Projects

Projects during Ph.D. program

Presenter: Le Pham Tuyen
Artificial Intelligence Laboratory
Department of Computer Science and Engineering

20^{th} December 2018

Projects

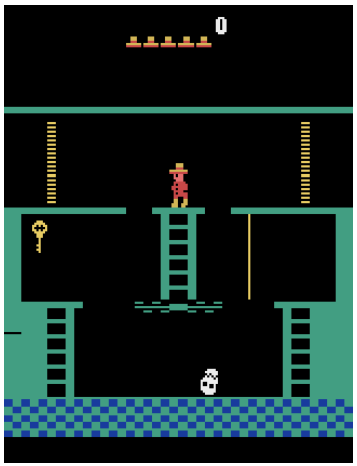
1. Deep Hierarchical Reinforcement Learning under Partially Observable Markov Decision Processes (Dissertation)

2. Learning Autonomous Bicycle/Ballbot using Deep Reinforcement Learning

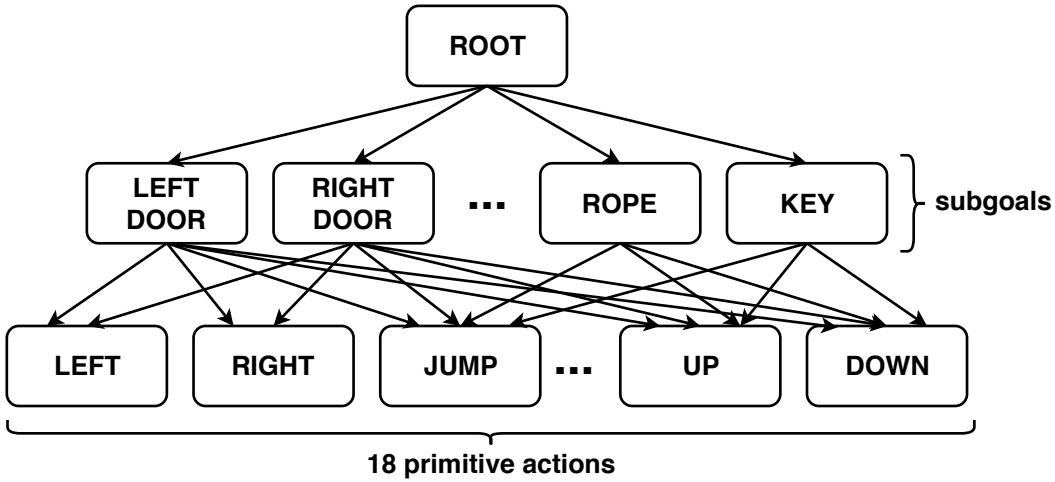
Deep HRL under POMDPs

Hierarchical Tasks

- Hierarchical tasks are popular in real-world applications. E.g.
  - An agent navigates to the key before reaching the door to open.
  - Tasks of a taxi: go to to the passengers, pick up, go to to the destination, take off.
  - A robot plans to go to the door before going to the destination.



(a) Montezuma's Revenge

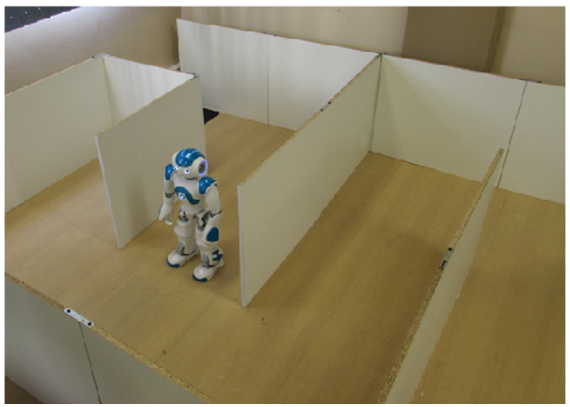


(b) The hierarchy of Montezuma's Revenge domain

Hierarchical Domain

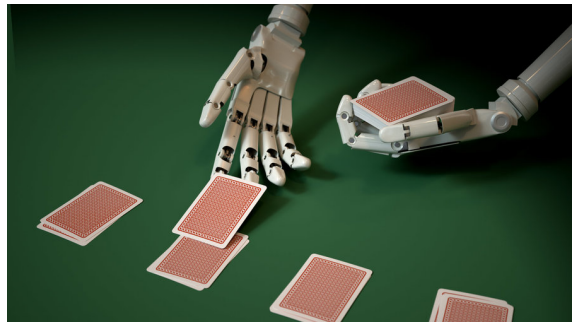
Partial Observability

- Agent only observes a part of the environment
- **POMDP** is popular in the real-world applications. E.g.
  - A robot with camera vision isn't told its absolute location
  - A trading agent only observes current prices
  - A poker playing agent only observes public cards



(a) Robot Navigation

(b) Trading Bot



(c) Poker Bot

Some POMDP domains

Problem and Proposed Concept

Problem Statement

Planning under partial observability still has many challenges.

- The complexity of hierarchical tasks
- Uncertanty under partial observability

Proposed concept

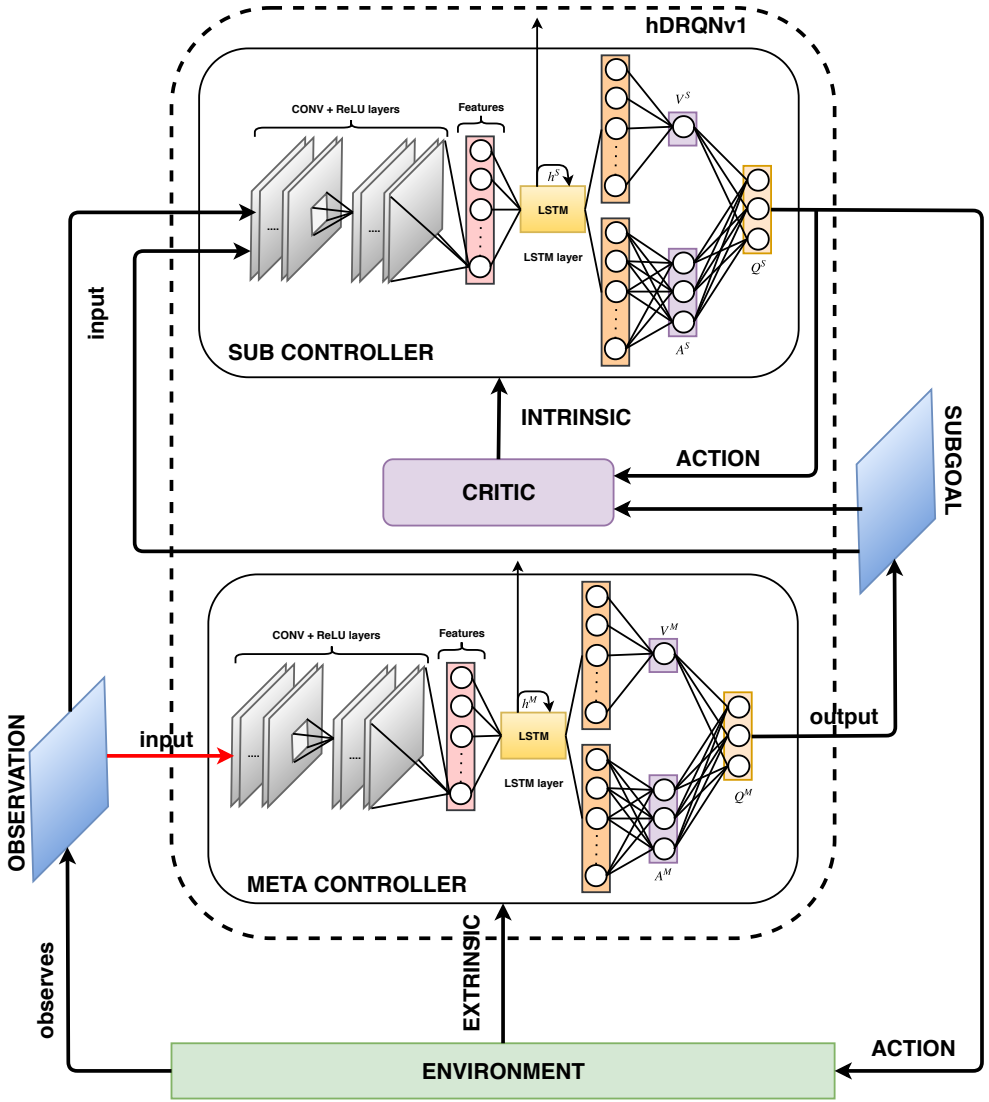
Hierarchical Deep Recurrent Q Learning Algorithms

- Using two levels of policy to control the hierarchical task.
- Integrate RNN to tacke partial observability problem.

Contributions

- **Develop:** **h**ierarchical **D**eep **R**ecurrent **Q**-Learning algorithms (**hDRQNs**) in order to handle **hierarchical tasks** in **POMDP**. Particularly,
  - We develop hDRQNv1 algorithm which learns a framework of hierarchical polices.
    - ★ Two levels of hierarchical polices: meta-controller is the upper policy and sub-controller is the lower policy.
    - ★ Two hierarchical policies integrated recurrent neural networks are expected to overcome the challenges under partial observability
  - We develop hDRQNv2 algorithm of a proposed framework which integrates recurrent neural networks in a different way, thus expected to have better performance.
- To the best of our knowledge, our research is the first study that learns Montezuma's Revenge under partial observability.

hDRQN: Framework 1



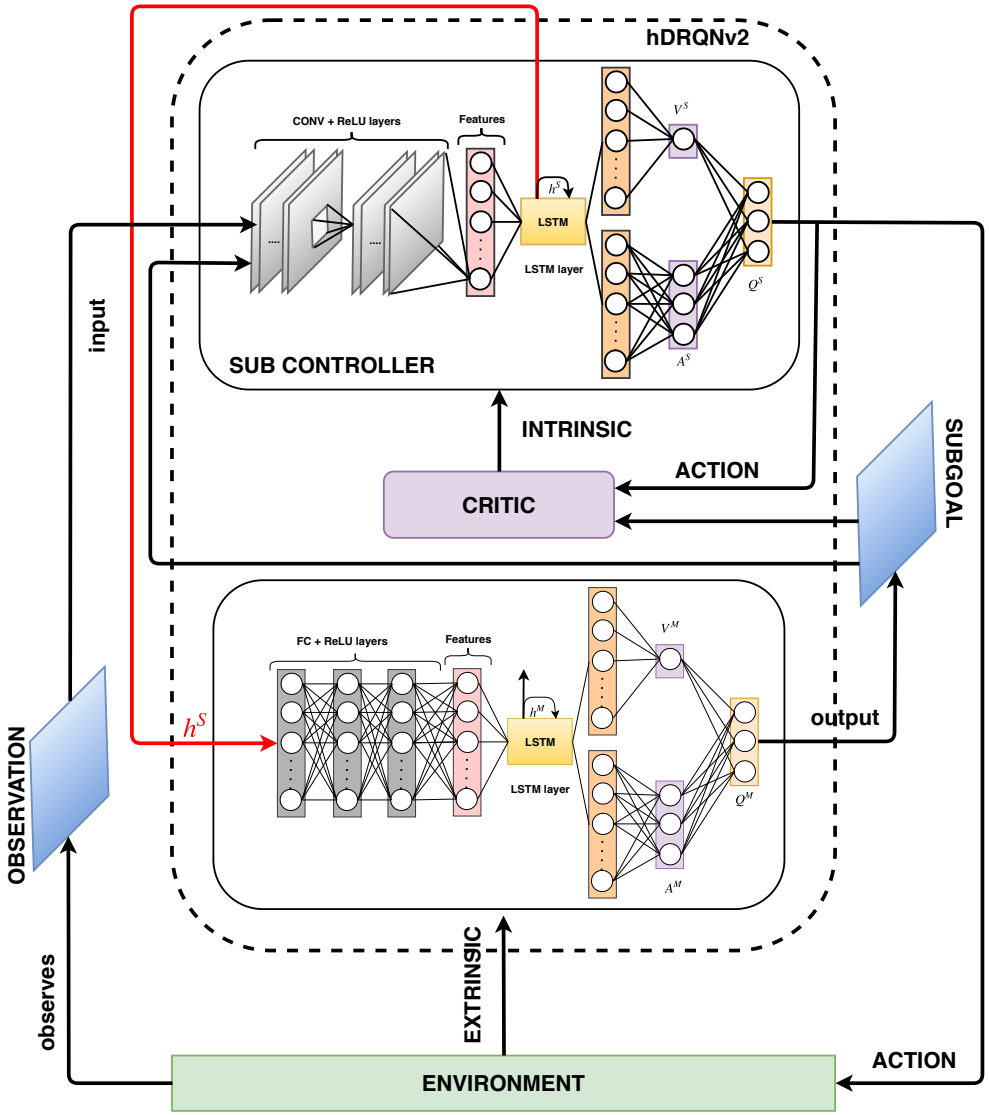
META:

- Input: Observation o
- Feature extraction: 4 CONV layers and ReLU layers.
- LSTM is integrated in front of the features.
- The output of LSTM is put into Dueling network
- Output: Q subgoal values $Q^M(o, g)$

SUB:

- Input: Observation o and current subgoal (g)
- Other part: same as META
- Output: Q action values $Q^S(\{o, g\}, a)$

hDRQN: Framework 2



META:

- Input: hidden states from SUB h^S
- Feature extraction: Three fully connected layers and ReLU layers.
- Other part has the same architecture as META of framework 1

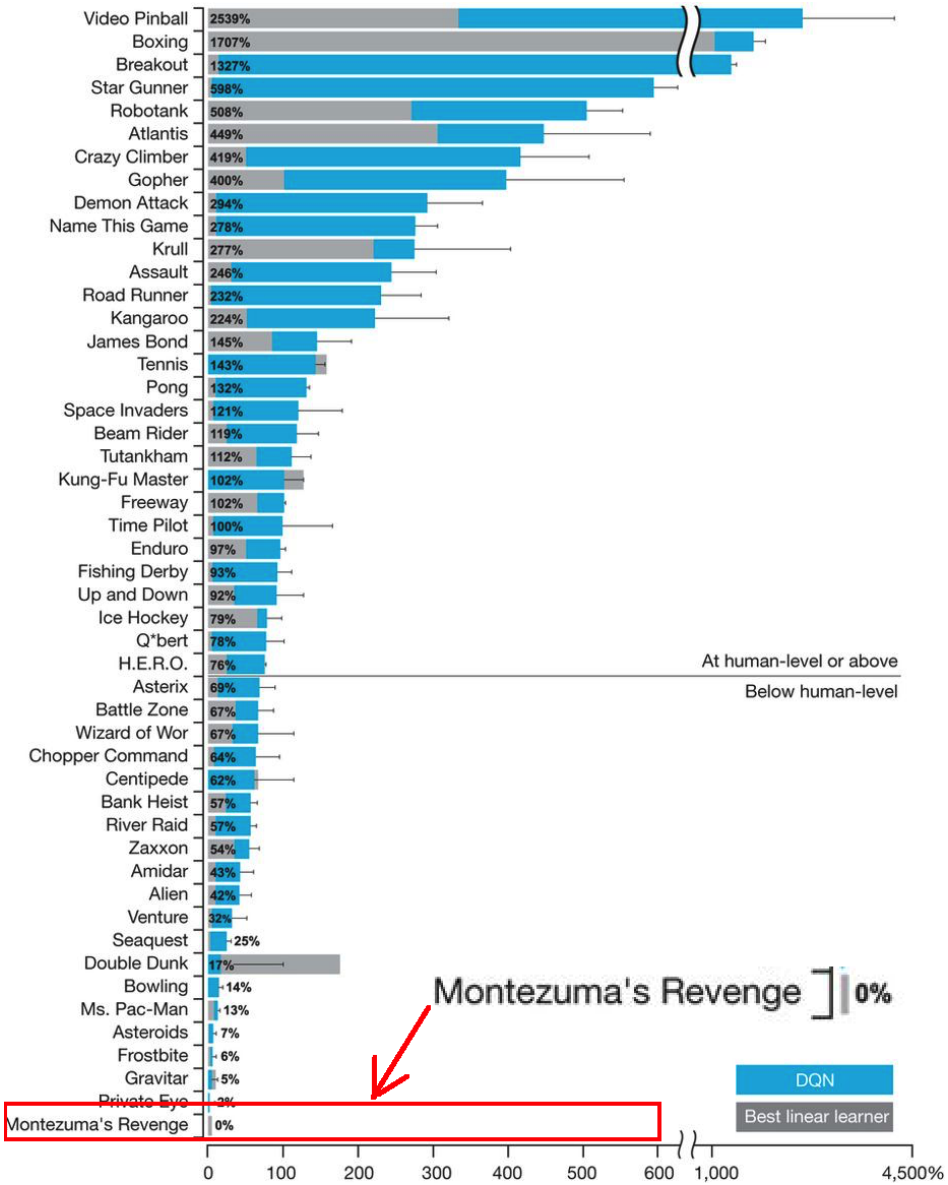
SUB:

- Same architecture as SUB of framework 1

Techniques

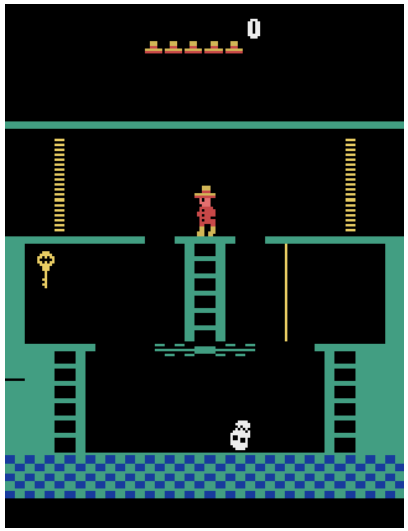
- DQN (Deep Q Learning)
- DDQN (Double Deep Q Learning)
- DRQN (Deep Recurrent Q Learning)
- hDQN (Hierarchical Deep Q Learning)
- Bootstrapped Random Updates

Montezuma's Revenge



Montezuma's Revenge

DQN as well as plain DRL algorithms fails to solve this game.



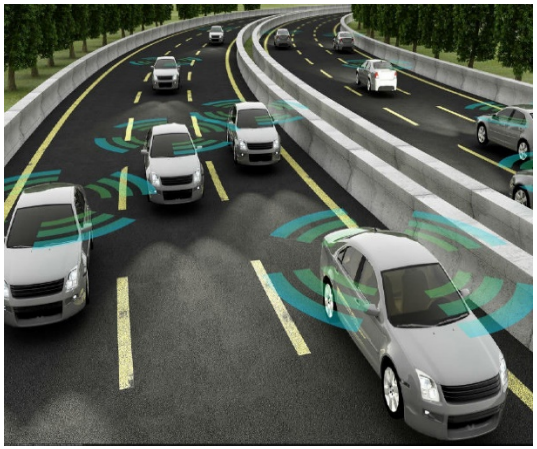
Montezuma's Revenge Game

Demo

Autonomous Bicycle/Ballbot

Context

- Autonomous vehicles become a trend for future.



(a) Autonomous Car

(b) Autonomous Drone

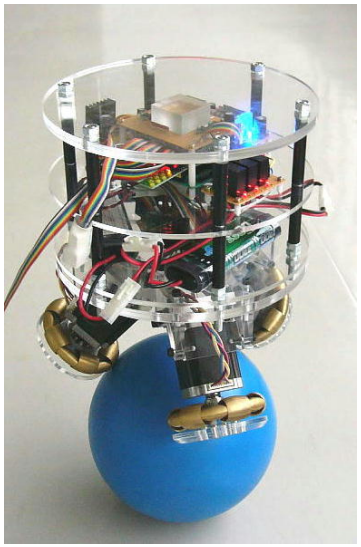
(c) Why not bicycle

Autonomous vehihcles

- Why not Bicycle/Ballbot?

Problems

- Unstable dynamics.



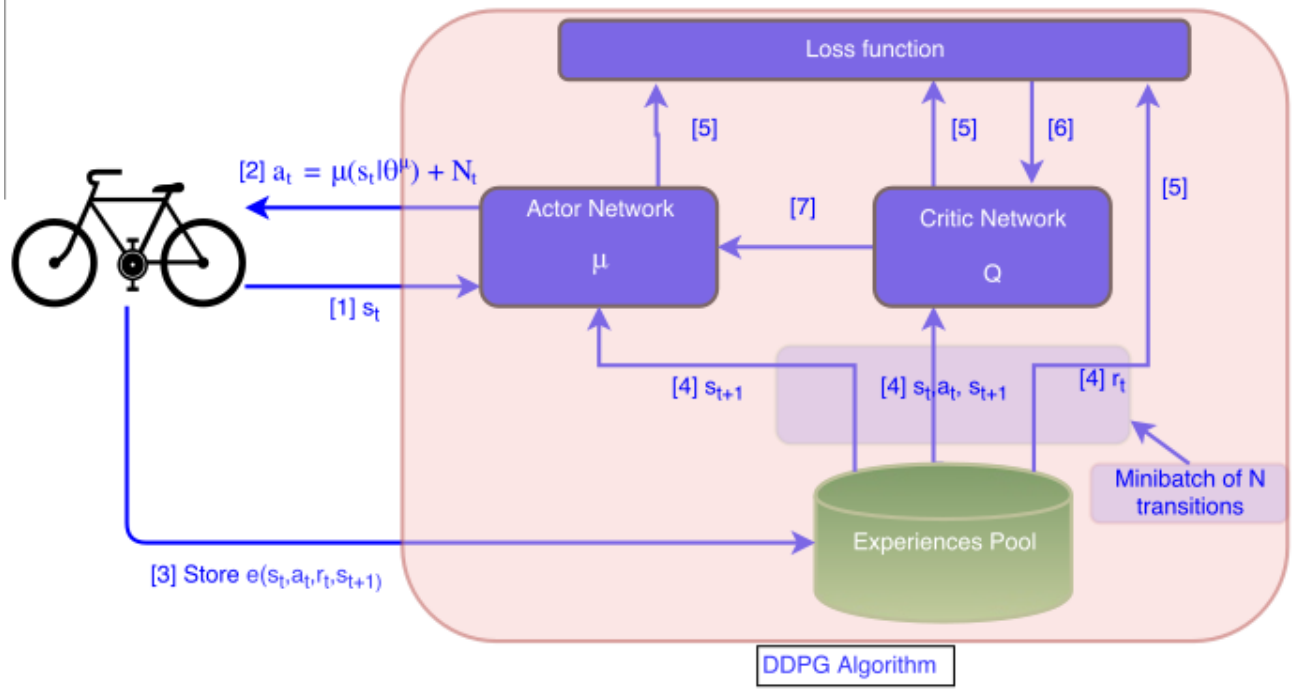
(a) Ballbot

(b) Bicycle

- Classical control methods such as PID and LQR requires a complex process to tune the gains.

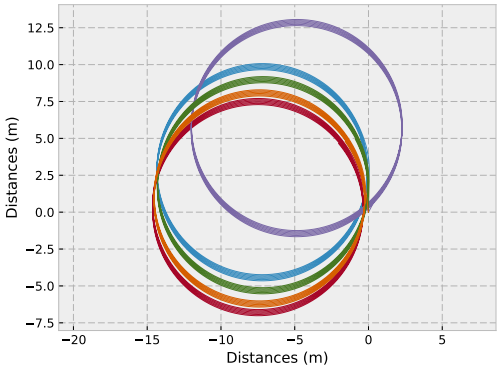
Proposed Studies

- Control Bicycle/Ballbot using deep neural networks.
- The deep neural networks are trained using state-of-the-art deep reinforcement learning
  - DDPG
  - NAF
  - PPO
  - ...

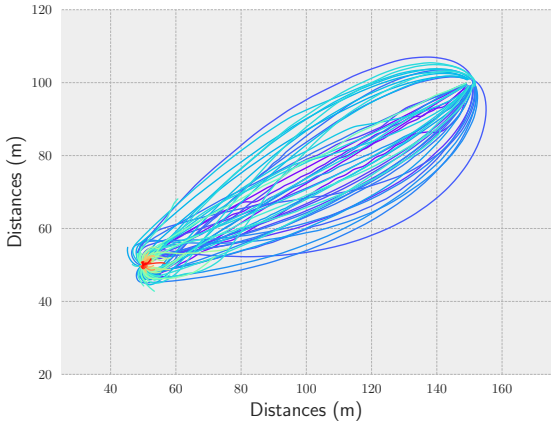


Trajectories

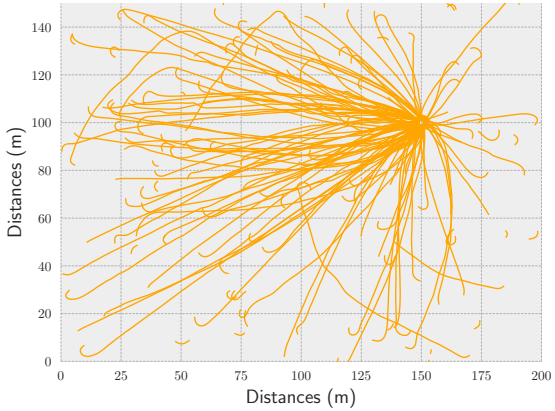
- Bicycle Demo Ballbot Demo



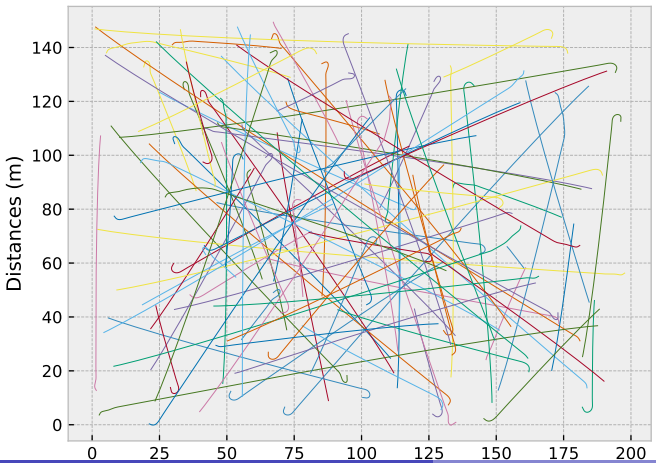
(c)

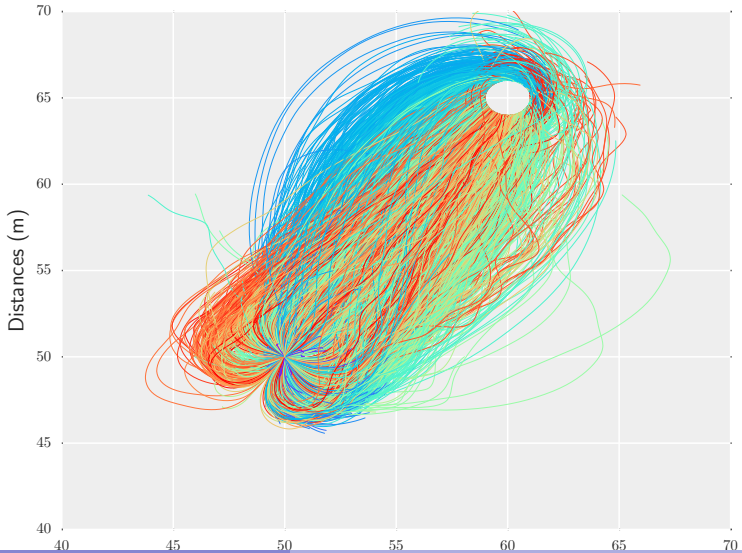


(d)



(e)





Thank You!