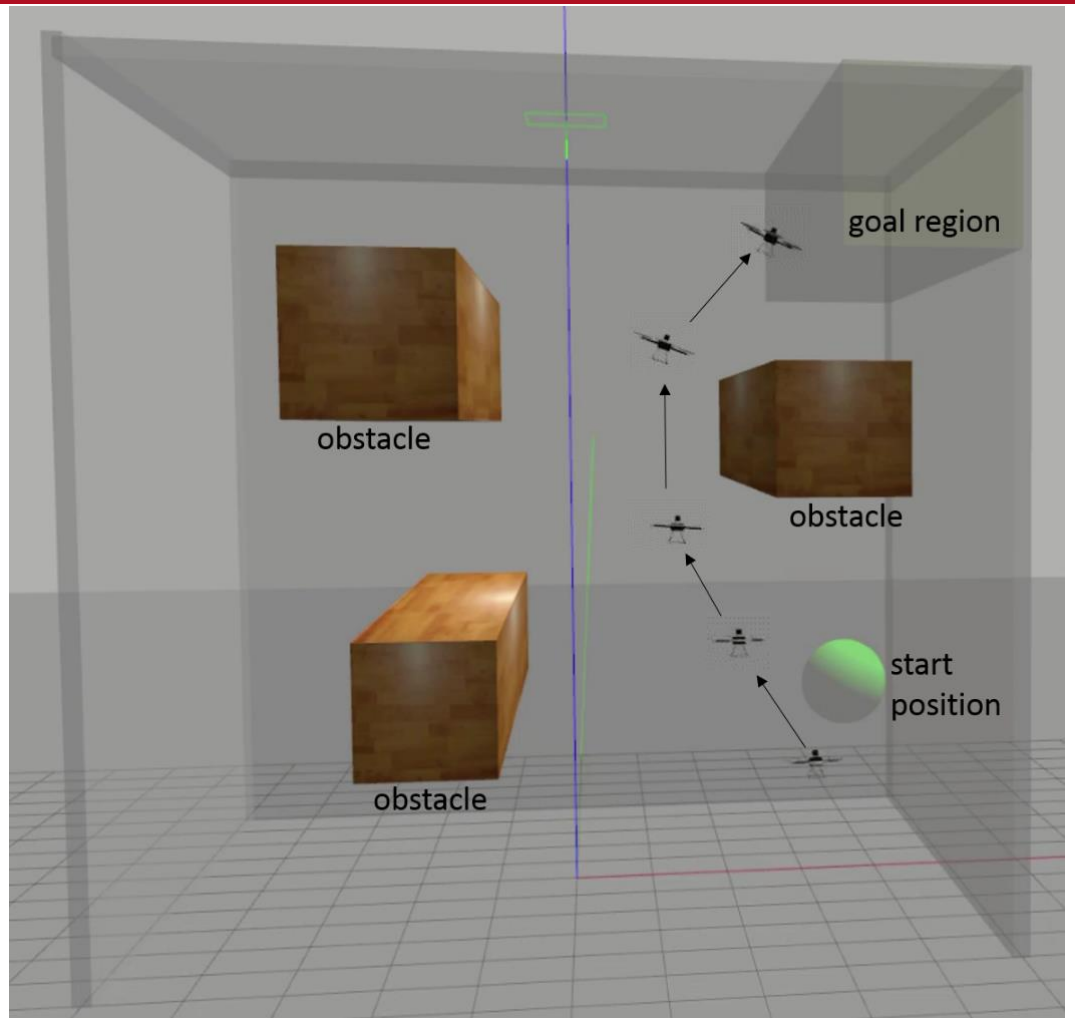
- Idea: compute the TTR approximately using a simplified system model $\hat{\dot{s}} = \hat{f}(\hat{s}, \hat{u})$
  - Approximately account for system dynamics

- Approximate TTR: $\hat{T}(\hat{s})$ → reward: $R(s) = -\hat{T}(\hat{s})$
  - "Good" states are those that require less time to reach the target, **given system dynamics**



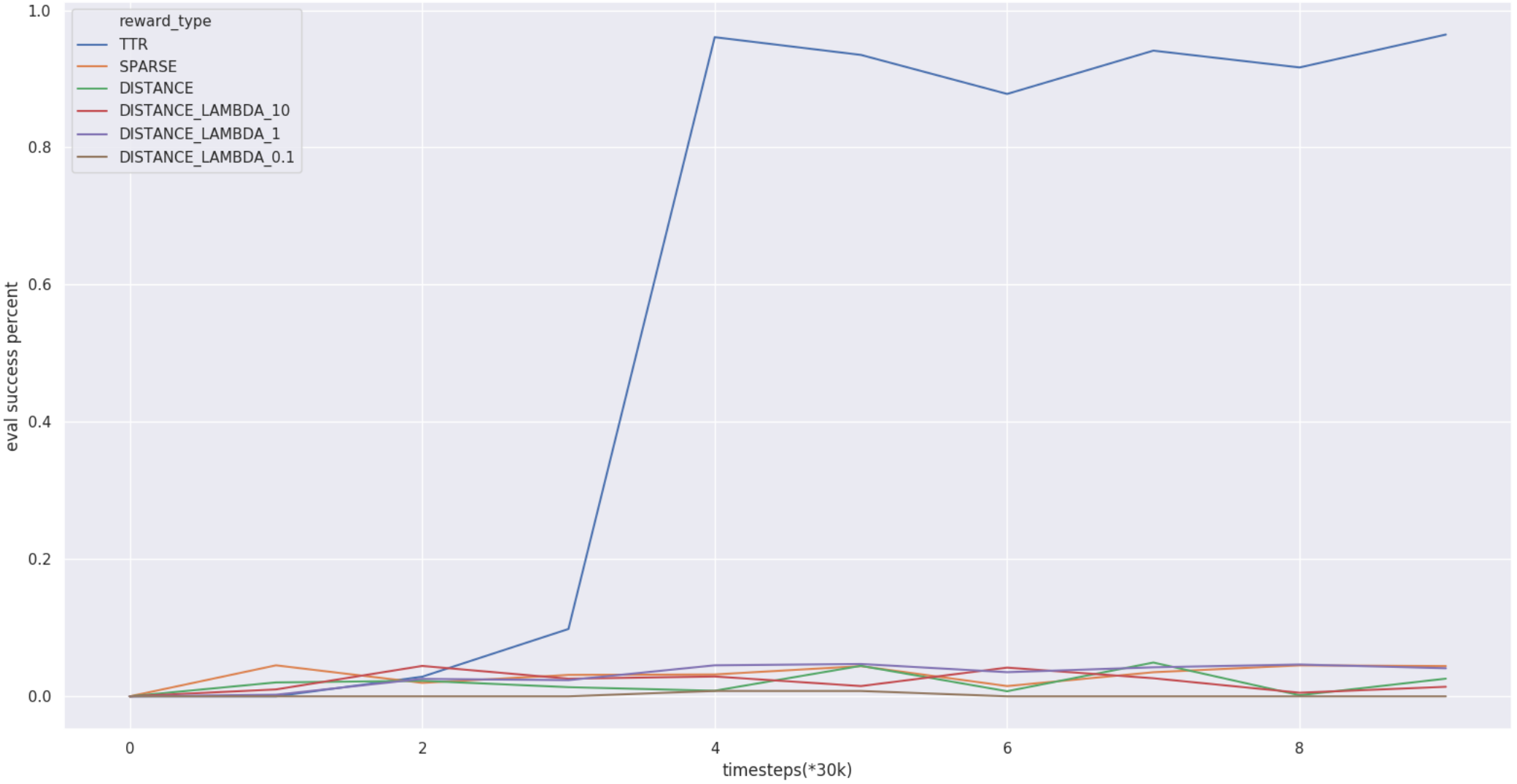
Lyu, Chen, IROS '20 (submitted)

Planar Quadrotor With LIDAR Example



Quadrotor simulation

Success From Start State: Value-Based RL

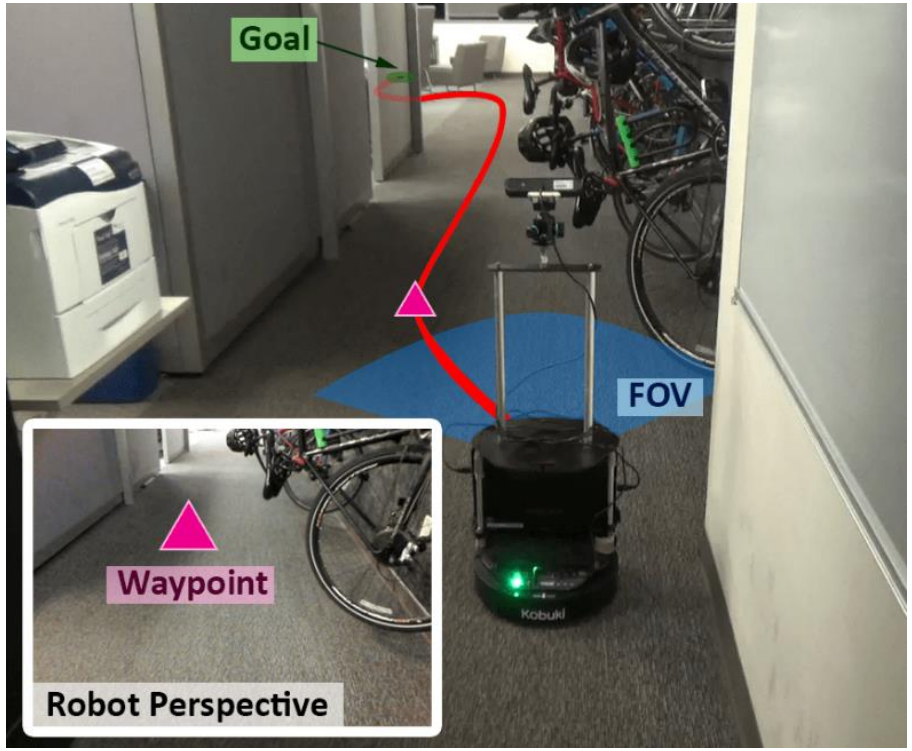


Success From Start State: Actor-Critic RL

Visual Navigation




- Navigate through an environment using camera only
  - Cameras are cheap
- End-to-end: pixels → actions
- Control-inspired: pixels → waypoint
  - Use control theory to plan trajectory to the waypoint
  - No need to learn what we already know
  - Train in simulation (Stanford indoor data set)
  - Results are transferrable to real life

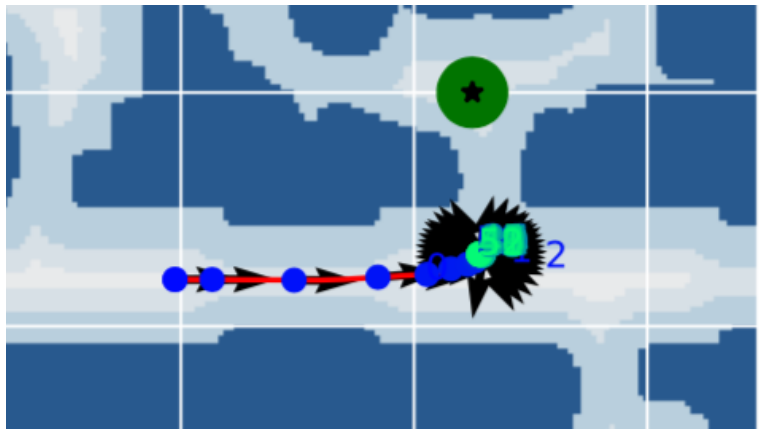


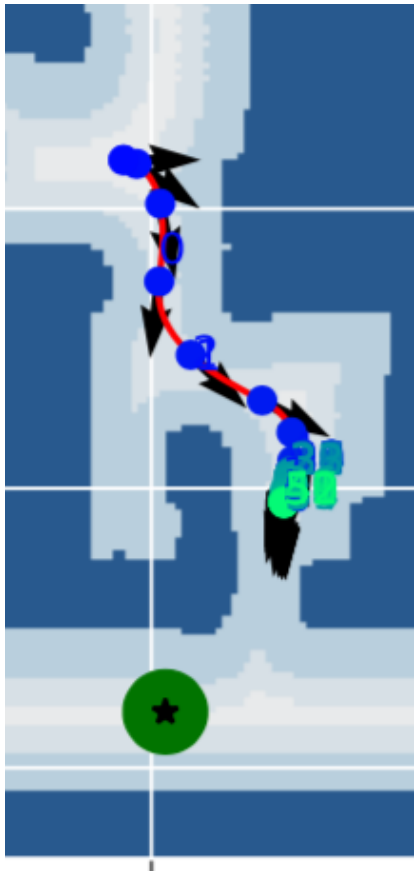
Bansal et al. CoRL 2019
Li et al. L4DC 2020

Visual Navigation

- Challenge:
  - How to generate training data?

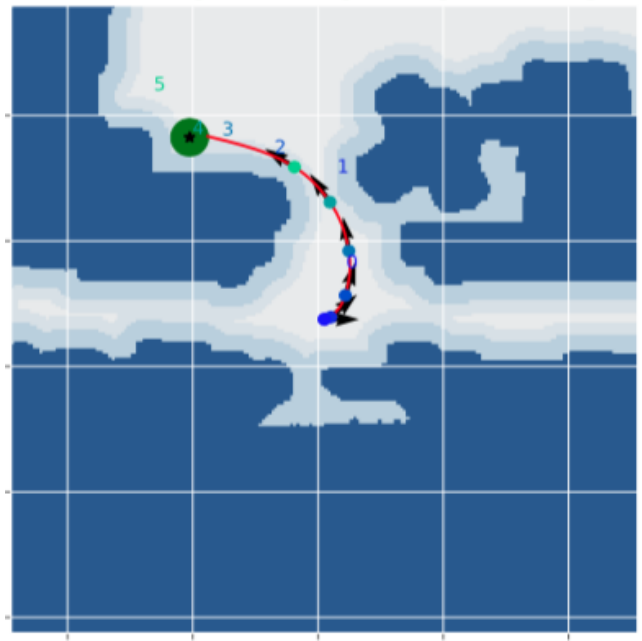
- Need to know what is a “good” waypoint
  - Waypoints are chosen based on simple heuristics involving distances to goal and obstacles may not be sufficient

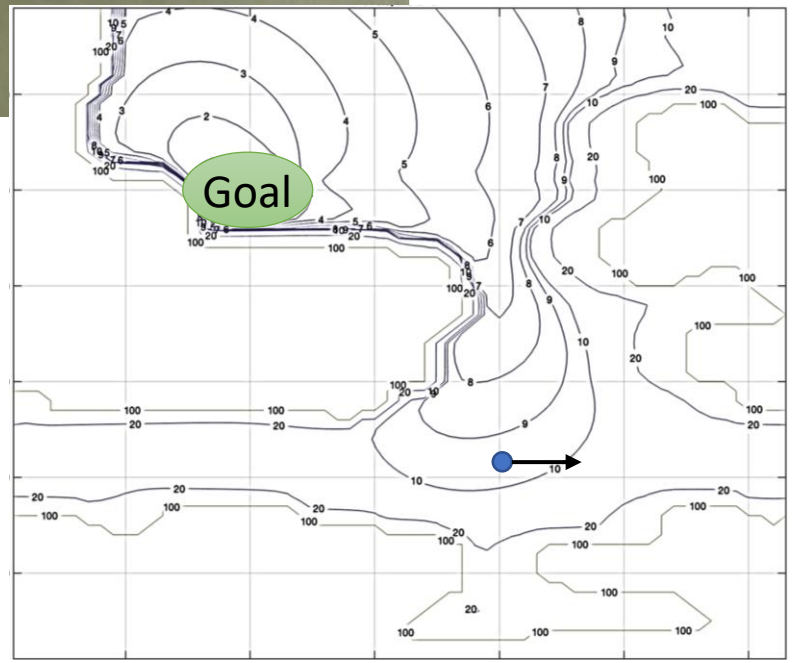




Visual Navigation

- Supervision by HJ PDE solution
  - Time-to-reach and time-to-collision functions can be used as a metric for the quality of a waypoint
- Data generation is time-consuming

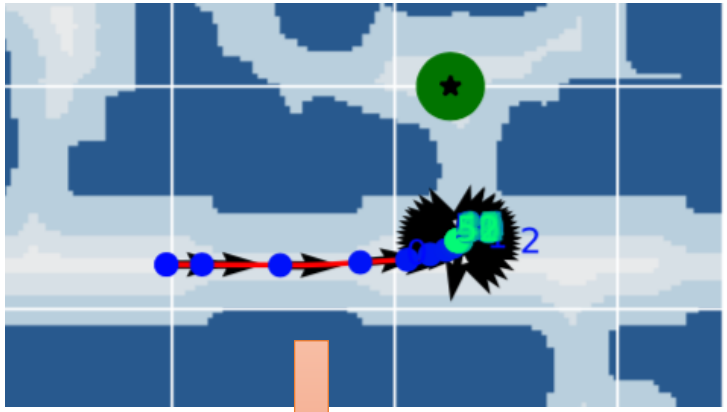




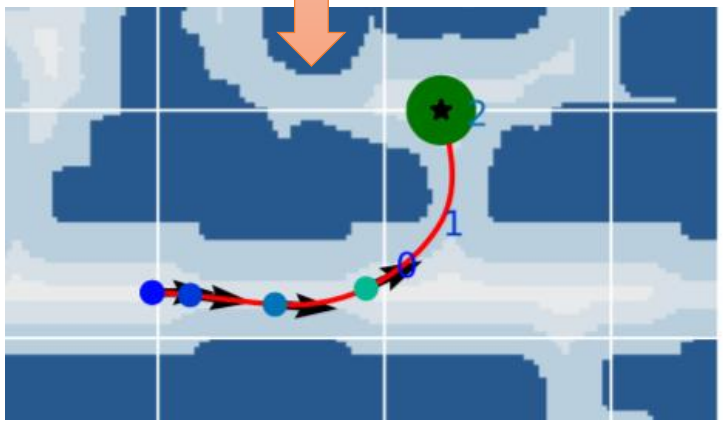
Li et al. L4DC 2020

Visual Navigation: Training Data

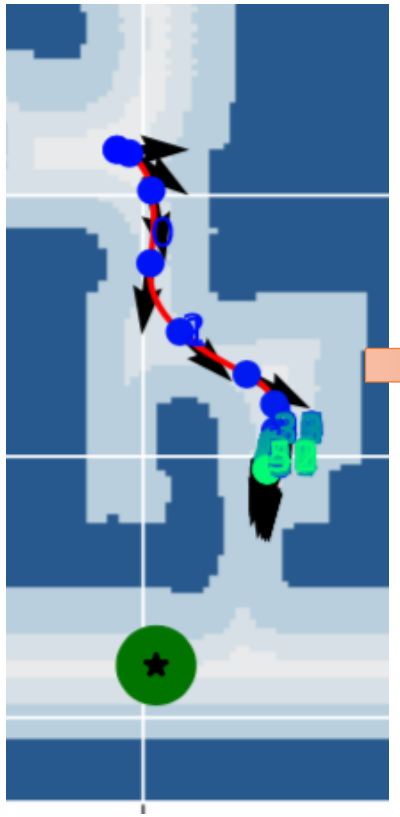
Distance heuristic



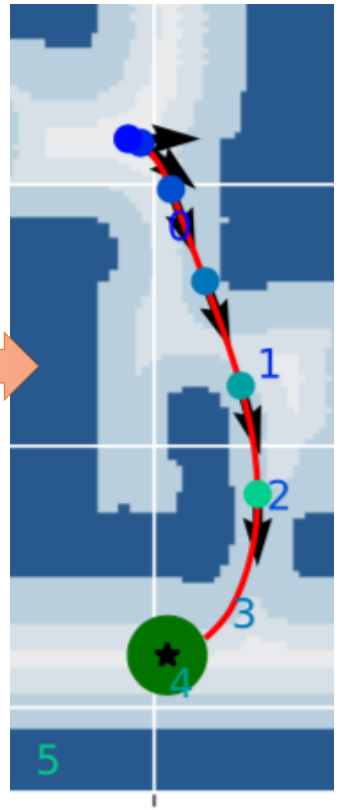
Time-to-reach



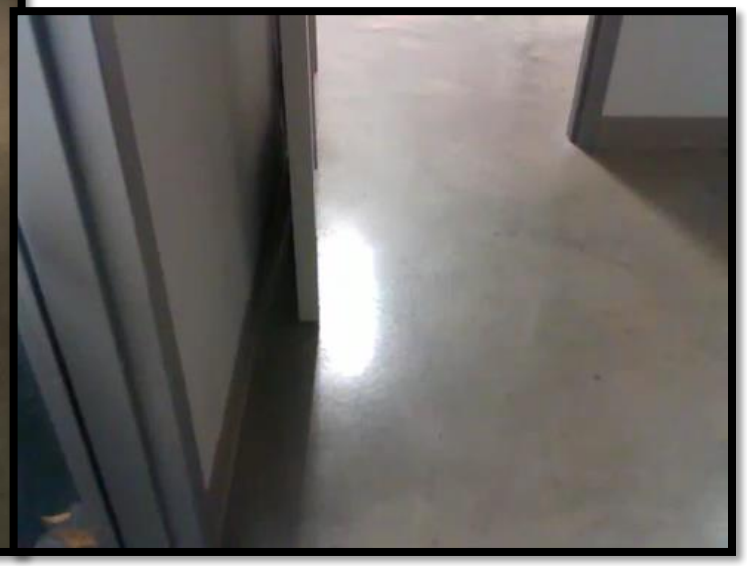
Distance heuristic



Time-to-reach



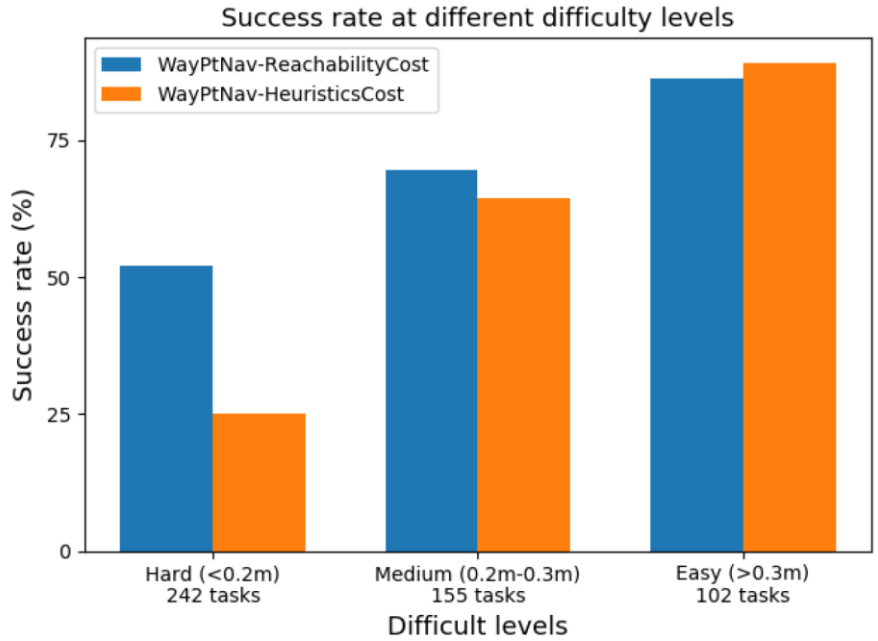
Visual Navigation – Test Performance



3rd person view

1st person view

Agent	Success (%)	Time taken (s)	Acceleration (m/s^2)	Jerk (m/s^3)
WayPtNav-ReachabilityCost	**63.82**	21.00 ±8.00	**0.06 ±0.01**	**0.94 ±0.13**
WayPtNav-HeuristicsCost	52.26	18.82 ±5.66	0.07 ±0.02	1.06 ±0.15
WayPtNav-ReachabilityCost-NoDstb	49.24	**16.19 ±4.8**	0.07 ±0.01	0.98 ±0.16
E2E-ReachabilityCost	8.04	19.55 ±4.72	0.07 ±0.01	2.16 ±0.30
E2E-HeuristicsCost	31.66	25.56 ±9.85	0.26 ±0.06	9.06 ±1.94



Thank you!

Mo Chen
mochen@cs.sfu.ca
https://sfumars.com

- The challenge: curse of dimensionality
  - Finite difference: a numerically convergent approach
- Dimensionality reduction
  - “Chained” systems and projections
- Optimizing software
  - Maximizing brute-force computation capabilities
- Novel applications of HJ PDE solutions in machine learning