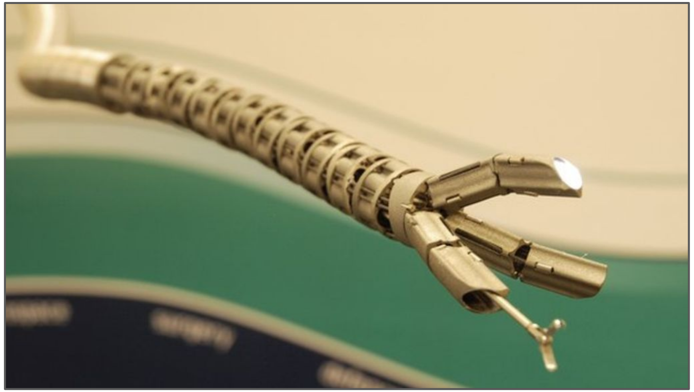
Building Locomotion Policies Using Online Reinforcement Learning

Group 12: Jared Buchanan, Steven Tran, Donald Dean
Advisor: Michael Yip, Ph.D - Computer Science and Engineering

Background: Snake Robotics



Problem Statement

How do we control a complex robot optimally without an explicit dynamical model?

Reinforcement Learning (RL)

State (s): variables describing the current system's orientation, location, speed, etc.

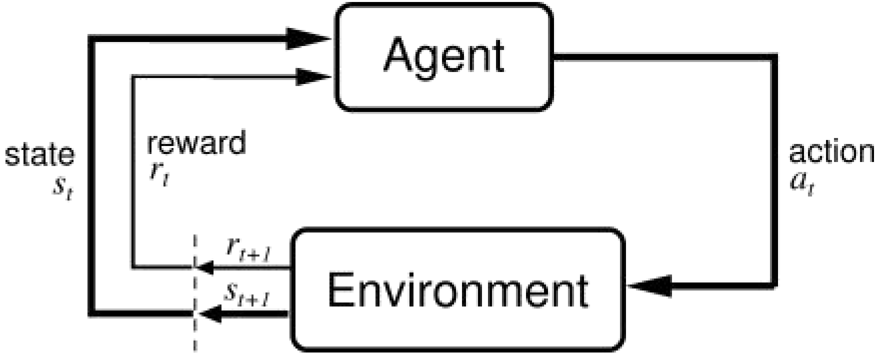
Action (a): something the agent does to move to the next state.

Policy (π(s) = a): a mapping of states to actions to be taken.

State-Action Space: all possible combinations of state and actions.

Reward Function: a user-defined function that provides a scalar feedback informing the agent how good an action choice was.

RL Goal: find policy that maximizes cumulative reward.

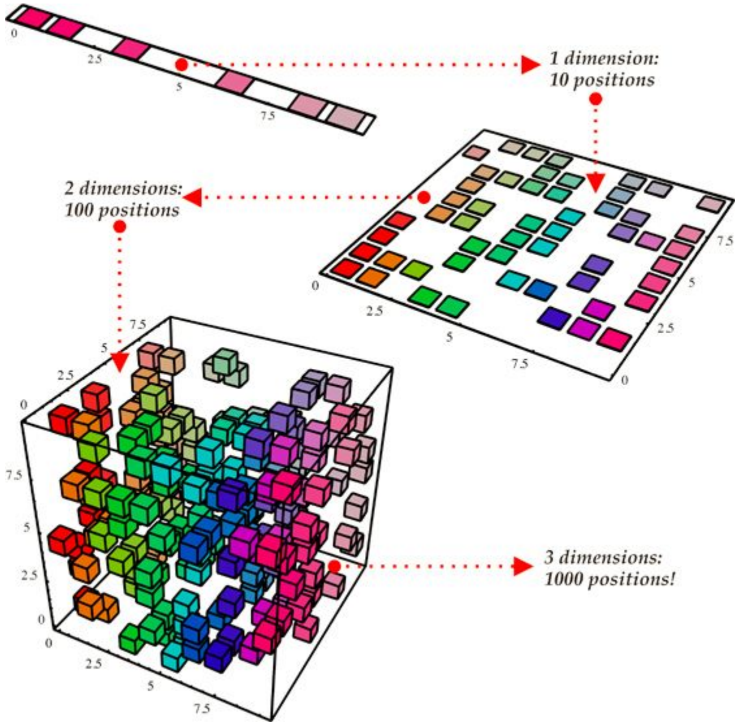


Curse of Dimensionality

- As you add more dimensions to your state representation, there is a combinatorial increase in you state-action space size.

- Computation becomes combinatorially more difficult as well.

- $10^1 > 10^2 > 10^3 > ....$



Design Criteria + Decision Matrix

Design Criteria

- Low Dependency and User Input
- High Quality Locomotion Policy Output
- Low Convergence and Runtime
- Low Software Development Cost

Decision Matrix

	Dependency on user input	Quality of the locomotion policy	Time to converge to near optimal policy	Computation power needed to run in real time	Development Cost	Total
Weights	0.2	0.4	0.15	0.15	0.1	1
Policy Iteration (Model-based)	5	7	7	10	6	6.95
Q-learning (Model-free)	10	7	9	8	10	8.35
Hard coded Algorithm	0	2	3	10	2	2.95

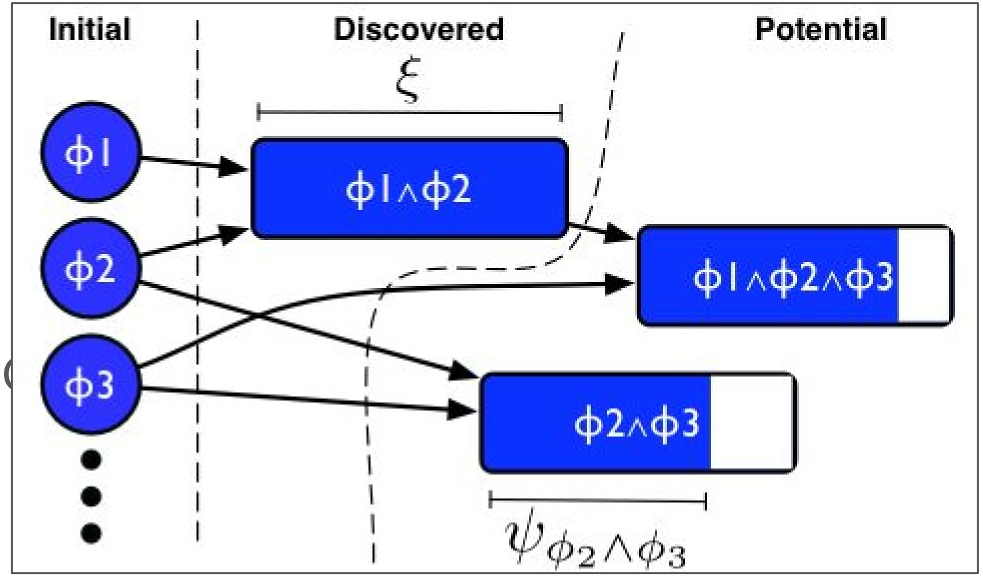
Final Solution (part 1)

Q-Learning: model-free RL technique

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Take action a, observe r, s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha\big[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\big]$
    $s \leftarrow s'$;
  until s is terminal

Final Solution (part 2)

Incremental Feature Dependency Discovery (iFDD)

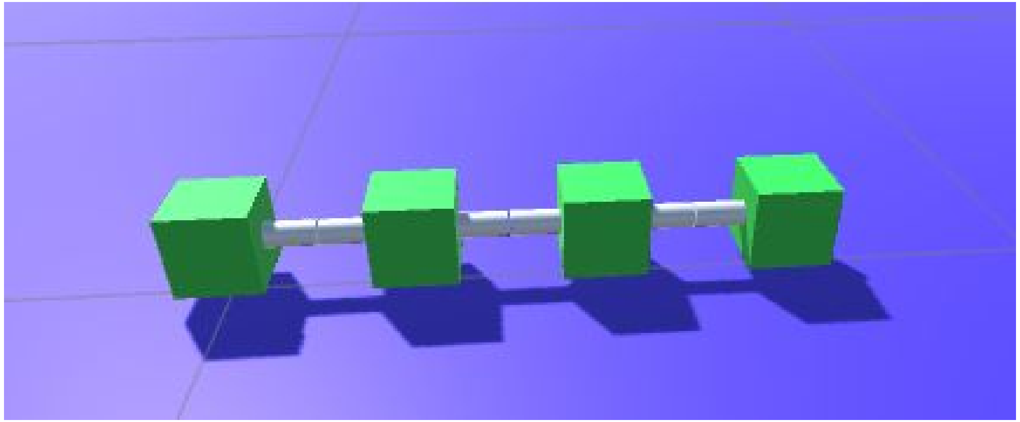


$$Q(s,a) = \sum_i x_i \Phi_i(s,a)$$

W = weight
Φ = binary feature

Snake Model and State-Action Representation

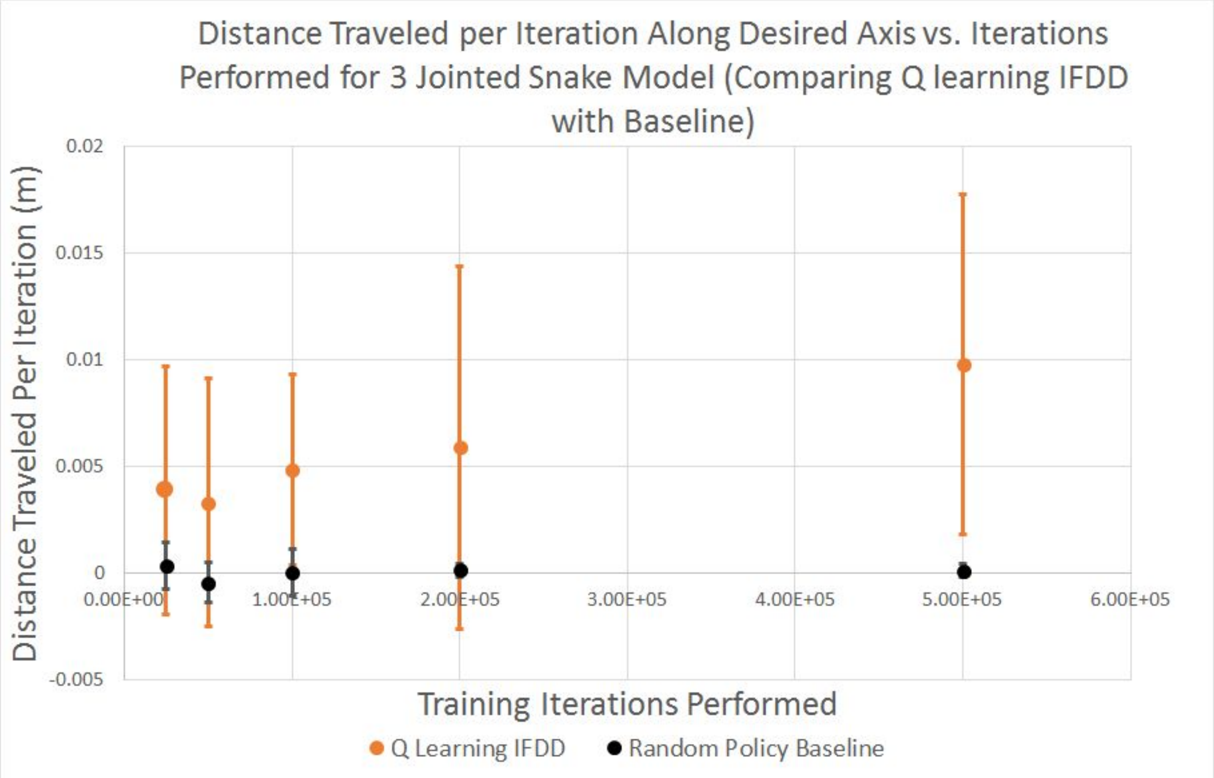
Number of Joints	State-Action Space Size Calculation	State-Action Space Size (i.e # state-action combos)
N	(7)^N * (10)^3 * (3)^N	-
2	(7)^2 * (10)^3 * (3)^2	441,000
3	(7)^3 * (10)^3 * (3)^3	9,261,000
4	(7)^4 * (10)^3 * (3)^4	194,481,000



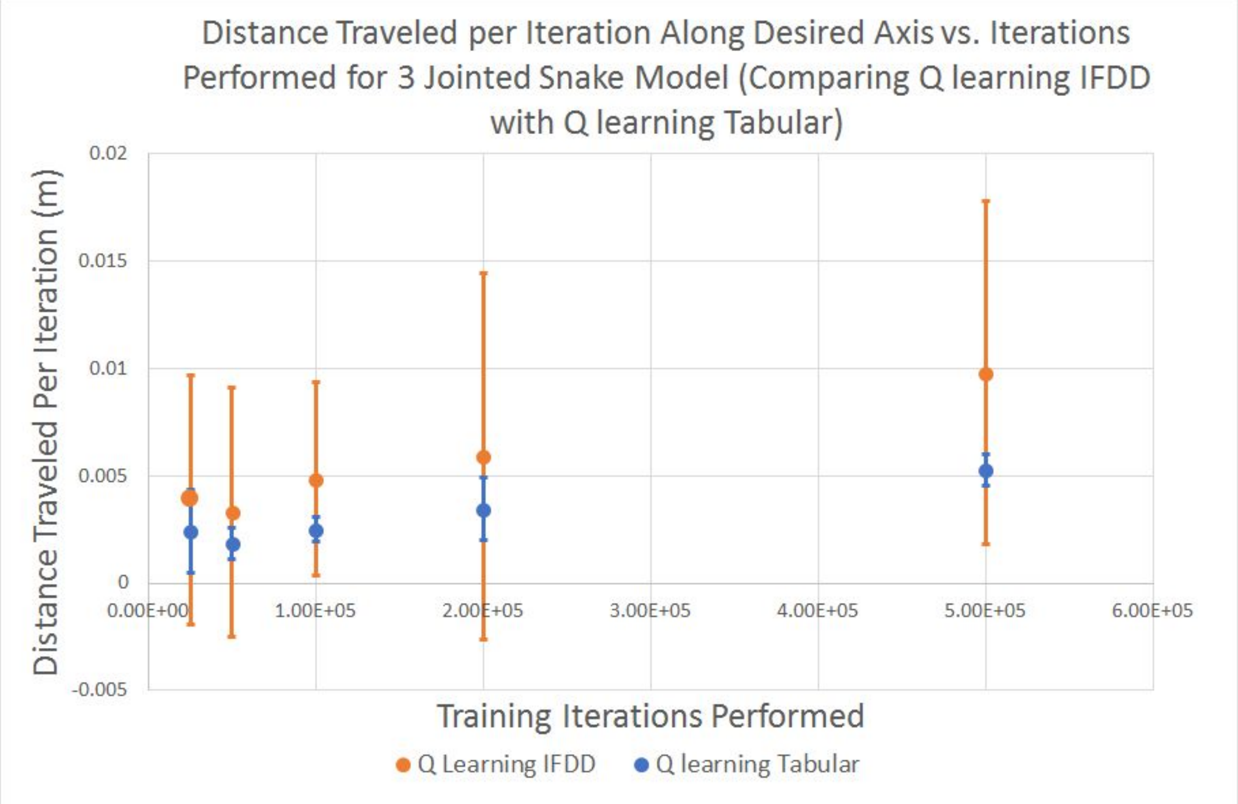
Software Implementation

Software: physics environment and C# scripting

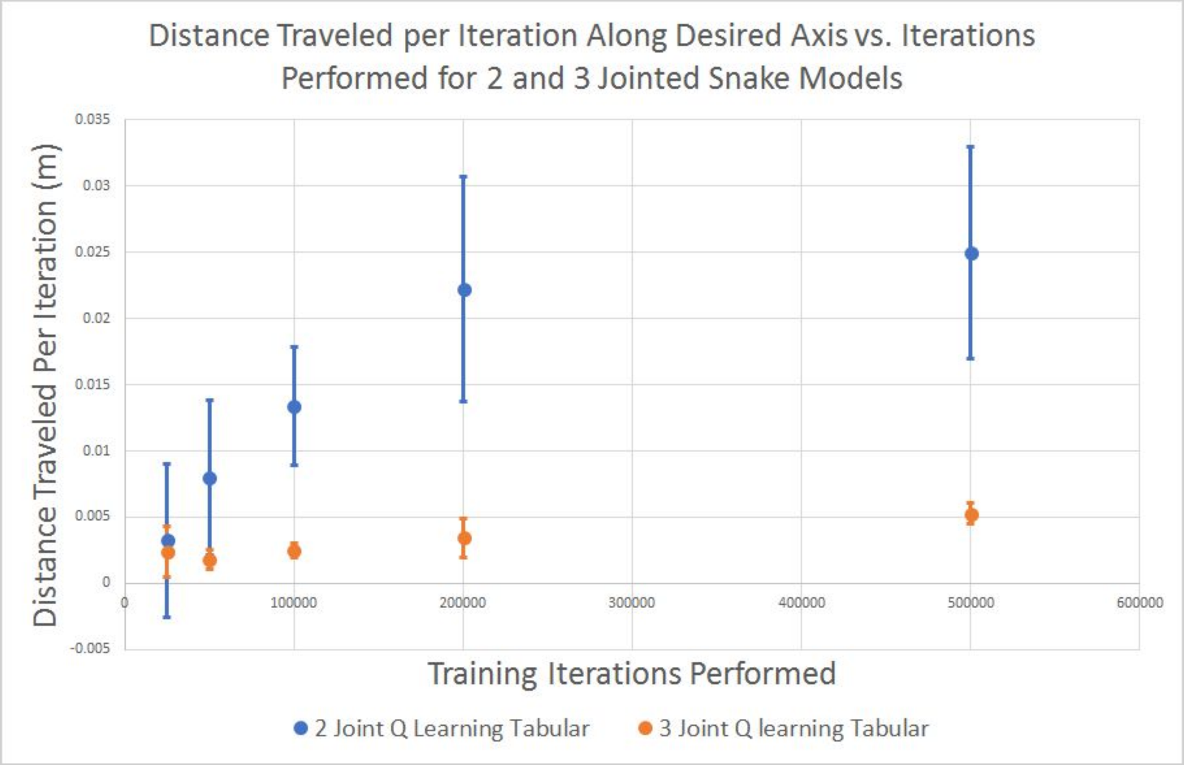
Results: Q-Learning w/ IFDD vs. Random Policy



Results: IFDD improves Q-Learning



Results: Curse of Dimensionality



Conclusion

- Q-learning with IFDD algorithm produced a policy that learns from experience.
- Can be applied to any jointed robotic model
- Feasibility test and pre-training method
- Limitations:
  - curse of dimensionality
    - Tractability
    - Environment
    - Tasks

Future Directions

- Deep Q-Learning
- Neural Networks
- Test hyperparameters such as **discount rate**, **learning rate,** and **relevance threshold**
- Increase the number of joints
- Test different robotic configurations

Cost of Project

Item	**Cost (relevant information)**
Personal Laptop	Free (through existing ownership)
Unity	Free
Labor	$7500 (250 hours at $30 per hour)

Cost of Next Phase

Item	Cost
Labor	$7500 (250 hours at $30 per hour)
Amazon Cloud Reserved Server	$1600 (16 servers for 50 hours at $2 per hour)
Luxury Highrise Apartment	$12000 (3 months at $4000/month)

Thank You

Special thanks to UCSD Bioengineering, UCSD Computer Science and Engineering, Dr. Bruce Wheeler, Dr. Michael Yip, Teryn Johnson, and Ismael Munoz!

UC San Diego

Bioengineering

UCSDCSE
Computer Science and Engineering